KallistiOS

##version##

Generated by Doxygen 1.9.8

1 KallistiOS

KallistiOS

Independent SDK for the Sega Dreamcast
Explore the docs »

1.0.1 Overview

KOS is an unofficial development kit for the SEGA Dreamcast game console with some support for the NAOMI and NAOMI 2 arcade boards.

KOS was developed from scratch over the internet by a group of free software developers and has no relation to the official Sega Katana or Microsoft Windows CE Dreamcast development kits. This has allowed it to fuel a thriving Dreamcast homebrew scene, powering many commercial releases for the platform over the years. It supports a signficiant portion of the Dreamcast's hardware capabilities and a wide variety of peripherals, accessories, and add-ons for the console, including custom hardware modifications that have been created by the scene.

Despite the console's age, KOS offers an extremely modern, programmer-friendly development environment. Using the latest GCC toolchain, it supports the entirety of C17 and C++20 including their standard libraries, along with support for portions of C23, C++23, Objective-C, and various POSIX APIs. Additionally, KOS-ports offers a rich set of add-on libraries such as SDL, OpenGL, OpenAL, and Lua for the platform.

1.0.2 Features

Core Functionality

- Concurrency with Kernel Threads, C11 Threads, C++11 `std::thread`, POSIX threads
- Virtual Filesystem Abstraction
- IPv4/IPv6 Network Stack
- Dynamically Loaded Libraries and Modules
- GDB Debugger Support

Dreamcast Hardware Support

- GD-ROM Optical Drive
- Low-level 3D PowerVR Graphics
- SH4 ASM-Optimized Math Routines
- SH4 SCIF Serial I/O
- DMA Controller
- Flashrom Filesystem
- AICA SPU Sound Processor Driver
- Cache and Store Queue Management
- Timer Peripherals, Real-Time Clock, Watchdog Timer
- Performance Counters
- MMU Management
- BIOS Font Rendering

Peripherals and Accessory Support

- Controller, ASCII Pad
- Arcade Stick, Twin Stick, Mission Stick
- Keyboard
- Mouse
- Visual Memory Unit
- Puru Puru Vibration Pack
- Seaman Microphone
- Dreameye Webcam
- Lightgun
- Racing Wheel
- Fishing Rod
- Samba De Amigo Maracas
- Dance Mat
- Dial-up Modem
- Broadband Adapter
- LAN Adapter
- VGA Adapter
- SD Card Reader

Hardware Modification Support

- IDE Hard Drive
- 32MB RAM Upgrade
- Custom BIOS Flashroms

1.0.3 Getting Started

A beginner's guide to development for the Sega Dreamcast along with detailed instructions for installing KOS and the required toolchains can be found on the Dreamcast Wiki. Additional documentation can be found in the docs folder.

1.0.4 Examples

Once you've set up the environment and are ready to begin developing, a good place to start learning is the examples directory, which provides demos for the various KOS APIs and for interacting with the Dreamcast's hardware. Examples include:

- Hello World
- Console Input/Output
- Assertions, stacktraces, threading
- Drawing directly to the framebuffer
- Rendering with OpenGL
- Rendering with KGL
- Rendering with KOS PVR API
- Texturing with libPNG
- Bump maps, modifier volumes, render-to-texture PVR effects
- Audio playback on the ARM SPU
- Audio playback using SDL Audio
- Audio playback using OGG, MP3, and CDDA
- Querying controller input
- Querying keyboard input
- Querying mouse input
- Querying lightgun input
- Accessing the VMU filesystem
- Accessing the SD card filesystem
- Networking with the modem, broadband adapter, and LAN adapter
- Taking pictures with the DreamEye webcam
- Reading and Writing to/from ATA devices
- Testing 32MB RAM hardware mod
- Interactive Lua interpreter terminal

1.0.5 Resources

`DCEmulation Forums`: Goldmine of Dreamcast development information and history
`Dreamcast Wiki`: Large collection of tutorials and articles for beginners
`Simulant Discord Chat`: Home to the official Discord channel of KOS
IRC Channel: irc.libera.chat `#dreamcastdev`

2 Deprecated List

Global cond_create () __depr("Use cond_init or COND_INITIALIZER.")

This function is formally deprecated and should not be used in new code. Instead you should use either the static initializer or the cond_init() function.

Global INIT_THD_PREEMPT

Already default mode

Global mutex_create (void) __depr("Use mutex_init or an initializer.")

This function allocates and initializes a new mutex for use. This function will always create mutexes of the type MUTEX_TYPE_NORMAL.

File recursive_lock.h

These are now just wrappers around the MUTEX_TYPE_RECURSIVE that is now provided and will be removed at some point in the future. Please update your code to use that type instead.

Global rlock_create (void) __depr("Use mutexes instead.")

This function allocates a new recurisve lock that is initially not locked.

Global rlock_destroy (recursive_lock_t *l) __depr("Use mutexes instead.")

This function cleans up a recursive lock. It is an error to attempt to destroy a locked recursive lock.

Global rlock_is_locked (recursive_lock_t *l) __depr("Use mutexes instead.")

This function checks whether or not a lock is currently held by any thread, including the calling thread. Note that this is **NOT** a safe way to check if a lock *will* be held by the time you get around to locking it.

Global rlock_lock (recursive_lock_t *l) __depr("Use mutexes instead.")

This function attempts to lock the requested lock, and if it cannot it will block until that is possible.

Global rlock_lock_timed (recursive_lock_t *l, int timeout) __depr("Use mutexes instead.")

This function attempts to lock the requested lock, and if it cannot it will block until either it is possible to acquire the lock or timeout milliseconds have elapsed.

Global rlock_trylock (recursive_lock_t *l) __depr("Use mutexes instead.")

This function attempts to lock a recursive lock without blocking. This function, unlike rlock_lock and rlock_lock_timed is safe to call inside an interrupt.

Global rlock_unlock (recursive_lock_t *l) __depr("Use mutexes instead.")

This function releases the lock one time from the current thread.

Global rwsem_create (void) __depr("Use rwsem_init or RWSEM_INITIALIZER.")

This function is formally deprecated, and should not be used in newly written code. Instead, please use rwsem_init().

Global sem_create (int value) __depr("Use sem_init or SEM_INITIALIZER.")

This function is formally deprecated. Please update your code to use sem_init() or static initialization with SEM_↩ INITIALIZER instead.

Global thd_get_mode (void) __deprecated

This is now deprecated.

Global THD_MODE_COOP

Global thd_set_mode (int mode) __deprecated

This is now deprecated.

3 Todo List

Global vmu_set_icon (const char *vmu_icon)

Prevent this routine from broadcasting to rear VMUs.

4 Topic Index

4.1 Topics

Here is a list of all topics with brief descriptions:

5 Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

7 Topic Documentation

7.1 ASIC IRQ levels

Macros

- #define ASIC_IRQ9 0

  IRQ level 9.
- #define ASIC_IRQB 1

  IRQ level B (11)
- #define ASIC_IRQD 2

IRQ level D (13)

- #define ASIC_IRQ_MAX 3

  Don't take irqs from here up.

- #define ASIC_IRQ_DEFAULT ASIC_IRQ9

  Pick an IRQ level for me!

7.1.1 Detailed Description

You can pick one at hook time, or don't choose anything and the default will be used instead.

7.1.2 Macro Definition Documentation

ASIC_IRQ9

```
#define ASIC_IRQ9 0
```

IRQ level 9.

ASIC_IRQ_DEFAULT

```
#define ASIC_IRQ_DEFAULT ASIC_IRQ9
```

Pick an IRQ level for me!

ASIC_IRQ_MAX

```
#define ASIC_IRQ_MAX 3
```

Don't take irqs from here up.

ASIC_IRQB

```
#define ASIC_IRQB 1
```

IRQ level B (11)

ASIC_IRQD

```
#define ASIC_IRQD 2
```

IRQ level D (13)

7.2 ASIC event codes

Modules

- Event codes for G2 bus DMA
- Event codes for the GD controller
- Event codes for the Maple controller
- Event codes for the PVR chip
- Event codes for the SPU
- Event codes for the external port

7.2.1 Detailed Description

7.2.2 Event codes for G2 bus DMA

Macros

- #define ASIC_EVT_G2_DMA0 0x000f
  G2 DMA channel 0 complete.
- #define ASIC_EVT_G2_DMA1 0x0010
  G2 DMA channel 1 complete.
- #define ASIC_EVT_G2_DMA2 0x0011
  G2 DMA channel 2 complete.
- #define ASIC_EVT_G2_DMA3 0x0012
  G2 DMA channel 3 complete.

7.2.2.1 Detailed Description

These are events that G2 bus DMA generates that can be hooked.

7.2.2.2 Macro Definition Documentation

ASIC_EVT_G2_DMA0

```
#define ASIC_EVT_G2_DMA0 0x000f
```

G2 DMA channel 0 complete.

ASIC_EVT_G2_DMA1

```
#define ASIC_EVT_G2_DMA1 0x0010
```

G2 DMA channel 1 complete.

ASIC_EVT_G2_DMA2

```
#define ASIC_EVT_G2_DMA2 0x0011
```

G2 DMA channel 2 complete.

ASIC_EVT_G2_DMA3

```
#define ASIC_EVT_G2_DMA3 0x0012
```

G2 DMA channel 3 complete.

7.2.3 Event codes for the GD controller

Macros

- #define ASIC_EVT_GD_COMMAND 0x0100
  GD-Rom Command Status.
- #define ASIC_EVT_GD_DMA 0x000e
  GD-Rom DMA complete.
- #define ASIC_EVT_GD_DMA_OVERRUN 0x020d
  GD-Rom DMA overrun.
- #define ASIC_EVT_GD_DMA_ILLADDR 0x020c
  GD-Rom DMA illegal address.

7.2.3.1 Detailed Description

These are events that the GD-ROM drive generates that can be hooked.

7.2.3.2 Macro Definition Documentation

ASIC_EVT_GD_COMMAND

```
#define ASIC_EVT_GD_COMMAND 0x0100
```

GD-Rom Command Status.

ASIC_EVT_GD_DMA

```
#define ASIC_EVT_GD_DMA 0x000e
```

GD-Rom DMA complete.

ASIC_EVT_GD_DMA_ILLADDR

```
#define ASIC_EVT_GD_DMA_ILLADDR 0x020c
```

GD-Rom DMA illegal address.

ASIC_EVT_GD_DMA_OVERRUN

```
#define ASIC_EVT_GD_DMA_OVERRUN 0x020d
```

GD-Rom DMA overrun.

7.2.4 Event codes for the Maple controller

Macros

- #define ASIC_EVT_MAPLE_DMA 0x000c
  Maple DMA complete.
- #define ASIC_EVT_MAPLE_ERROR 0x000d
  Maple error (?)

7.2.4.1 Detailed Description

These are events that Maple generates that can be hooked.

7.2.4.2 Macro Definition Documentation

ASIC_EVT_MAPLE_DMA

```
#define ASIC_EVT_MAPLE_DMA 0x000c
```

Maple DMA complete.

ASIC_EVT_MAPLE_ERROR

```
#define ASIC_EVT_MAPLE_ERROR 0x000d
```

Maple error (?)

7.2.5 Event codes for the PVR chip

Macros

- #define ASIC_EVT_PVR_RENDERDONE_VIDEO 0x0000

  Video render stage completed.
- #define ASIC_EVT_PVR_RENDERDONE_ISP 0x0001

  ISP render stage completed.
- #define ASIC_EVT_PVR_RENDERDONE_TSP 0x0002

  TSP render stage completed.
- #define ASIC_EVT_PVR_VBLANK_BEGIN 0x0003

  VBLANK begin interrupt.
- #define ASIC_EVT_PVR_VBLANK_END 0x0004

  VBLANK end interrupt.
- #define ASIC_EVT_PVR_HBLANK_BEGIN 0x0005

  HBLANK begin interrupt.
- #define ASIC_EVT_PVR_YUV_DONE 0x0007

  YUV completed.
- #define ASIC_EVT_PVR_OPAQUEDONE 0x0007

  Opaque list completed.
- #define ASIC_EVT_PVR_OPAQUEMODDONE 0x0008

  Opaque modifiers completed.
- #define ASIC_EVT_PVR_TRANSDONE 0x0009

  Transparent list completed.
- #define ASIC_EVT_PVR_TRANSMODDONE 0x000a

  Transparent modifiers completed.
- #define ASIC_EVT_PVR_DMA 0x0013

  PVR DMA complete.
- #define ASIC_EVT_PVR_PTDONE 0x0015

  Punch-thrus completed.
- #define ASIC_EVT_PVR_ISP_OUTOFMEM 0x0200

  ISP out of memory.
- #define ASIC_EVT_PVR_STRIP_HALT 0x0201

  Halt due to strip buffer error.
- #define ASIC_EVT_PVR_PARAM_OUTOFMEM 0x0202

  Param out of memory.
- #define ASIC_EVT_PVR_OPB_OUTOFMEM 0x0203

  OPB went past PVR_TA_OPB_END.
- #define ASIC_EVT_PVR_TA_INPUT_ERR 0x0204

  Vertex input error.
- #define ASIC_EVT_PVR_TA_INPUT_OVERFLOW 0x0205

  Vertex input overflowed queue.

7.2.5.1 Detailed Description

These are events that the PVR itself generates that can be hooked.

7.2.5.2 Macro Definition Documentation

ASIC_EVT_PVR_DMA

```
#define ASIC_EVT_PVR_DMA 0x0013
```

PVR DMA complete.

ASIC_EVT_PVR_HBLANK_BEGIN

```
#define ASIC_EVT_PVR_HBLANK_BEGIN 0x0005
```

HBLANK begin interrupt.

ASIC_EVT_PVR_ISP_OUTOFMEM

```
#define ASIC_EVT_PVR_ISP_OUTOFMEM 0x0200
```

ISP out of memory.

ASIC_EVT_PVR_OPAQUEDONE

```
#define ASIC_EVT_PVR_OPAQUEDONE 0x0007
```

Opaque list completed.

ASIC_EVT_PVR_OPAQUEMODDONE

```
#define ASIC_EVT_PVR_OPAQUEMODDONE 0x0008
```

Opaque modifiers completed.

ASIC_EVT_PVR_OPB_OUTOFMEM

```
#define ASIC_EVT_PVR_OPB_OUTOFMEM 0x0203
```

OPB went past PVR_TA_OPB_END.

ASIC_EVT_PVR_PARAM_OUTOFMEM

```
#define ASIC_EVT_PVR_PARAM_OUTOFMEM 0x0202
```

Param out of memory.

ASIC_EVT_PVR_PTDONE

```
#define ASIC_EVT_PVR_PTDONE 0x0015
```

Punch-thrus completed.

ASIC_EVT_PVR_RENDERDONE_ISP

```
#define ASIC_EVT_PVR_RENDERDONE_ISP 0x0001
```

ISP render stage completed.

ASIC_EVT_PVR_RENDERDONE_TSP

```
#define ASIC_EVT_PVR_RENDERDONE_TSP 0x0002
```

TSP render stage completed.

ASIC_EVT_PVR_RENDERDONE_VIDEO

```
#define ASIC_EVT_PVR_RENDERDONE_VIDEO 0x0000
```

Video render stage completed.

ASIC_EVT_PVR_STRIP_HALT

```
#define ASIC_EVT_PVR_STRIP_HALT 0x0201
```

Halt due to strip buffer error.

ASIC_EVT_PVR_TA_INPUT_ERR

```
#define ASIC_EVT_PVR_TA_INPUT_ERR 0x0204
```

Vertex input error.

ASIC_EVT_PVR_TA_INPUT_OVERFLOW

```
#define ASIC_EVT_PVR_TA_INPUT_OVERFLOW 0x0205
```

Vertex input overflowed queue.

ASIC_EVT_PVR_TRANSDONE

```
#define ASIC_EVT_PVR_TRANSDONE 0x0009
```

Transparent list completed.

ASIC_EVT_PVR_TRANSMODDONE

```
#define ASIC_EVT_PVR_TRANSMODDONE 0x000a
```

Transparent modifiers completed.

ASIC_EVT_PVR_VBLANK_BEGIN

```
#define ASIC_EVT_PVR_VBLANK_BEGIN 0x0003
```

VBLANK begin interrupt.

ASIC_EVT_PVR_VBLANK_END

```
#define ASIC_EVT_PVR_VBLANK_END 0x0004
```

VBLANK end interrupt.

ASIC_EVT_PVR_YUV_DONE

```
#define ASIC_EVT_PVR_YUV_DONE 0x0007
```

YUV completed.

7.2.6 Event codes for the SPU

Macros

- #define ASIC_EVT_SPU_DMA 0x000f
  SPU (G2 channel 0) DMA complete.
- #define ASIC_EVT_SPU_IRQ 0x0101
  SPU interrupt.

7.2.6.1 Detailed Description

These are events that the SPU (AICA) generates that can be hooked.

7.2.6.2 Macro Definition Documentation

ASIC_EVT_SPU_DMA

```
#define ASIC_EVT_SPU_DMA 0x000f
```

SPU (G2 channel 0) DMA complete.

ASIC_EVT_SPU_IRQ

```
#define ASIC_EVT_SPU_IRQ 0x0101
```

SPU interrupt.

7.2.7 Event codes for the external port

Macros

- #define ASIC_EVT_EXP_8BIT 0x0102
  Modem / Lan Adapter.
- #define ASIC_EVT_EXP_PCI 0x0103
  BBA IRQ.

7.2.7.1 Detailed Description

These are events that external devices generate that can be hooked.

7.2.7.2 Macro Definition Documentation

ASIC_EVT_EXP_8BIT

```
#define ASIC_EVT_EXP_8BIT 0x0102
```

Modem / Lan Adapter.

ASIC_EVT_EXP_PCI

```
#define ASIC_EVT_EXP_PCI 0x0103
```

BBA IRQ.

7.3 ASIC registers

Macros

- #define ASIC_ACK_A 0xa05f6900
  IRQD ACK register.
- #define ASIC_ACK_B 0xa05f6904
  IRQB ACK register.
- #define ASIC_ACK_C 0xa05f6908
  IRQ9 ACK register.
- #define ASIC_IRQD_A 0xa05f6910
  IRQD first register.
- #define ASIC_IRQD_B 0xa05f6914
  IRQD second register.
- #define ASIC_IRQD_C 0xa05f6918
  IRQD third register.
- #define ASIC_IRQB_A 0xa05f6920
  IRQB first register.
- #define ASIC_IRQB_B 0xa05f6924
  IRQB second register.
- #define ASIC_IRQB_C 0xa05f6928
  IRQB third register.
- #define ASIC_IRQ9_A 0xa05f6930
  IRQ9 first register.
- #define ASIC_IRQ9_B 0xa05f6934
  IRQ9 second register.
- #define ASIC_IRQ9_C 0xa05f6938
  IRQ9 third register.

7.3.1 Detailed Description

These are the locations in memory where the ASIC registers sit.

7.3.2 Macro Definition Documentation

ASIC_ACK_A

```
#define ASIC_ACK_A 0xa05f6900
```

IRQD ACK register.

ASIC_ACK_B

```
#define ASIC_ACK_B 0xa05f6904
```

IRQB ACK register.

ASIC_ACK_C

```
#define ASIC_ACK_C 0xa05f6908
```

IRQ9 ACK register.

ASIC_IRQ9_A

```
#define ASIC_IRQ9_A 0xa05f6930
```

IRQ9 first register.

ASIC_IRQ9_B

```
#define ASIC_IRQ9_B 0xa05f6934
```

IRQ9 second register.

ASIC_IRQ9_C

```
#define ASIC_IRQ9_C 0xa05f6938
```

IRQ9 third register.

ASIC_IRQB_A

```
#define ASIC_IRQB_A 0xa05f6920
```

IRQB first register.

ASIC_IRQB_B

```
#define ASIC_IRQB_B 0xa05f6924
```

IRQB second register.

ASIC_IRQB_C

```
#define ASIC_IRQB_C 0xa05f6928
```

IRQB third register.

ASIC_IRQD_A

```
#define ASIC_IRQD_A 0xa05f6910
```

IRQD first register.

ASIC_IRQD_B

```
#define ASIC_IRQD_B 0xa05f6914
```

IRQD second register.

ASIC_IRQD_C

```
#define ASIC_IRQD_C 0xa05f6918
```

IRQD third register.

7.4 ATA device definitions

Macros

- #define G1_ATA_MASTER 0x00

  ATA master device.
- #define G1_ATA_MASTER_ALT 0x90

  ATA master device (compatible with old drives).
- #define G1_ATA_SLAVE 0xB0

  ATA slave device.
- #define G1_ATA_LBA_MODE 0x40

  Select LBA addressing mode.

7.4.1 Detailed Description

The constants here represent the valid values that can be set as the active device on the ATA bus. You should pass one of these values to the g1_ata_select_device() function to select the appropriate device.

Note

Many times, the value returned by the g1_ata_select_device() function will have other bits set than the constants below. You should AND the value returned from that function with 0x10 if you really need to know what device is actually selected. If the value returned is true when ANDed with 0x10, then the slave device was selected, otherwise the master device was. The other bits are either command-specific or are reserved for compatibility sake.

7.4.2 Macro Definition Documentation

G1_ATA_LBA_MODE

```
#define G1_ATA_LBA_MODE 0x40
```

Select LBA addressing mode.

OR this constant with one of the device constants (G1_ATA_MASTER or G1_ATA_SLAVE) to select LBA addressing mode. The various g1_ata_* functions all do this as appropriate already, so you shouldn't have to worry about this one at all. This bit is irrelevant for packet devices.

G1_ATA_MASTER

```
#define G1_ATA_MASTER 0x00
```

ATA master device.

This constant selects the master device on the ATA bus. This is normally the GD-ROM drive.

Note

The GD-ROM really does not like the reserved bits being set in the device select register, hence why this constant doesn't select them. Some hard drives may require them, however. If you find one that does, then you should use the G1_ATA_MASTER_ALT constant to access it if it is the master device on the bus.

G1_ATA_MASTER_ALT

```
#define G1_ATA_MASTER_ALT 0x90
```

ATA master device (compatible with old drives).

This constant selects the master device on the ATA bus, with the old reserved bits set to 1. If you have a drive that predates ATA-2, then this will probably be the constant you want to access it as the master device.

Note

Do not use this constant to access the GD-ROM. It will not work. Use G1_ATA_MASTER instead.

G1_ATA_SLAVE

```
#define G1_ATA_SLAVE 0xB0
```

ATA slave device.

This constant selects the slave device on the ATA bus. This is where you would find a hard drive, if the user has an adapter installed.

7.5 Available flags for initialization

Macros

- #define INIT_DEFAULT

  Default init flags (IRQs on, preemption enabled, romdisks).
- #define INIT_NONE 0x00000000

  Don't init optional things.
- #define INIT_IRQ 0x00000001

  Enable IRQs at startup.
- #define INIT_THD_PREEMPT 0x00000002
- #define INIT_NET 0x00000004

  Enable built-in networking.
- #define INIT_MALLOCSTATS 0x00000008

  Enable malloc statistics.
- #define INIT_QUIET 0x00000010

  Disable dbgio.
- #define INIT_EXPORT 0x00000020

  Export kernel symbols.
- #define INIT_FS_ROMDISK 0x00000040

  Enable support for romdisks.

7.5.1 Detailed Description

These are the architecture-independent flags that can be specified with KOS_INIT_FLAGS.

See also

Dreamcast-specific initialization flags.

7.5.2 Macro Definition Documentation

INIT_DEFAULT

```
#define INIT_DEFAULT
```

Value:

```
(INIT_IRQ | INIT_THD_PREEMPT | INIT_FS_ROMDISK | \
INIT_DEFAULT_ARCH)
```

Default init flags (IRQs on, preemption enabled, romdisks).

INIT_EXPORT

```
#define INIT_EXPORT 0x00000020
```

Export kernel symbols.

INIT_FS_ROMDISK

```
#define INIT_FS_ROMDISK 0x00000040
```

Enable support for romdisks.

INIT_IRQ

```
#define INIT_IRQ 0x00000001
```

Enable IRQs at startup.

INIT_MALLOCSTATS

```
#define INIT_MALLOCSTATS 0x00000008
```

Enable malloc statistics.

INIT_NET

```
#define INIT_NET 0x00000004
```

Enable built-in networking.

INIT_NONE

```
#define INIT_NONE 0x00000000
```

Don't init optional things.

INIT_QUIET

```
#define INIT_QUIET 0x00000010
```

Disable dbgio.

INIT_THD_PREEMPT

```
#define INIT_THD_PREEMPT 0x00000002
```

Deprecated Already default mode

7.6 Available sizes for primitive bins

Macros

- #define PVR_BINSIZE_0 0
  0-length (disables the list)
- #define PVR_BINSIZE_8 8
  8-word (32-byte) length
- #define PVR_BINSIZE_16 16
  16-word (64-byte) length
- #define PVR_BINSIZE_32 32
  32-word (128-byte) length

7.6.1 Detailed Description

7.6.2 Macro Definition Documentation

PVR_BINSIZE_0

```
#define PVR_BINSIZE_0 0
```

0-length (disables the list)

PVR_BINSIZE_16

```
#define PVR_BINSIZE_16 16
```

16-word (64-byte) length

PVR_BINSIZE_32

```
#define PVR_BINSIZE_32 32
```

32-word (128-byte) length

PVR_BINSIZE_8

```
#define PVR_BINSIZE_8 8
```

8-word (32-byte) length

7.7 CD-ROM ATA status

Macros

- #define ATA_STAT_INTERNAL 0x00
- #define ATA_STAT_IRQ 0x01
- #define ATA_STAT_DRQ_0 0x02
- #define ATA_STAT_DRQ_1 0x03
- #define ATA_STAT_BUSY 0x04

7.7.1 Detailed Description

7.7.2 Macro Definition Documentation

ATA_STAT_BUSY

```
#define ATA_STAT_BUSY 0x04
```

ATA_STAT_DRQ_0

```
#define ATA_STAT_DRQ_0 0x02
```

ATA_STAT_DRQ_1

```
#define ATA_STAT_DRQ_1 0x03
```

ATA_STAT_INTERNAL

```
#define ATA_STAT_INTERNAL 0x00
```

ATA_STAT_IRQ

```
#define ATA_STAT_IRQ 0x01
```

7.8 CD-ROM Command Status responses

Macros

- #define FAILED -1

  Command failed.
- #define NO_ACTIVE 0

  System inactive?
- #define PROCESSING 1

  Processing command.
- #define COMPLETED 2

  Command completed successfully.
- #define STREAMING 3

  Stream type command is in progress.
- #define BUSY 4

  GD syscalls is busy.

7.8.1 Detailed Description

These are the raw values the status syscall returns.

7.8.2 Macro Definition Documentation

BUSY

```
#define BUSY 4
```

GD syscalls is busy.

COMPLETED

```
#define COMPLETED 2
```

Command completed successfully.

FAILED

```
#define FAILED -1
```

Command failed.

NO_ACTIVE

```
#define NO_ACTIVE 0
```

System inactive?

PROCESSING

```
#define PROCESSING 1
```

Processing command.

STREAMING

```
#define STREAMING 3
```

Stream type command is in progress.

7.9 CD-ROM Read Sector Mode

Macros

- #define CDROM_READ_PIO 0
  Read sector(s) in PIO mode.
- #define CDROM_READ_DMA 1
  Read sector(s) in DMA mode.

7.9.1 Detailed Description

How to read the sectors of a CD, via PIO or DMA. 4th parameter of cdrom_read_sectors_ex.

7.9.2 Macro Definition Documentation

CDROM_READ_DMA

```
#define CDROM_READ_DMA 1
```

Read sector(s) in DMA mode.

CDROM_READ_PIO

```
#define CDROM_READ_PIO 0
```

Read sector(s) in PIO mode.

7.10 CD-ROM Read Sector Part

Macros

- #define CDROM_READ_WHOLE_SECTOR 0x1000

  Read the whole sector.
- #define CDROM_READ_DATA_AREA 0x2000

  Read the data area.

7.10.1 Detailed Description

Parts of the a CD-ROM sector to read. These are possible values for the third parameter word sent with the change data type syscall.

7.10.2 Macro Definition Documentation

CDROM_READ_DATA_AREA

```
#define CDROM_READ_DATA_AREA 0x2000
```

Read the data area.

CDROM_READ_WHOLE_SECTOR

```
#define CDROM_READ_WHOLE_SECTOR 0x1000
```

Read the whole sector.

7.11 CD-ROM Read Subcode Type

Macros

- #define CD_SUB_Q_ALL 0

  Read all Subcode Data.
- #define CD_SUB_Q_CHANNEL 1

  Read Q Channel Subcode Data.
- #define CD_SUB_MEDIA_CATALOG 2

  Read the Media Catalog Subcode Data.
- #define CD_SUB_TRACK_ISRC 3

  Read the ISRC Subcode Data.
- #define CD_SUB_RESERVED 4

  Reserved.

7.11.1 Detailed Description

Types of data available to read from the sector subcode. These are possible values for the first parameter sent to the GETSCD syscall.

7.11.2 Macro Definition Documentation

CD_SUB_MEDIA_CATALOG

```
#define CD_SUB_MEDIA_CATALOG 2
```

Read the Media Catalog Subcode Data.

CD_SUB_Q_ALL

```
#define CD_SUB_Q_ALL 0
```

Read all Subcode Data.

CD_SUB_Q_CHANNEL

```
#define CD_SUB_Q_CHANNEL 1
```

Read Q Channel Subcode Data.

CD_SUB_RESERVED

```
#define CD_SUB_RESERVED 4
```

Reserved.

CD_SUB_TRACK_ISRC

```
#define CD_SUB_TRACK_ISRC 3
```

Read the ISRC Subcode Data.

7.12 CD-ROM Subcode audio status

Macros

- #define CD_SUB_AUDIO_STATUS_INVALID 0x00
- #define CD_SUB_AUDIO_STATUS_PLAYING 0x11
- #define CD_SUB_AUDIO_STATUS_PAUSED 0x12
- #define CD_SUB_AUDIO_STATUS_ENDED 0x13
- #define CD_SUB_AUDIO_STATUS_ERROR 0x14
- #define CD_SUB_AUDIO_STATUS_NO_INFO 0x15

7.12.1 Detailed Description

Information about CDDA playback from GETSCD syscall.

7.12.2 Macro Definition Documentation

CD_SUB_AUDIO_STATUS_ENDED

```
#define CD_SUB_AUDIO_STATUS_ENDED 0x13
```

CD_SUB_AUDIO_STATUS_ERROR

```
#define CD_SUB_AUDIO_STATUS_ERROR 0x14
```

CD_SUB_AUDIO_STATUS_INVALID

```
#define CD_SUB_AUDIO_STATUS_INVALID 0x00
```

CD_SUB_AUDIO_STATUS_NO_INFO

```
#define CD_SUB_AUDIO_STATUS_NO_INFO 0x15
```

CD_SUB_AUDIO_STATUS_PAUSED

```
#define CD_SUB_AUDIO_STATUS_PAUSED 0x12
```

CD_SUB_AUDIO_STATUS_PLAYING

```
#define CD_SUB_AUDIO_STATUS_PLAYING 0x11
```

7.13 CD-ROM TOC access macros

Macros

- #define TOC_LBA(n) ((n) & 0x00ffffff)

  Get the FAD address of a TOC entry.
- #define TOC_ADR(n) (((n) & 0x0f000000) >> 24)

  Get the address of a TOC entry.
- #define TOC_CTRL(n) (((n) & 0xf0000000) >> 28)

  Get the control data of a TOC entry.
- #define TOC_TRACK(n) (((n) & 0x00ff0000) >> 16)

  Get the track number of a TOC entry.

7.13.1 Detailed Description

7.13.2 Macro Definition Documentation

TOC_ADR

```
#define TOC_ADR(
              n ) ( ((n) & 0x0f000000) >> 24 )
```

Get the address of a TOC entry.

Parameters

n	The entry from the TOC to look at.

Returns

The entry's address.

TOC_CTRL

```
#define TOC_CTRL(
              n ) ( ((n) & 0xf0000000) >> 28 )
```

Get the control data of a TOC entry.

Parameters

n	The entry from the TOC to look at.

Returns

The entry's control value.

TOC_LBA

```
#define TOC_LBA(
              n ) ((n) & 0x00ffffff)
```

Get the FAD address of a TOC entry.

Parameters

n	The actual entry from the TOC to look at.

Returns

The FAD of the entry.

TOC_TRACK

```
#define TOC_TRACK(
              n ) ( ((n) & 0x00ff0000) >> 16 )
```

Get the track number of a TOC entry.

Parameters

n	The entry from the TOC to look at.

Returns

The entry's track.

7.14 CD-ROM command responses

Macros

- #define ERR_OK 0

  No error.
- #define ERR_NO_DISC 1

  No disc in drive.
- #define ERR_DISC_CHG 2

  Disc changed, but not reinitted yet.
- #define ERR_SYS 3

  System error.
- #define ERR_ABORTED 4

  Command aborted.
- #define ERR_NO_ACTIVE 5

  System inactive?
- #define ERR_TIMEOUT 6

  Aborted due to timeout.

7.14.1 Detailed Description

These are the values that the various functions can return as error codes.

7.14.2 Macro Definition Documentation

ERR_ABORTED

```
#define ERR_ABORTED 4
```

Command aborted.

ERR_DISC_CHG

```
#define ERR_DISC_CHG 2
```

Disc changed, but not reinitted yet.

ERR_NO_ACTIVE

```
#define ERR_NO_ACTIVE 5
```

System inactive?

ERR_NO_DISC

```
#define ERR_NO_DISC 1
```

No disc in drive.

ERR_OK

```
#define ERR_OK 0
```

No error.

ERR_SYS

```
#define ERR_SYS 3
```

System error.

ERR_TIMEOUT

```
#define ERR_TIMEOUT 6
```

Aborted due to timeout.

7.15 CD-ROM drive disc types

Macros

- #define CD_CDDA 0x00
  Audio CD (Red book) or no disc.
- #define CD_CDROM 0x10
  CD-ROM or CD-R (Yellow book)
- #define CD_CDROM_XA 0x20
  CD-ROM XA (Yellow book extension)
- #define CD_CDI 0x30
  CD-i (Green book)
- #define CD_GDROM 0x80
  GD-ROM.
- #define CD_FAIL 0xf0
  Need reset syscalls.

7.15.1 Detailed Description

These are the values that can be returned as the disc_type parameter from the cdrom_get_status() function.

7.15.2 Macro Definition Documentation

CD_CDDA

```
#define CD_CDDA 0x00
```

Audio CD (Red book) or no disc.

CD_CDI

```
#define CD_CDI 0x30
```

CD-i (Green book)

CD_CDROM

```
#define CD_CDROM 0x10
```

CD-ROM or CD-R (Yellow book)

CD_CDROM_XA

```
#define CD_CDROM_XA 0x20
```

CD-ROM XA (Yellow book extension)

CD_FAIL

```
#define CD_FAIL 0xf0
```

Need reset syscalls.

CD_GDROM

```
#define CD_GDROM 0x80
```

GD-ROM.

7.16 CD-ROM status values

Macros

- #define CD_STATUS_READ_FAIL -1
  Can't read status.
- #define CD_STATUS_BUSY 0
  Drive is busy.
- #define CD_STATUS_PAUSED 1
  Disc is paused.
- #define CD_STATUS_STANDBY 2
  Drive is in standby.
- #define CD_STATUS_PLAYING 3
  Drive is currently playing.
- #define CD_STATUS_SEEKING 4
  Drive is currently seeking.
- #define CD_STATUS_SCANNING 5
  Drive is scanning.
- #define CD_STATUS_OPEN 6
  Disc tray is open.
- #define CD_STATUS_NO_DISC 7
  No disc inserted.
- #define CD_STATUS_RETRY 8
  Retry is needed.
- #define CD_STATUS_ERROR 9
  System error.
- #define CD_STATUS_FATAL 12
  Need reset syscalls.

7.16.1 Detailed Description

These are the values that can be returned as the status parameter from the cdrom_get_status() function.

7.16.2 Macro Definition Documentation

CD_STATUS_BUSY

```
#define CD_STATUS_BUSY 0
```

Drive is busy.

CD_STATUS_ERROR

```
#define CD_STATUS_ERROR 9
```

System error.

CD_STATUS_FATAL

```
#define CD_STATUS_FATAL 12
```

Need reset syscalls.

CD_STATUS_NO_DISC

```
#define CD_STATUS_NO_DISC 7
```

No disc inserted.

CD_STATUS_OPEN

```
#define CD_STATUS_OPEN 6
```

Disc tray is open.

CD_STATUS_PAUSED

```
#define CD_STATUS_PAUSED 1
```

Disc is paused.

CD_STATUS_PLAYING

```
#define CD_STATUS_PLAYING 3
```

Drive is currently playing.

CD_STATUS_READ_FAIL

```
#define CD_STATUS_READ_FAIL -1
```

Can't read status.

CD_STATUS_RETRY

```
#define CD_STATUS_RETRY 8
```

Retry is needed.

CD_STATUS_SCANNING

```
#define CD_STATUS_SCANNING 5
```

Drive is scanning.

CD_STATUS_SEEKING

```
#define CD_STATUS_SEEKING 4
```

Drive is currently seeking.

CD_STATUS_STANDBY

```
#define CD_STATUS_STANDBY 2
```

Drive is in standby.

7.17 CD-ROM syscall command codes

Macros

- #define CMD_CHECK_LICENSE 2

  Check license.
- #define CMD_REQ_SPI_CMD 4

  Request to Sega Packet Interface.
- #define CMD_PIOREAD 16

  Read via PIO.
- #define CMD_DMAREAD 17

  Read via DMA.
- #define CMD_GETTOC 18

  Read TOC.
- #define CMD_GETTOC2 19

  Read TOC.
- #define CMD_PLAY 20

  Play track.
- #define CMD_PLAY2 21

  Play sectors.
- #define CMD_PAUSE 22

  Pause playback.
- #define CMD_RELEASE 23

  Resume from pause.
- #define CMD_INIT 24

  Initialize the drive.
- #define CMD_DMA_ABORT 25

  Abort DMA transfer.
- #define CMD_OPEN_TRAY 26

  Open CD tray (on DevBox?)
- #define CMD_SEEK 27

  Seek to a new position.
- #define CMD_DMAREAD_STREAM 28

  Stream DMA until end/abort.
- #define CMD_NOP 29

  No operation.
- #define CMD_REQ_MODE 30

  Request mode.
- #define CMD_SET_MODE 31

  Setup mode.
- #define CMD_SCAN_CD 32

  Scan CD.
- #define CMD_STOP 33

  Stop the disc from spinning.
- #define CMD_GETSCD 34

  Get subcode data.
- #define CMD_GETSES 35

Get session.

- #define CMD_REQ_STAT 36

  Request stat.

- #define CMD_PIOREAD_STREAM 37

  Stream PIO until end/abort.

- #define CMD_DMAREAD_STREAM_EX 38

  Stream DMA transfer.

- #define CMD_PIOREAD_STREAM_EX 39

  Stream PIO transfer.

- #define CMD_GET_VERS 40

  Get syscall driver version.

- #define CMD_MAX 47

  Max of GD syscall commands.

7.17.1 Detailed Description

These are the syscall command codes used to actually do stuff with the GD-ROM drive. These were originally provided by maiwe.

7.17.2 Macro Definition Documentation

CMD_CHECK_LICENSE

```
#define CMD_CHECK_LICENSE 2
```

Check license.

CMD_DMA_ABORT

```
#define CMD_DMA_ABORT 25
```

Abort DMA transfer.

CMD_DMAREAD

```
#define CMD_DMAREAD 17
```

Read via DMA.

CMD_DMAREAD_STREAM

```
#define CMD_DMAREAD_STREAM 28
```

Stream DMA until end/abort.

CMD_DMAREAD_STREAM_EX

```
#define CMD_DMAREAD_STREAM_EX 38
```

Stream DMA transfer.

CMD_GET_VERS

```
#define CMD_GET_VERS 40
```

Get syscall driver version.

CMD_GETSCD

```
#define CMD_GETSCD 34
```

Get subcode data.

CMD_GETSES

```
#define CMD_GETSES 35
```

Get session.

CMD_GETTOC

```
#define CMD_GETTOC 18
```

Read TOC.

CMD_GETTOC2

```
#define CMD_GETTOC2 19
```

Read TOC.

CMD_INIT

```
#define CMD_INIT 24
```

Initialize the drive.

CMD_MAX

```
#define CMD_MAX 47
```

Max of GD syscall commands.

CMD_NOP

```
#define CMD_NOP 29
```

No operation.

CMD_OPEN_TRAY

```
#define CMD_OPEN_TRAY 26
```

Open CD tray (on DevBox?)

CMD_PAUSE

```
#define CMD_PAUSE 22
```

Pause playback.

CMD_PIOREAD

```
#define CMD_PIOREAD 16
```

Read via PIO.

CMD_PIOREAD_STREAM

```
#define CMD_PIOREAD_STREAM 37
```

Stream PIO until end/abort.

CMD_PIOREAD_STREAM_EX

```
#define CMD_PIOREAD_STREAM_EX 39
```

Stream PIO transfer.

CMD_PLAY

```
#define CMD_PLAY 20
```

Play track.

CMD_PLAY2

```
#define CMD_PLAY2 21
```

Play sectors.

CMD_RELEASE

```
#define CMD_RELEASE 23
```

Resume from pause.

CMD_REQ_MODE

```
#define CMD_REQ_MODE 30
```

Request mode.

CMD_REQ_SPI_CMD

```
#define CMD_REQ_SPI_CMD 4
```

Request to Sega Packet Interface.

CMD_REQ_STAT

```
#define CMD_REQ_STAT 36
```

Request stat.

CMD_SCAN_CD

```
#define CMD_SCAN_CD 32
```

Scan CD.

CMD_SEEK

```
#define CMD_SEEK 27
```

Seek to a new position.

CMD_SET_MODE

```
#define CMD_SET_MODE 31
```

Setup mode.

CMD_STOP

```
#define CMD_STOP 33
```

Stop the disc from spinning.

7.18 CDDA read modes

Macros

- #define CDDA_TRACKS 1
  Play by track number.
- #define CDDA_SECTORS 2
  Play by sector number.

7.18.1 Detailed Description

Valid values to pass to the cdrom_cdda_play() function for the mode parameter.

7.18.2 Macro Definition Documentation

CDDA_SECTORS

```
#define CDDA_SECTORS 2
```

Play by sector number.

CDDA_TRACKS

```
#define CDDA_TRACKS 1
```

Play by track number.

7.19 Connection method types

Macros

- #define FLASHROM_ISP_DIALUP 0
  Dialup ISP.
- #define FLASHROM_ISP_DHCP 1
  DHCP-based ethernet.
- #define FLASHROM_ISP_PPPOE 2
  PPPoE-based ethernet.
- #define FLASHROM_ISP_STATIC 3
  Static IP-based ethernet.

7.19.1 Detailed Description

These values are representative of what type of ISP is configured in the flashrom.

7.19.2 Macro Definition Documentation

FLASHROM_ISP_DHCP

```
#define FLASHROM_ISP_DHCP 1
```

DHCP-based ethernet.

FLASHROM_ISP_DIALUP

```
#define FLASHROM_ISP_DIALUP 0
```

Dialup ISP.

FLASHROM_ISP_PPPOE

```
#define FLASHROM_ISP_PPPOE 2
```

PPPoE-based ethernet.

FLASHROM_ISP_STATIC

```
#define FLASHROM_ISP_STATIC 3
```

Static IP-based ethernet.

7.20 Console memory sizes

Macros

- #define HW_MEM_16 16777216
  16M retail Dreamcast
- #define HW_MEM_32 33554432
  32M NAOMI/modded Dreamcast

7.20.1 Detailed Description

These are the various memory sizes, in bytes, that can be returned by the HW_MEMSIZE macro.

7.20.2 Macro Definition Documentation

HW_MEM_16

```
#define HW_MEM_16 16777216
```

16M retail Dreamcast

HW_MEM_32

```
#define HW_MEM_32 33554432
```

32M NAOMI/modded Dreamcast

7.21 Console types

Macros

- #define HW_TYPE_RETAIL 0x0
  A retail Dreamcast.
- #define HW_TYPE_SET5 0x9
  A Set5.xx devkit.

7.21.1 Detailed Description

These are the various console types that can be returned by the hardware_sys_mode() function.

7.21.2 Macro Definition Documentation

HW_TYPE_RETAIL

```
#define HW_TYPE_RETAIL 0x0
```

A retail Dreamcast.

HW_TYPE_SET5

```
#define HW_TYPE_SET5 0x9
```

A Set5.xx devkit.

7.22 Constants and bitmasks for handling polygon

Macros

- #define PVR_TA_CMD_TYPE_SHIFT 24
- #define PVR_TA_CMD_TYPE_MASK (7 << PVR_TA_CMD_TYPE_SHIFT)
- #define PVR_TA_CMD_USERCLIP_SHIFT 16
- #define PVR_TA_CMD_USERCLIP_MASK (3 << PVR_TA_CMD_USERCLIP_SHIFT)
- #define PVR_TA_CMD_CLRFMT_SHIFT 4
- #define PVR_TA_CMD_CLRFMT_MASK (7 << PVR_TA_CMD_CLRFMT_SHIFT)
- #define PVR_TA_CMD_SPECULAR_SHIFT 2
- #define PVR_TA_CMD_SPECULAR_MASK (1 << PVR_TA_CMD_SPECULAR_SHIFT)
- #define PVR_TA_CMD_SHADE_SHIFT 1
- #define PVR_TA_CMD_SHADE_MASK (1 << PVR_TA_CMD_SHADE_SHIFT)
- #define PVR_TA_CMD_UVFMT_SHIFT 0
- #define PVR_TA_CMD_UVFMT_MASK (1 << PVR_TA_CMD_UVFMT_SHIFT)
- #define PVR_TA_CMD_MODIFIER_SHIFT 7
- #define PVR_TA_CMD_MODIFIER_MASK (1 << PVR_TA_CMD_MODIFIER_SHIFT)
- #define PVR_TA_CMD_MODIFIERMODE_SHIFT 6
- #define PVR_TA_CMD_MODIFIERMODE_MASK (1 << PVR_TA_CMD_MODIFIERMODE_SHIFT)
- #define PVR_TA_PM1_DEPTHCMP_SHIFT 29
- #define PVR_TA_PM1_DEPTHCMP_MASK (7 << PVR_TA_PM1_DEPTHCMP_SHIFT)
- #define PVR_TA_PM1_CULLING_SHIFT 27
- #define PVR_TA_PM1_CULLING_MASK (3 << PVR_TA_PM1_CULLING_SHIFT)
- #define PVR_TA_PM1_DEPTHWRITE_SHIFT 26
- #define PVR_TA_PM1_DEPTHWRITE_MASK (1 << PVR_TA_PM1_DEPTHWRITE_SHIFT)
- #define PVR_TA_PM1_TXRENABLE_SHIFT 25
- #define PVR_TA_PM1_TXRENABLE_MASK (1 << PVR_TA_PM1_TXRENABLE_SHIFT)
- #define PVR_TA_PM1_MODIFIERINST_SHIFT 29
- #define PVR_TA_PM1_MODIFIERINST_MASK (3 << PVR_TA_PM1_MODIFIERINST_SHIFT)
- #define PVR_TA_PM2_SRCBLEND_SHIFT 29
- #define PVR_TA_PM2_SRCBLEND_MASK (7 << PVR_TA_PM2_SRCBLEND_SHIFT)
- #define PVR_TA_PM2_DSTBLEND_SHIFT 26

- #define PVR_TA_PM2_DSTBLEND_MASK (7 << PVR_TA_PM2_DSTBLEND_SHIFT)
- #define PVR_TA_PM2_SRCENABLE_SHIFT 25
- #define PVR_TA_PM2_SRCENABLE_MASK (1 << PVR_TA_PM2_SRCENABLE_SHIFT)
- #define PVR_TA_PM2_DSTENABLE_SHIFT 24
- #define PVR_TA_PM2_DSTENABLE_MASK (1 << PVR_TA_PM2_DSTENABLE_SHIFT)
- #define PVR_TA_PM2_FOG_SHIFT 22
- #define PVR_TA_PM2_FOG_MASK (3 << PVR_TA_PM2_FOG_SHIFT)
- #define PVR_TA_PM2_CLAMP_SHIFT 21
- #define PVR_TA_PM2_CLAMP_MASK (1 << PVR_TA_PM2_CLAMP_SHIFT)
- #define PVR_TA_PM2_ALPHA_SHIFT 20
- #define PVR_TA_PM2_ALPHA_MASK (1 << PVR_TA_PM2_ALPHA_SHIFT)
- #define PVR_TA_PM2_TXRALPHA_SHIFT 19
- #define PVR_TA_PM2_TXRALPHA_MASK (1 << PVR_TA_PM2_TXRALPHA_SHIFT)
- #define PVR_TA_PM2_UVFLIP_SHIFT 17
- #define PVR_TA_PM2_UVFLIP_MASK (3 << PVR_TA_PM2_UVFLIP_SHIFT)
- #define PVR_TA_PM2_UVCLAMP_SHIFT 15
- #define PVR_TA_PM2_UVCLAMP_MASK (3 << PVR_TA_PM2_UVCLAMP_SHIFT)
- #define PVR_TA_PM2_FILTER_SHIFT 12
- #define PVR_TA_PM2_FILTER_MASK (7 << PVR_TA_PM2_FILTER_SHIFT)
- #define PVR_TA_PM2_MIPBIAS_SHIFT 8
- #define PVR_TA_PM2_MIPBIAS_MASK (15 << PVR_TA_PM2_MIPBIAS_SHIFT)
- #define PVR_TA_PM2_TXRENV_SHIFT 6
- #define PVR_TA_PM2_TXRENV_MASK (3 << PVR_TA_PM2_TXRENV_SHIFT)
- #define PVR_TA_PM2_USIZE_SHIFT 3
- #define PVR_TA_PM2_USIZE_MASK (7 << PVR_TA_PM2_USIZE_SHIFT)
- #define PVR_TA_PM2_VSIZE_SHIFT 0
- #define PVR_TA_PM2_VSIZE_MASK (7 << PVR_TA_PM2_VSIZE_SHIFT)
- #define PVR_TA_PM3_MIPMAP_SHIFT 31
- #define PVR_TA_PM3_MIPMAP_MASK (1 << PVR_TA_PM3_MIPMAP_SHIFT)
- #define PVR_TA_PM3_TXRFMT_SHIFT 0
- #define PVR_TA_PM3_TXRFMT_MASK 0xffffffff

7.22.1 Detailed Description

headers.

Note that thanks to the arrangement of constants, this is mainly a matter of bit shifting to compile headers...

7.22.2 Macro Definition Documentation

PVR_TA_CMD_CLRFMT_MASK

```
#define PVR_TA_CMD_CLRFMT_MASK (7 << PVR_TA_CMD_CLRFMT_SHIFT)
```

PVR_TA_CMD_CLRFMT_SHIFT

```
#define PVR_TA_CMD_CLRFMT_SHIFT 4
```

PVR_TA_CMD_MODIFIER_MASK

```
#define PVR_TA_CMD_MODIFIER_MASK (1 << PVR_TA_CMD_MODIFIER_SHIFT)
```

PVR_TA_CMD_MODIFIER_SHIFT

```
#define PVR_TA_CMD_MODIFIER_SHIFT 7
```

PVR_TA_CMD_MODIFIERMODE_MASK

```
#define PVR_TA_CMD_MODIFIERMODE_MASK (1 << PVR_TA_CMD_MODIFIERMODE_SHIFT)
```

PVR_TA_CMD_MODIFIERMODE_SHIFT

```
#define PVR_TA_CMD_MODIFIERMODE_SHIFT 6
```

PVR_TA_CMD_SHADE_MASK

```
#define PVR_TA_CMD_SHADE_MASK (1 << PVR_TA_CMD_SHADE_SHIFT)
```

PVR_TA_CMD_SHADE_SHIFT

```
#define PVR_TA_CMD_SHADE_SHIFT 1
```

PVR_TA_CMD_SPECULAR_MASK

```
#define PVR_TA_CMD_SPECULAR_MASK (1 << PVR_TA_CMD_SPECULAR_SHIFT)
```

PVR_TA_CMD_SPECULAR_SHIFT

```
#define PVR_TA_CMD_SPECULAR_SHIFT 2
```

PVR_TA_CMD_TYPE_MASK

```
#define PVR_TA_CMD_TYPE_MASK (7 << PVR_TA_CMD_TYPE_SHIFT)
```

PVR_TA_CMD_TYPE_SHIFT

```
#define PVR_TA_CMD_TYPE_SHIFT 24
```

PVR_TA_CMD_USERCLIP_MASK

```
#define PVR_TA_CMD_USERCLIP_MASK (3 << PVR_TA_CMD_USERCLIP_SHIFT)
```

PVR_TA_CMD_USERCLIP_SHIFT

```
#define PVR_TA_CMD_USERCLIP_SHIFT 16
```

PVR_TA_CMD_UVFMT_MASK

```
#define PVR_TA_CMD_UVFMT_MASK (1 << PVR_TA_CMD_UVFMT_SHIFT)
```

PVR_TA_CMD_UVFMT_SHIFT

```
#define PVR_TA_CMD_UVFMT_SHIFT 0
```

PVR_TA_PM1_CULLING_MASK

```
#define PVR_TA_PM1_CULLING_MASK (3 << PVR_TA_PM1_CULLING_SHIFT)
```

PVR_TA_PM1_CULLING_SHIFT

```
#define PVR_TA_PM1_CULLING_SHIFT 27
```

PVR_TA_PM1_DEPTHCMP_MASK

```
#define PVR_TA_PM1_DEPTHCMP_MASK (7 << PVR_TA_PM1_DEPTHCMP_SHIFT)
```

PVR_TA_PM1_DEPTHCMP_SHIFT

```
#define PVR_TA_PM1_DEPTHCMP_SHIFT 29
```

PVR_TA_PM1_DEPTHWRITE_MASK

```
#define PVR_TA_PM1_DEPTHWRITE_MASK (1 << PVR_TA_PM1_DEPTHWRITE_SHIFT)
```

PVR_TA_PM1_DEPTHWRITE_SHIFT

```
#define PVR_TA_PM1_DEPTHWRITE_SHIFT 26
```

PVR_TA_PM1_MODIFIERINST_MASK

```
#define PVR_TA_PM1_MODIFIERINST_MASK (3 << PVR_TA_PM1_MODIFIERINST_SHIFT)
```

PVR_TA_PM1_MODIFIERINST_SHIFT

```
#define PVR_TA_PM1_MODIFIERINST_SHIFT 29
```

PVR_TA_PM1_TXRENABLE_MASK

```
#define PVR_TA_PM1_TXRENABLE_MASK (1 << PVR_TA_PM1_TXRENABLE_SHIFT)
```

PVR_TA_PM1_TXRENABLE_SHIFT

```
#define PVR_TA_PM1_TXRENABLE_SHIFT 25
```

PVR_TA_PM2_ALPHA_MASK

```
#define PVR_TA_PM2_ALPHA_MASK (1 << PVR_TA_PM2_ALPHA_SHIFT)
```

PVR_TA_PM2_ALPHA_SHIFT

```
#define PVR_TA_PM2_ALPHA_SHIFT 20
```

PVR_TA_PM2_CLAMP_MASK

```
#define PVR_TA_PM2_CLAMP_MASK (1 << PVR_TA_PM2_CLAMP_SHIFT)
```

PVR_TA_PM2_CLAMP_SHIFT

```
#define PVR_TA_PM2_CLAMP_SHIFT 21
```

PVR_TA_PM2_DSTBLEND_MASK

```
#define PVR_TA_PM2_DSTBLEND_MASK (7 << PVR_TA_PM2_DSTBLEND_SHIFT)
```

PVR_TA_PM2_DSTBLEND_SHIFT

```
#define PVR_TA_PM2_DSTBLEND_SHIFT 26
```

PVR_TA_PM2_DSTENABLE_MASK

```
#define PVR_TA_PM2_DSTENABLE_MASK (1 << PVR_TA_PM2_DSTENABLE_SHIFT)
```

PVR_TA_PM2_DSTENABLE_SHIFT

```
#define PVR_TA_PM2_DSTENABLE_SHIFT 24
```

PVR_TA_PM2_FILTER_MASK

```
#define PVR_TA_PM2_FILTER_MASK (7 << PVR_TA_PM2_FILTER_SHIFT)
```

PVR_TA_PM2_FILTER_SHIFT

```
#define PVR_TA_PM2_FILTER_SHIFT 12
```

PVR_TA_PM2_FOG_MASK

```
#define PVR_TA_PM2_FOG_MASK (3 << PVR_TA_PM2_FOG_SHIFT)
```

PVR_TA_PM2_FOG_SHIFT

```
#define PVR_TA_PM2_FOG_SHIFT 22
```

PVR_TA_PM2_MIPBIAS_MASK

```
#define PVR_TA_PM2_MIPBIAS_MASK (15 << PVR_TA_PM2_MIPBIAS_SHIFT)
```

PVR_TA_PM2_MIPBIAS_SHIFT

```
#define PVR_TA_PM2_MIPBIAS_SHIFT 8
```

PVR_TA_PM2_SRCBLEND_MASK

```
#define PVR_TA_PM2_SRCBLEND_MASK (7 << PVR_TA_PM2_SRCBLEND_SHIFT)
```

PVR_TA_PM2_SRCBLEND_SHIFT

```
#define PVR_TA_PM2_SRCBLEND_SHIFT 29
```

PVR_TA_PM2_SRCENABLE_MASK

```
#define PVR_TA_PM2_SRCENABLE_MASK (1 << PVR_TA_PM2_SRCENABLE_SHIFT)
```

PVR_TA_PM2_SRCENABLE_SHIFT

```
#define PVR_TA_PM2_SRCENABLE_SHIFT 25
```

PVR_TA_PM2_TXRALPHA_MASK

```
#define PVR_TA_PM2_TXRALPHA_MASK (1 << PVR_TA_PM2_TXRALPHA_SHIFT)
```

PVR_TA_PM2_TXRALPHA_SHIFT

```
#define PVR_TA_PM2_TXRALPHA_SHIFT 19
```

PVR_TA_PM2_TXRENV_MASK

```
#define PVR_TA_PM2_TXRENV_MASK (3 << PVR_TA_PM2_TXRENV_SHIFT)
```

PVR_TA_PM2_TXRENV_SHIFT

```
#define PVR_TA_PM2_TXRENV_SHIFT 6
```

PVR_TA_PM2_USIZE_MASK

```
#define PVR_TA_PM2_USIZE_MASK (7 << PVR_TA_PM2_USIZE_SHIFT)
```

PVR_TA_PM2_USIZE_SHIFT

```
#define PVR_TA_PM2_USIZE_SHIFT 3
```

PVR_TA_PM2_UVCLAMP_MASK

```
#define PVR_TA_PM2_UVCLAMP_MASK (3 << PVR_TA_PM2_UVCLAMP_SHIFT)
```

PVR_TA_PM2_UVCLAMP_SHIFT

```
#define PVR_TA_PM2_UVCLAMP_SHIFT 15
```

PVR_TA_PM2_UVFLIP_MASK

```
#define PVR_TA_PM2_UVFLIP_MASK (3 << PVR_TA_PM2_UVFLIP_SHIFT)
```

PVR_TA_PM2_UVFLIP_SHIFT

```
#define PVR_TA_PM2_UVFLIP_SHIFT 17
```

PVR_TA_PM2_VSIZE_MASK

```
#define PVR_TA_PM2_VSIZE_MASK (7 << PVR_TA_PM2_VSIZE_SHIFT)
```

PVR_TA_PM2_VSIZE_SHIFT

```
#define PVR_TA_PM2_VSIZE_SHIFT 0
```

PVR_TA_PM3_MIPMAP_MASK

```
#define PVR_TA_PM3_MIPMAP_MASK (1 << PVR_TA_PM3_MIPMAP_SHIFT)
```

PVR_TA_PM3_MIPMAP_SHIFT

```
#define PVR_TA_PM3_MIPMAP_SHIFT 31
```

PVR_TA_PM3_TXRFMT_MASK

```
#define PVR_TA_PM3_TXRFMT_MASK 0xffffffff
```

PVR_TA_PM3_TXRFMT_SHIFT

```
#define PVR_TA_PM3_TXRFMT_SHIFT 0
```

7.23 Controller

Controller Maple Device API.

Modules

- Querying Capabilities

  API used to query for a controller's capabilities.
- Querying Inputs

  API used to query for input state.
- Querying Types

  API for determining controller types.

Files

- file controller.h

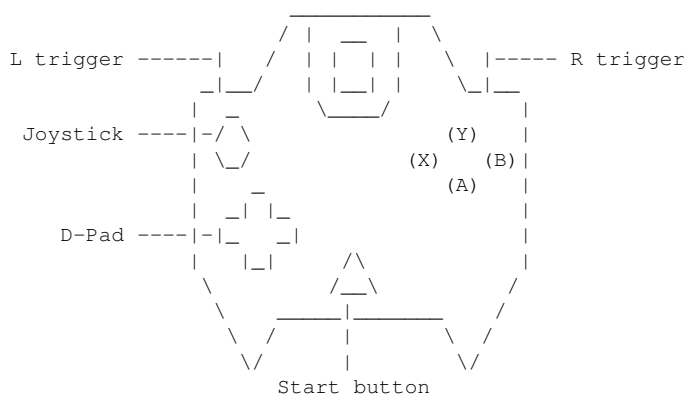
  Definitions for using the controller device.

7.23.1 Detailed Description

Controller Maple Device API.

This module contains the public API for the controller maple driver.

A standard, first-party Dreamcast controller has the following button configuration:



You can grab a pointer to a connected controller by using the following:

```
maple_device_t *device = maple_enum_type(N, MAPLE_FUNC_CONTROLLER);

if(device) printf("Controller found!\n");
else printf("Controller not found!\n");
```

where N is the controller number. 0 would be the first controller found, which may not necessarily be on port A.

7.23.2 Querying Capabilities

API used to query for a controller's capabilities.

Modules

- Capabilities

  Bit masks used to identify controller capabilities.
- Capability Groups

  Bit masks representing common groups of capabilities.

Functions

- int cont_has_capabilities (const struct maple_device *cont, uint32_t capabilities)

  Check for controller capabilities.

7.23.2.1 Detailed Description

API used to query for a controller's capabilities.

The following API is used to query for the support of individual or groups of capabilities by a particular device.

7.23.2.2 Function Documentation

cont_has_capabilities()

```
int cont_has_capabilities (
            const struct maple_device * cont,
            uint32_t capabilities )
```

Check for controller capabilities.

Checks whether or not a controller implements the capabilities associated with the given type.

Note

Controller capability reporting is an extremely generic mechanism, such that many peripherals may implement the same capability in completely different ways. For example, the Samba De Amigo maraca controller will advertise itself as a dual-analog device, with each maraca being an analog stick.

Parameters

cont	Pointer to a Maple device structure which implements the CONTROLLER function.
capabilities	Capability mask the controller is expected to implement

Return values

1	The controller supports the given capabilities.
0	The controller doesn't support the given capabilities.
-1	Invalid controller.

See also

cont_is_type

7.23.2.3 Capabilities

Bit masks used to identify controller capabilities.

Macros

- #define CONT_CAPABILITY_C (1<<24)
  C button capability mask.
- #define CONT_CAPABILITY_B (1<<25)
  B button capability mask.
- #define CONT_CAPABILITY_A (1<<26)
  A button capability mask.
- #define CONT_CAPABILITY_START (1<<27)
  Start button capability mask.
- #define CONT_CAPABILITY_DPAD_UP (1<<28)
  First Dpad up capability mask.
- #define CONT_CAPABILITY_DPAD_DOWN (1<<29)
  First Dpad down capability mask.
- #define CONT_CAPABILITY_DPAD_LEFT (1<<30)
  First Dpad left capability mask.
- #define CONT_CAPABILITY_DPAD_RIGHT (1<<31)
  First Dpad right capability mask.
- #define CONT_CAPABILITY_Z (1<<16)
  Z button capability mask.
- #define CONT_CAPABILITY_Y (1<<17)
  Y button capability mask.
- #define CONT_CAPABILITY_X (1<<18)
  X button capability mask.
- #define CONT_CAPABILITY_D (1<<19)
  D button capability mask.
- #define CONT_CAPABILITY_DPAD2_UP (1<<20)
  Second Dpad up capability mask.
- #define CONT_CAPABILITY_DPAD2_DOWN (1<<21)
  Second Dpad down capability mask.
- #define CONT_CAPABILITY_DPAD2_LEFT (1<<22)

Second Dpad left capability mask.

- #define CONT_CAPABILITY_DPAD2_RIGHT (1<<23)

  Second Dpad right capability mask.
- #define CONT_CAPABILITY_RTRIG (1<<8)

  Right trigger capability mask.
- #define CONT_CAPABILITY_LTRIG (1<<9)

  Left trigger capability mask.
- #define CONT_CAPABILITY_ANALOG_X (1<<10)

  First analog X axis capability mask.
- #define CONT_CAPABILITY_ANALOG_Y (1<<11)

  First analog Y axis capability mask.
- #define CONT_CAPABILITY_ANALOG2_X (1<<12)

  Second analog X axis capability mask.
- #define CONT_CAPABILITY_ANALOG2_Y (1<<13)

  Second analog Y axis capability mask.

7.23.2.3.1 Detailed Description

Bit masks used to identify controller capabilities.

These bits will be set in the function_data for the controller's deviceinfo if the controller supports the corresponding button/axis capability.

Note

The ordering here is so that they match the order found in Inputs.

7.23.2.3.2 Macro Definition Documentation

CONT_CAPABILITY_A

```
#define CONT_CAPABILITY_A (1<<26)
```

A button capability mask.

CONT_CAPABILITY_ANALOG2_X

```
#define CONT_CAPABILITY_ANALOG2_X (1<<12)
```

Second analog X axis capability mask.

CONT_CAPABILITY_ANALOG2_Y

```
#define CONT_CAPABILITY_ANALOG2_Y (1<<13)
```

Second analog Y axis capability mask.

CONT_CAPABILITY_ANALOG_X

```
#define CONT_CAPABILITY_ANALOG_X (1<<10)
```

First analog X axis capability mask.

CONT_CAPABILITY_ANALOG_Y

```
#define CONT_CAPABILITY_ANALOG_Y (1<<11)
```

First analog Y axis capability mask.

CONT_CAPABILITY_B

```
#define CONT_CAPABILITY_B (1<<25)
```

B button capability mask.

CONT_CAPABILITY_C

```
#define CONT_CAPABILITY_C (1<<24)
```

C button capability mask.

CONT_CAPABILITY_D

```
#define CONT_CAPABILITY_D (1<<19)
```

D button capability mask.

CONT_CAPABILITY_DPAD2_DOWN

```
#define CONT_CAPABILITY_DPAD2_DOWN (1<<21)
```

Second Dpad down capability mask.

CONT_CAPABILITY_DPAD2_LEFT

```
#define CONT_CAPABILITY_DPAD2_LEFT (1<<22)
```

Second Dpad left capability mask.

CONT_CAPABILITY_DPAD2_RIGHT

```
#define CONT_CAPABILITY_DPAD2_RIGHT (1<<23)
```

Second Dpad right capability mask.

CONT_CAPABILITY_DPAD2_UP

```
#define CONT_CAPABILITY_DPAD2_UP (1<<20)
```

Second Dpad up capability mask.

CONT_CAPABILITY_DPAD_DOWN

```
#define CONT_CAPABILITY_DPAD_DOWN (1<<29)
```

First Dpad down capability mask.

CONT_CAPABILITY_DPAD_LEFT

```
#define CONT_CAPABILITY_DPAD_LEFT (1<<30)
```

First Dpad left capability mask.

CONT_CAPABILITY_DPAD_RIGHT

```
#define CONT_CAPABILITY_DPAD_RIGHT (1<<31)
```

First Dpad right capability mask.

CONT_CAPABILITY_DPAD_UP

```
#define CONT_CAPABILITY_DPAD_UP (1<<28)
```

First Dpad up capability mask.

CONT_CAPABILITY_LTRIG

```
#define CONT_CAPABILITY_LTRIG (1<<9)
```

Left trigger capability mask.

CONT_CAPABILITY_RTRIG

```
#define CONT_CAPABILITY_RTRIG (1<<8)
```

Right trigger capability mask.

CONT_CAPABILITY_START

```
#define CONT_CAPABILITY_START (1<<27)
```

Start button capability mask.

CONT_CAPABILITY_X

```
#define CONT_CAPABILITY_X (1<<18)
```

X button capability mask.

CONT_CAPABILITY_Y

```
#define CONT_CAPABILITY_Y (1<<17)
```

Y button capability mask.

CONT_CAPABILITY_Z

```
#define CONT_CAPABILITY_Z (1<<16)
```

Z button capability mask.

7.23.2.4 Capability Groups

Bit masks representing common groups of capabilities.

Macros

- #define CONT_CAPABILITIES_STANDARD_BUTTONS
  Standard button (A, B, X, Y, Start) controller capabilities.
- #define CONT_CAPABILITIES_DPAD
  Directional pad (up, down, left right) controller capabilities.
- #define CONT_CAPABILITIES_ANALOG
  Analog stick (X, Y axes) controller capabilities.
- #define CONT_CAPABILITIES_TRIGGERS
  Trigger (L, R lever) controller capabilities.
- #define CONT_CAPABILITIES_EXTENDED_BUTTONS
  Extended button (C, Z) controller capabilities.
- #define CONT_CAPABILITIES_SECONDARY_DPAD
  Secondary directional pad (up, down, left, right) controller capabilities.
- #define CONT_CAPABILITIES_SECONDARY_ANALOG
  Secondary analog stick (X, Y axes) controller capabilities.
- #define CONT_CAPABILITIES_DUAL_DPAD
  Both directional pads (up, down, left right) controller capabilities.
- #define CONT_CAPABILITIES_DUAL_ANALOG
  Both analog sticks (X, Y axes) controller capabilities.

7.23.2.4.1 Detailed Description

Bit masks representing common groups of capabilities.

These are a sets of capabilties providing a convenient way to test for high-level features, such as dual-analog sticks or extra buttons.

7.23.2.4.2 Macro Definition Documentation

CONT_CAPABILITIES_ANALOG

```
#define CONT_CAPABILITIES_ANALOG
```

Value:

```
(CONT_CAPABILITY_ANALOG_X | \
CONT_CAPABILITY_ANALOG_Y)
```

Analog stick (X, Y axes) controller capabilities.

CONT_CAPABILITIES_DPAD

```
#define CONT_CAPABILITIES_DPAD
```

Value:

```
(CONT_CAPABILITY_DPAD_UP | \
CONT_CAPABILITY_DPAD_DOWN | \
CONT_CAPABILITY_DPAD_LEFT | \
CONT_CAPABILITY_DPAD_RIGHT)
```

Directional pad (up, down, left right) controller capabilities.

CONT_CAPABILITIES_DUAL_ANALOG

```
#define CONT_CAPABILITIES_DUAL_ANALOG
```

Value:

```
(CONT_CAPABILITIES_ANALOG | \
CONT_CAPABILITIES_SECONDARY_ANALOG)
```

Both analog sticks (X, Y axes) controller capabilities.

CONT_CAPABILITIES_DUAL_DPAD

```
#define CONT_CAPABILITIES_DUAL_DPAD
```

Value:

```
(CONT_CAPABILITIES_DPAD | \
CONT_CAPABILITIES_SECONDARY_DPAD)
```

Both directional pads (up, down, left right) controller capabilities.

CONT_CAPABILITIES_EXTENDED_BUTTONS

```
#define CONT_CAPABILITIES_EXTENDED_BUTTONS
```

Value:

```
(CONT_CAPABILITY_C | \
CONT_CAPABILITY_Z)
```

Extended button (C, Z) controller capabilities.

CONT_CAPABILITIES_SECONDARY_ANALOG

```
#define CONT_CAPABILITIES_SECONDARY_ANALOG
```

Value:

```
(CONT_CAPABILITY_ANALOG2_X | \
CONT_CAPABILITY_ANALOG2_Y)
```

Secondary analog stick (X, Y axes) controller capabilities.

CONT_CAPABILITIES_SECONDARY_DPAD

```
#define CONT_CAPABILITIES_SECONDARY_DPAD
```

Value:

```
(CONT_CAPABILITY_DPAD2_UP | \
CONT_CAPABILITY_DPAD2_DOWN | \
CONT_CAPABILITY_DPAD2_LEFT | \
CONT_CAPABILITY_DPAD2_RIGHT)
```

Secondary directional pad (up, down, left, right) controller capabilities.

CONT_CAPABILITIES_STANDARD_BUTTONS

```
#define CONT_CAPABILITIES_STANDARD_BUTTONS
```

Value:

```
(CONT_CAPABILITY_A | \
CONT_CAPABILITY_B | \
CONT_CAPABILITY_X | \
CONT_CAPABILITY_Y | \
CONT_CAPABILITY_START)
```

Standard button (A, B, X, Y, Start) controller capabilities.

CONT_CAPABILITIES_TRIGGERS

```
#define CONT_CAPABILITIES_TRIGGERS
```

Value:

```
(CONT_CAPABILITY_LTRIG | \
CONT_CAPABILITY_RTRIG)
```

Trigger (L, R lever) controller capabilities.

7.23.3 Querying Inputs

API used to query for input state.

Modules

- Inputs

  Collection of all status masks for checking input.

Data Structures

- struct cont_state_t

  Controller state structure.

Macros

- #define CONT_RESET_BUTTONS (CONT_A | CONT_B | CONT_X | CONT_Y | CONT_START)

  Controller buttons for standard reset action.

Typedefs

- typedef void(* cont_btn_callback_t) (uint8_t addr, uint32_t btns)

  Controller automatic callback type.

Functions

- void cont_btn_callback (uint8_t addr, uint32_t btns, cont_btn_callback_t cb)

  Set an automatic button press callback.

7.23.3.1 Detailed Description

API used to query for input state.

The following API is used to check for a controller's input state.

You can grab a controller's state structure, containing the state of all of its inputs by using:

```
cont_state_t *state = (cont_state_t *)maple_dev_status(device);
```

Next you can check for the state of a particular button with:

```
if(state->a)                // Check via bitfield
    printf("Pressed A".);
```

or

```
if(state->buttons & CONT_A) // Check via applying bitmask
    printf("Pressed A.")
```

7.23.3.2 Macro Definition Documentation

CONT_RESET_BUTTONS

```
#define CONT_RESET_BUTTONS (CONT_A | CONT_B | CONT_X | CONT_Y | CONT_START)
```

Controller buttons for standard reset action.

Convenience macro providing the standard button combination used as a reset mechanism by most retail games.

7.23.3.3 Typedef Documentation

cont_btn_callback_t

```
typedef void(* cont_btn_callback_t) (uint8_t addr, uint32_t btns)
```

Controller automatic callback type.

Functions of this type can be set with cont_btn_callback() to respond automatically to the specified set of buttons being pressed. This can be used, for instance, to implement the standard A+B+X+Y+Start method of ending the program running.

Warning

Your callback will be invoked within a context with interrupts disabled. See cont_btn_callback for more information.

Parameters

addr	Maple BUS address to poll for the button mask on, or 0 for all ports.
btns	Mask of all buttons which should be pressed to trigger the callback.

See also

cont_btn_callback

7.23.3.4 Function Documentation

cont_btn_callback()

```
void cont_btn_callback (
            uint8_t addr,
            uint32_t btns,
            cont_btn_callback_t cb )
```

Set an automatic button press callback.

This function sets a callback function to be called when the specified controller has the set of buttons given pressed.

Note

The callback gets invoked for the given maple port; however, providing an address of '0' will cause it to be invoked for any port with a device pressing the given buttons. Since you are passed back the address of this device, You are free to implement your own filtering logic within your callback.

Warning

The provided callback function is invoked within a context which has interrupts disabled. This means that you should not do any sort of complex processing or make any API calls which depend on interrupts to complete, such as Maple or etherent processing whichy rely on packet transmission, any sleeping or threading calls, blocking on any sort of file I/O, etc. This mechanism is typically used to quickly terminate the application and should be used with caution.

Parameters

addr	The controller to listen on (or 0 for all ports). This value can be obtained by using maple_addr().
btns	The buttons bitmask to match.
cb	The callback to call when the buttons are pressed.

7.23.3.5 Inputs

Collection of all status masks for checking input.

Macros

- #define CONT_C (1<<0)
  C button Mask.
- #define CONT_B (1<<1)
  B button Mask.
- #define CONT_A (1<<2)
  A button Mask.
- #define CONT_START (1<<3)
  Start button Mask.
- #define CONT_DPAD_UP (1<<4)
  Main Dpad Up button Mask.
- #define CONT_DPAD_DOWN (1<<5)
  Main Dpad Down button Mask.
- #define CONT_DPAD_LEFT (1<<6)
  Main Dpad Left button Mask.
- #define CONT_DPAD_RIGHT (1<<7)
  Main Dpad right button Mask.
- #define CONT_Z (1<<8)
  Z button Mask.
- #define CONT_Y (1<<9)
  Y button Mask.
- #define CONT_X (1<<10)
  X button Mask.
- #define CONT_D (1<<11)
  D button Mask.
- #define CONT_DPAD2_UP (1<<12)
  Secondary Dpad Up button Mask.
- #define CONT_DPAD2_DOWN (1<<13)
  Secondary Dpad Down button Mask.
- #define CONT_DPAD2_LEFT (1<<14)
  Secondary Dpad Left button Mask.
- #define CONT_DPAD2_RIGHT (1<<15)
  Secondary Dpad Right button Mask.

7.23.3.5.1 Detailed Description

Collection of all status masks for checking input.

A set of bitmasks representing each input source on a controller, used to check its status.

7.23.3.5.2 Macro Definition Documentation

CONT_A

```
#define CONT_A (1<<2)
```

A button Mask.

CONT_B

```
#define CONT_B (1<<1)
```

B button Mask.

CONT_C

```
#define CONT_C (1<<0)
```

C button Mask.

CONT_D

```
#define CONT_D (1<<11)
```

D button Mask.

CONT_DPAD2_DOWN

```
#define CONT_DPAD2_DOWN (1<<13)
```

Secondary Dpad Down button Mask.

CONT_DPAD2_LEFT

```
#define CONT_DPAD2_LEFT (1<<14)
```

Secondary Dpad Left button Mask.

CONT_DPAD2_RIGHT

```
#define CONT_DPAD2_RIGHT (1<<15)
```

Secondary Dpad Right button Mask.

CONT_DPAD2_UP

```
#define CONT_DPAD2_UP (1<<12)
```

Secondary Dpad Up button Mask.

CONT_DPAD_DOWN

```
#define CONT_DPAD_DOWN (1<<5)
```

Main Dpad Down button Mask.

CONT_DPAD_LEFT

```
#define CONT_DPAD_LEFT (1<<6)
```

Main Dpad Left button Mask.

CONT_DPAD_RIGHT

```
#define CONT_DPAD_RIGHT (1<<7)
```

Main Dpad right button Mask.

CONT_DPAD_UP

```
#define CONT_DPAD_UP (1<<4)
```

Main Dpad Up button Mask.

CONT_START

```
#define CONT_START (1<<3)
```

Start button Mask.

CONT_X

```
#define CONT_X (1<<10)
```

X button Mask.

CONT_Y

```
#define CONT_Y (1<<9)
```

Y button Mask.

CONT_Z

```
#define CONT_Z (1<<8)
```

Z button Mask.

7.23.4 Querying Types

API for determining controller types.

Modules

- Types

  Preconfigured capabilities for standard controllers.

Functions

- int cont_is_type (const struct maple_device *cont, uint32_t type)

  Check for controller type.

7.23.4.1 Detailed Description

API for determining controller types.

The following API is for detecting between different types of standard controllers. These controllers are not identified by specific model but are instead identified solely by capabilities, so that homebrew software can remain generic and future-proof to later homebrew controllers or exotic, untested 3rd party peripherals.

Warning

Usually you want to check if a controller *supports the capabilities* of another controller, not whether it is has the *exact* same capabilities of a controller. For example, a controller that happens to come along supporting a dual analog stick but is otherwise the same layout as a standard controller would not match the standard controller type; however, it would implement its capabilities. There exist 3rd party adapters for connecting dual-analog PS2 controllers to DC which operate like this today.

Note

If you really want to hard-code the detection of a certain exact model or brand of controller, instead of basing your detection upon capabilities, check for its product_name or license within the maple_devinfo structure.

See also

cont_has_capabilities, maple_devinfo

7.23.4.2 Function Documentation

cont_is_type()

```
int cont_is_type (
            const struct maple_device * cont,
            uint32_t type )
```

Check for controller type.

Checks whether or not a controller has the *exact* capabilities associated with the given type.

Warning

Just becase a controller has all of the same capabilities of a type does not mean that it's that exact type. For example, the ASCII Pad and Arcade Stick both implement the same capabilties, although they are not the same controllers. They would be indistinguishable here, by design, so that you are able to generalize to a collection of 1st or 3rd party controllers easily.

Parameters

cont	Pointer to a Maple device structure which implements the CONTROLLER function.
type	Type identifier or capability mask the controller is expected to match

Return values

1	The controller matches the given type.
0	The controller doesn't match the given type.
-1	Invalid controller.

See also

cont_has_capabilities

7.23.4.3 Types

Preconfigured capabilities for standard controllers.

Macros

- #define CONT_TYPE_STANDARD_CONTROLLER

  Standard controller type.
- #define CONT_TYPE_DUAL_ANALOG_CONTROLLER

  Dual analog controller type.

- #define CONT_TYPE_ASCII_PAD
  ASCII fighting pad controller type.
- #define CONT_TYPE_ARCADE_STICK
  Arcade stick controller type.
- #define CONT_TYPE_TWIN_STICK
  Twin stick joystick controller type.
- #define CONT_TYPE_ASCII_MISSION_STICK
  ASCII Mission Stick controller type.
- #define CONT_TYPE_RACING_CONTROLLER
  Racing wheel/controller type.
- #define CONT_TYPE_MARACAS
  Samba De Amigo maraca controller type.
- #define CONT_TYPE_DANCE_MAT
  Dance Dance Revolution mat controller type.
- #define CONT_TYPE_FISHING_ROD
  Fishing rod controller type.
- #define CONT_TYPE_POP_N_MUSIC
  Pop'n'Music controller type.
- #define CONT_TYPE_DENSHA_DE_GO
  Densha de Go! controller type.

7.23.4.3.1 Detailed Description

Preconfigured capabilities for standard controllers.

Aggregate capability mask containing all capabilities which are implemented for a particular controller type. For example, the standard controller type is simply a combination of the following capabilities:

- Standard buttons
- Triggers
- Dpad
- Analog

Note

Because these are technically just capability masks, a type may also be passed to cont_has_capabilities() for detecting whether something has *at least* the capabilities of a type.

7.23.4.3.2 Macro Definition Documentation

CONT_TYPE_ARCADE_STICK

```
#define CONT_TYPE_ARCADE_STICK
```

Value:

```
(CONT_CAPABILITIES_STANDARD_BUTTONS | \
CONT_CAPABILITIES_EXTENDED_BUTTONS | \
CONT_CAPABILITIES_DPAD)
```

Arcade stick controller type.

CONT_TYPE_ASCII_MISSION_STICK

```
#define CONT_TYPE_ASCII_MISSION_STICK
```

Value:

```
(CONT_CAPABILITIES_STANDARD_BUTTONS | \
CONT_CAPABILITIES_DUAL_DPAD | \
CONT_CAPABILITIES_TRIGGERS | \
CONT_CAPABILITIES_ANALOG)
```

ASCII Mission Stick controller type.

CONT_TYPE_ASCII_PAD

```
#define CONT_TYPE_ASCII_PAD
```

Value:

```
(CONT_CAPABILITIES_STANDARD_BUTTONS | \
CONT_CAPABILITIES_EXTENDED_BUTTONS | \
CONT_CAPABILITIES_DPAD)
```

ASCII fighting pad controller type.

CONT_TYPE_DANCE_MAT

```
#define CONT_TYPE_DANCE_MAT
```

Value:

```
(CONT_CAPABILITY_A | \
CONT_CAPABILITY_B | \
CONT_CAPABILITY_START | \
CONT_CAPABILITIES_DPAD)
```

Dance Dance Revolution mat controller type.

CONT_TYPE_DENSHA_DE_GO

```
#define CONT_TYPE_DENSHA_DE_GO
```

Value:

```
(CONT_CAPABILITIES_STANDARD_BUTTONS | \
CONT_CAPABILITIES_EXTENDED_BUTTONS | \
CONT_CAPABILITY_D | \
CONT_CAPABILITIES_DPAD)
```

Densha de Go! controller type.

CONT_TYPE_DUAL_ANALOG_CONTROLLER

```
#define CONT_TYPE_DUAL_ANALOG_CONTROLLER
```

Value:

```
(CONT_CAPABILITIES_STANDARD_BUTTONS | \
CONT_CAPABILITIES_TRIGGERS | \
CONT_CAPABILITIES_DPAD | \
CONT_CAPABILITIES_DUAL_ANALOG)
```

Dual analog controller type.

CONT_TYPE_FISHING_ROD

```
#define CONT_TYPE_FISHING_ROD
```

Value:

```
(CONT_CAPABILITIES_STANDARD_BUTTONS | \
CONT_CAPABILITIES_DPAD | \
CONT_CAPABILITIES_TRIGGERS | \
CONT_CAPABILITIES_DUAL_ANALOG)
```

Fishing rod controller type.

CONT_TYPE_MARACAS

```
#define CONT_TYPE_MARACAS
```

Value:

```
(CONT_CAPABILITY_A | \
CONT_CAPABILITY_B | \
CONT_CAPABILITY_D | \
CONT_CAPABILITY_START | \
CONT_CAPABILITIES_EXTENDED_BUTTONS | \
CONT_CAPABILITIES_DUAL_ANALOG)
```

Samba De Amigo maraca controller type.

CONT_TYPE_POP_N_MUSIC

```
#define CONT_TYPE_POP_N_MUSIC
```

Value:

```
(CONT_CAPABILITIES_STANDARD_BUTTONS | \
CONT_CAPABILITY_C | \
CONT_CAPABILITIES_DPAD)
```

Pop'n'Music controller type.

CONT_TYPE_RACING_CONTROLLER

```
#define CONT_TYPE_RACING_CONTROLLER
```

Value:

```
(CONT_CAPABILITY_DPAD_UP | \
CONT_CAPABILITY_DPAD_DOWN | \
CONT_CAPABILITY_A | \
CONT_CAPABILITY_B | \
CONT_CAPABILITY_START | \
CONT_CAPABILITIES_TRIGGERS | \
CONT_CAPABILITY_ANALOG_X \
CONT_CAPABILITIES_SECONDARY_ANALOG)
```

Racing wheel/controller type.

CONT_TYPE_STANDARD_CONTROLLER

```
#define CONT_TYPE_STANDARD_CONTROLLER
```

Value:

```
(CONT_CAPABILITIES_STANDARD_BUTTONS | \
CONT_CAPABILITIES_TRIGGERS | \
CONT_CAPABILITIES_DPAD | \
CONT_CAPABILITIES_ANALOG)
```

Standard controller type.

CONT_TYPE_TWIN_STICK

```
#define CONT_TYPE_TWIN_STICK
```

Value:

```
(CONT_CAPABILITIES_STANDARD_BUTTONS | \
CONT_CAPABILITIES_EXTENDED_BUTTONS | \
CONT_CAPABILITY_D | \
CONT_CAPABILITIES_DUAL_DPAD)
```

Twin stick joystick controller type.

7.24 Dimensions of the Bios Font

Macros

- #define BFONT_THIN_WIDTH 12

  Width of Thin Font (ISO8859_1, half-JP)
- #define BFONT_WIDE_WIDTH BFONT_THIN_WIDTH ∗ 2

  Width of Wide Font (full-JP)
- #define BFONT_HEIGHT 24

  Height of All Fonts.

7.24.1 Detailed Description

7.24.2 Macro Definition Documentation

BFONT_HEIGHT

```
#define BFONT_HEIGHT 24
```

Height of All Fonts.

BFONT_THIN_WIDTH

```
#define BFONT_THIN_WIDTH 12
```

Width of Thin Font (ISO8859_1, half-JP)

BFONT_WIDE_WIDTH

```
#define BFONT_WIDE_WIDTH BFONT_THIN_WIDTH * 2
```

Width of Wide Font (full-JP)

7.25 Dreamcast-specific initialization flags.

Dreamcast-specific KOS_INIT Exports.

Dreamcast-specific KOS_INIT Exports.

This macro contains a list of all of the possible DC-specific exported functions based on their associated initialization flags.

Note

This is not typically used directly and is instead included within the top-level architecture-independent KOS_INIT_FLAGS() macro.

Parameters

flags	Parts of KOS to initialize.

See also

KOS_INIT_FLAGS()

7.26 ELF architecture types

Macros

- #define EM_386 3

  x86 (IA32)
- #define EM_ARM 40

  ARM.
- #define EM_SH 42

  SuperH.

7.26.1 Detailed Description

These are the various architectures that we might care about for ELF files.

7.26.2 Macro Definition Documentation

EM_386

```
#define EM_386 3
```

x86 (IA32)

EM_ARM

```
#define EM_ARM 40
```

ARM.

EM_SH

```
#define EM_SH 42
```

SuperH.

7.27 ELF relocation types

Macros

- #define R_SH_DIR32 1

  SuperH: Rel = Symbol + Addend.
- #define R_386_32 1

  x86: Rel = Symbol + Addend
- #define R_386_PC32 2

  x86: Rel = Symbol + Addend - Value

7.27.1 Detailed Description

These define the types of operations that can be done to calculate relocations within ELF files.

7.27.2 Macro Definition Documentation

R_386_32

```
#define R_386_32 1
```

x86: Rel = Symbol + Addend

R_386_PC32

```
#define R_386_PC32 2
```

x86: Rel = Symbol + Addend - Value

R_SH_DIR32

```
#define R_SH_DIR32 1
```

SuperH: Rel = Symbol + Addend.

7.28 Enable or disable PVR depth writes

Macros

- #define PVR_DEPTHWRITE_ENABLE 0
  Update the Z value.
- #define PVR_DEPTHWRITE_DISABLE 1
  Do not update the Z value.

7.28.1 Detailed Description

7.28.2 Macro Definition Documentation

PVR_DEPTHWRITE_DISABLE

```
#define PVR_DEPTHWRITE_DISABLE 1
```

Do not update the Z value.

PVR_DEPTHWRITE_ENABLE

```
#define PVR_DEPTHWRITE_ENABLE 0
```

Update the Z value.

7.29 Enable or disable PVR mipmap processing

Macros

- #define PVR_MIPMAP_DISABLE 0

  Disable mipmap processing.
- #define PVR_MIPMAP_ENABLE 1

  Enable mipmap processing.

7.29.1 Detailed Description

7.29.2 Macro Definition Documentation

PVR_MIPMAP_DISABLE

```
#define PVR_MIPMAP_DISABLE 0
```

Disable mipmap processing.

PVR_MIPMAP_ENABLE

```
#define PVR_MIPMAP_ENABLE 1
```

Enable mipmap processing.

7.30 Enable or disable U/V flipping on the PVR

Macros

- #define PVR_UVFLIP_NONE 0

  No flipped coordinates.
- #define PVR_UVFLIP_V 1

  Flip V only.
- #define PVR_UVFLIP_U 2

  Flip U only.
- #define PVR_UVFLIP_UV 3

  Flip U and V.

7.30.1 Detailed Description

These flags determine what happens when U/V coordinate values exceed 1.0. In any of the flipped cases, the specified coordinate value will flip around after 1.0, essentially mirroring the image. So, if you displayed an image with a U coordinate of 0.0 on the left hand side and 2.0 on the right hand side with U flipping turned on, you'd have an image that was displayed twice as if mirrored across the middle. This mirroring behavior happens at every unit boundary (so at 2.0 it returns to normal, at 3.0 it flips, etc).

The default case is to disable mirroring. In addition, clamping of the U/V coordinates by PVR_UVCLAMP_U, PVR_↩ UVCLAMP_V, or PVR_UVCLAMP_UV will disable the mirroring behavior.

7.30.2 Macro Definition Documentation

PVR_UVFLIP_NONE

```
#define PVR_UVFLIP_NONE 0
```

No flipped coordinates.

PVR_UVFLIP_U

```
#define PVR_UVFLIP_U 2
```

Flip U only.

PVR_UVFLIP_UV

```
#define PVR_UVFLIP_UV 3
```

Flip U and V.

PVR_UVFLIP_V

```
#define PVR_UVFLIP_V 1
```

Flip V only.

7.31 Enable or disable alpha blending

Macros

- #define PVR_ALPHA_DISABLE 0

  Disable alpha blending.
- #define PVR_ALPHA_ENABLE 1

  Enable alpha blending.

7.31.1 Detailed Description

This causes the alpha value in the vertex color to be paid attention to. It really only makes sense to enable this for translucent or punch-thru polys.

7.31.2 Macro Definition Documentation

PVR_ALPHA_DISABLE

```
#define PVR_ALPHA_DISABLE 0
```

Disable alpha blending.

PVR_ALPHA_ENABLE

```
#define PVR_ALPHA_ENABLE 1
```

Enable alpha blending.

7.32 Enable or disable blending

Macros

- #define PVR_BLEND_DISABLE 0
  Disable blending.
- #define PVR_BLEND_ENABLE 1
  Enable blending.

7.32.1 Detailed Description

7.32.2 Macro Definition Documentation

PVR_BLEND_DISABLE

```
#define PVR_BLEND_DISABLE 0
```

Disable blending.

PVR_BLEND_ENABLE

```
#define PVR_BLEND_ENABLE 1
```

Enable blending.

7.33 Enable or disable clamping of U/V on the PVR

Macros

- #define PVR_UVCLAMP_NONE 0

  Disable clamping.
- #define PVR_UVCLAMP_V 1

  Clamp V only.
- #define PVR_UVCLAMP_U 2

  Clamp U only.
- #define PVR_UVCLAMP_UV 3

  Clamp U and V.

7.33.1 Detailed Description

These flags determine whether clamping will be applied to U/V coordinate values that exceed 1.0. If enabled, these modes will explicitly override the flip/mirroring modes (PVR_UVFLIP_U, PVR_UVFLIP_V, and PVR_UVFLIP_UV), and will instead ensure that the coordinate(s) in question never exceed 1.0.

7.33.2 Macro Definition Documentation

PVR_UVCLAMP_NONE

```
#define PVR_UVCLAMP_NONE 0
```

Disable clamping.

PVR_UVCLAMP_U

```
#define PVR_UVCLAMP_U 2
```

Clamp U only.

PVR_UVCLAMP_UV

```
#define PVR_UVCLAMP_UV 3
```

Clamp U and V.

PVR_UVCLAMP_V

```
#define PVR_UVCLAMP_V 1
```

Clamp V only.

7.34 Enable or disable color clamping

Macros

- #define PVR_CLRCLAMP_DISABLE 0
  Disable color clamping.
- #define PVR_CLRCLAMP_ENABLE 1
  Enable color clamping.

7.34.1 Detailed Description

Enabling color clamping will clamp colors between the minimum and maximum values before any sort of fog processing.

7.34.2 Macro Definition Documentation

PVR_CLRCLAMP_DISABLE

```
#define PVR_CLRCLAMP_DISABLE 0
```

Disable color clamping.

PVR_CLRCLAMP_ENABLE

```
#define PVR_CLRCLAMP_ENABLE 1
```

Enable color clamping.

7.35 Enable or disable modifier effects

Macros

- #define PVR_MODIFIER_DISABLE 0
  Disable modifier effects.
- #define PVR_MODIFIER_ENABLE 1
  Enable modifier effects.

7.35.1 Detailed Description

7.35.2 Macro Definition Documentation

PVR_MODIFIER_DISABLE

```
#define PVR_MODIFIER_DISABLE 0
```

Disable modifier effects.

PVR_MODIFIER_ENABLE

```
#define PVR_MODIFIER_ENABLE 1
```

Enable modifier effects.

7.36 Enable or disable offset color

Macros

- #define PVR_SPECULAR_DISABLE 0
  Disable offset colors.
- #define PVR_SPECULAR_ENABLE 1
  Enable offset colors.

7.36.1 Detailed Description

Enabling offset color calculation allows for "specular" like effects on a per-vertex basis, by providing an additive color in the calculation of the final pixel colors. In vertex types with a "oargb" parameter, that's what it is for.

Note that this must be enabled for bumpmap polygons in order to allow you to specify the parameters in the oargb field of the vertices.

7.36.2 Macro Definition Documentation

PVR_SPECULAR_DISABLE

```
#define PVR_SPECULAR_DISABLE 0
```

Disable offset colors.

PVR_SPECULAR_ENABLE

```
#define PVR_SPECULAR_ENABLE 1
```

Enable offset colors.

7.37 Enable or disable texture alpha blending

Macros

- #define PVR_TXRALPHA_ENABLE 0
  Enable alpha blending.
- #define PVR_TXRALPHA_DISABLE 1
  Disable alpha blending.

7.37.1 Detailed Description

This causes the alpha value in the texel color to be paid attention to. It really only makes sense to enable this for translucent or punch-thru polys.

7.37.2 Macro Definition Documentation

PVR_TXRALPHA_DISABLE

```
#define PVR_TXRALPHA_DISABLE 1
```

Disable alpha blending.

PVR_TXRALPHA_ENABLE

```
#define PVR_TXRALPHA_ENABLE 0
```

Enable alpha blending.

7.38 Enable or disable texturing on polygons

Macros

- #define PVR_TEXTURE_DISABLE 0
  Disable texturing.
- #define PVR_TEXTURE_ENABLE 1
  Enable texturing.

7.38.1 Detailed Description

7.38.2 Macro Definition Documentation

PVR_TEXTURE_DISABLE

```
#define PVR_TEXTURE_DISABLE 0
```

Disable texturing.

PVR_TEXTURE_ENABLE

```
#define PVR_TEXTURE_ENABLE 1
```

Enable texturing.

7.39 Error values for the flashrom_get_block() function

Macros

- #define FLASHROM_ERR_NONE 0
  Success.
- #define FLASHROM_ERR_NOT_FOUND -1
  Block not found.
- #define FLASHROM_ERR_NO_PARTITION -2
  Partition not found.
- #define FLASHROM_ERR_READ_PART -3
  Error reading partition.
- #define FLASHROM_ERR_BAD_MAGIC -4
  Invalid block magic.
- #define FLASHROM_ERR_BOGUS_PART -5
  Bogus partition size.
- #define FLASHROM_ERR_NOMEM -6
  Memory allocation failure.
- #define FLASHROM_ERR_READ_BITMAP -7
  Error reading bitmap.
- #define FLASHROM_ERR_EMPTY_PART -8
  Empty partition.
- #define FLASHROM_ERR_READ_BLOCK -9
  Error reading block.

7.39.1 Detailed Description

7.39.2 Macro Definition Documentation

FLASHROM_ERR_BAD_MAGIC

```
#define FLASHROM_ERR_BAD_MAGIC -4
```

Invalid block magic.

FLASHROM_ERR_BOGUS_PART

```
#define FLASHROM_ERR_BOGUS_PART -5
```

Bogus partition size.

FLASHROM_ERR_EMPTY_PART

```
#define FLASHROM_ERR_EMPTY_PART -8
```

Empty partition.

FLASHROM_ERR_NO_PARTITION

```
#define FLASHROM_ERR_NO_PARTITION -2
```

Partition not found.

FLASHROM_ERR_NOMEM

```
#define FLASHROM_ERR_NOMEM -6
```

Memory allocation failure.

FLASHROM_ERR_NONE

```
#define FLASHROM_ERR_NONE 0
```

Success.

FLASHROM_ERR_NOT_FOUND

```
#define FLASHROM_ERR_NOT_FOUND -1
```

Block not found.

FLASHROM_ERR_READ_BITMAP

```
#define FLASHROM_ERR_READ_BITMAP -7
```

Error reading bitmap.

FLASHROM_ERR_READ_BLOCK

```
#define FLASHROM_ERR_READ_BLOCK -9
```

Error reading block.

FLASHROM_ERR_READ_PART

```
#define FLASHROM_ERR_READ_PART -3
```

Error reading partition.

7.40 Error values for the h_errno variable

Macros

- #define HOST_NOT_FOUND 1

  Hostname not found.
- #define TRY_AGAIN 2

  Try the request again.
- #define NO_RECOVERY 3

  A non-recoverable error.
- #define NO_DATA 4

  Host found, but no data.

7.40.1 Detailed Description

These are the possible values for h_errno, indicating errors returns from the gethostbyname() function.

7.40.2 Macro Definition Documentation

HOST_NOT_FOUND

```
#define HOST_NOT_FOUND 1
```

Hostname not found.

NO_DATA

```
#define NO_DATA 4
```

Host found, but no data.

NO_RECOVERY

```
#define NO_RECOVERY 3
```

A non-recoverable error.

TRY_AGAIN

```
#define TRY_AGAIN 2
```

Try the request again.

7.41 Errors for the getaddrinfo() function

Macros

- #define EAI_AGAIN 1
  Try the request again.
- #define EAI_BADFLAGS 2
  Invalid hint flags.
- #define EAI_FAIL 3
  A non-recoverable error.
- #define EAI_FAMILY 4
  Invalid address family.
- #define EAI_MEMORY 5
  Memory allocation error.
- #define EAI_NONAME 6
  Hostname not found.
- #define EAI_SERVICE 7
  Invalid service value.
- #define EAI_SOCKTYPE 8
  Invalid socket type.
- #define EAI_SYSTEM 9
  System error, check errno.
- #define EAI_OVERFLOW 10
  Argument buffer overflow.

7.41.1 Detailed Description

These are the possible error return values from the getaddrinfo() function.

7.41.2 Macro Definition Documentation

EAI_AGAIN

```
#define EAI_AGAIN 1
```

Try the request again.

EAI_BADFLAGS

```
#define EAI_BADFLAGS 2
```

Invalid hint flags.

EAI_FAIL

```
#define EAI_FAIL 3
```

A non-recoverable error.

EAI_FAMILY

```
#define EAI_FAMILY 4
```

Invalid address family.

EAI_MEMORY

```
#define EAI_MEMORY 5
```

Memory allocation error.

EAI_NONAME

```
#define EAI_NONAME 6
```

Hostname not found.

EAI_OVERFLOW

```
#define EAI_OVERFLOW 10
```

Argument buffer overflow.

EAI_SERVICE

```
#define EAI_SERVICE 7
```

Invalid service value.

EAI_SOCKTYPE

```
#define EAI_SOCKTYPE 8
```

Invalid socket type.

EAI_SYSTEM

```
#define EAI_SYSTEM 9
```

System error, check errno.

7.42 Events for the poll() function

Macros

- #define POLLRDNORM (1 << 0)
  Normal data may be read.
- #define POLLRDBAND (1 << 1)
  Priority data may be read.
- #define POLLPRI (1 << 2)
  High-priority data may be read.
- #define POLLOUT (1 << 3)
  Normal data may be written.
- #define POLLWRNORM POLLOUT
  Normal data may be written.
- #define POLLWRBAND (1 << 4)
  Priority data may be written.
- #define POLLERR (1 << 5)
  Error has occurred (revents only)
- #define POLLHUP (1 << 6)
  Peer disconnected (revents only)
- #define POLLNVAL (1 << 7)
  Invalid fd (revents only)
- #define POLLIN (POLLRDNORM | POLLRDBAND)
  Data other than high-priority data may be read.

7.42.1 Detailed Description

These are the events that can be set in the events or revents fields of the struct pollfd.

7.42.2 Macro Definition Documentation

POLLERR

```
#define POLLERR (1 << 5)
```

Error has occurred (revents only)

POLLHUP

```
#define POLLHUP (1 << 6)
```

Peer disconnected (revents only)

POLLIN

```
#define POLLIN (POLLRDNORM | POLLRDBAND)
```

Data other than high-priority data may be read.

POLLNVAL

```
#define POLLNVAL (1 << 7)
```

Invalid fd (revents only)

POLLOUT

```
#define POLLOUT (1 << 3)
```

Normal data may be written.

POLLPRI

```
#define POLLPRI (1 << 2)
```

High-priority data may be read.

POLLRDBAND

```
#define POLLRDBAND (1 << 1)
```

Priority data may be read.

POLLRDNORM

```
#define POLLRDNORM (1 << 0)
```

Normal data may be read.

POLLWRBAND

```
#define POLLWRBAND (1 << 4)
```

Priority data may be written.

POLLWRNORM

```
#define POLLWRNORM POLLOUT
```

Normal data may be written.

7.43 Eyecatch types.

Macros

- #define VMUPKG_EC_NONE 0
  No eyecatch.
- #define VMUPKG_EC_16BIT 1
  16-bit ARGB4444
- #define VMUPKG_EC_256COL 2
  256-color palette
- #define VMUPKG_EC_16COL 3
  16-color palette

7.43.1 Detailed Description

All eyecatches are 72x56, but the pixel format is variable. Note that in all of the cases which use a palette, the palette entries are in ARGB4444 format and come directly before the pixel data itself.

7.43.2 Macro Definition Documentation

VMUPKG_EC_16BIT

```
#define VMUPKG_EC_16BIT 1
```

16-bit ARGB4444

VMUPKG_EC_16COL

```
#define VMUPKG_EC_16COL 3
```

16-color palette

VMUPKG_EC_256COL

```
#define VMUPKG_EC_256COL 2
```

256-color palette

VMUPKG_EC_NONE

```
#define VMUPKG_EC_NONE 0
```

No eyecatch.

7.44 File open modes

Macros

- #define O_MODE_MASK 0x0f
  Mask for mode numbers.
- #define O_ASYNC 0x0200
  Open for asynchronous I/O.
- #define O_DIR 0x1000
  Open as directory.
- #define O_META 0x2000
  Open as metadata.

7.44.1 Detailed Description

7.44.2 Macro Definition Documentation

O_ASYNC

```
#define O_ASYNC 0x0200
```

Open for asynchronous I/O.

O_DIR

```
#define O_DIR 0x1000
```

Open as directory.

O_META

```
#define O_META 0x2000
```

Open as metadata.

O_MODE_MASK

```
#define O_MODE_MASK 0x0f
```

Mask for mode numbers.

7.45 Flags for ai_flags in struct addrinfo

Macros

- #define AI_PASSIVE 0x00000001
  Address intended for bind().
- #define AI_CANONNAME 0x00000002
  Request canonical name.
- #define AI_NUMERICHOST 0x00000004
  Inhibit host resolution.
- #define AI_NUMERICSERV 0x00000008
  Inhibit service resolution.
- #define AI_V4MAPPED 0x00000010
  Return v4-mapped IPv6 addrs.
- #define AI_ALL 0x00000020
  Query for both IPv4 and IPv6.
- #define AI_ADDRCONFIG 0x00000040
  Only query for IPv4/IPv6 addrs the system has a valid addr.

7.45.1 Detailed Description

These are the flags that can be set in the ai_flags field of struct addrinfo. These values can be bitwise ORed together.

Currently only AI_PASSIVE is actually supported by the getaddrinfo() function.

7.45.2 Macro Definition Documentation

AI_ADDRCONFIG

```
#define AI_ADDRCONFIG 0x00000040
```

Only query for IPv4/IPv6 addrs the system has a valid addr.

AI_ALL

```
#define AI_ALL 0x00000020
```

Query for both IPv4 and IPv6.

AI_CANONNAME

```
#define AI_CANONNAME 0x00000002
```

Request canonical name.

AI_NUMERICHOST

```
#define AI_NUMERICHOST 0x00000004
```

Inhibit host resolution.

AI_NUMERICSERV

```
#define AI_NUMERICSERV 0x00000008
```

Inhibit service resolution.

AI_PASSIVE

```
#define AI_PASSIVE 0x00000001
```

Address intended for bind().

AI_V4MAPPED

```
#define AI_V4MAPPED 0x00000010
```

Return v4-mapped IPv6 addrs.

7.46 Flags for netif_t

Macros

- #define NETIF_NO_FLAGS 0x00000000
  No flags set.
- #define NETIF_REGISTERED 0x00000001
  Is it registered?
- #define NETIF_DETECTED 0x00000002
  Is it detected?
- #define NETIF_INITIALIZED 0x00000004
  Has it been initialized?
- #define NETIF_RUNNING 0x00000008
  Has start() been called?
- #define NETIF_PROMISC 0x00010000
  Promiscuous mode.
- #define NETIF_NEEDSPOLL 0x01000000
  Needs to be polled for input.
- #define NETIF_NOETH 0x10000000
  Does not use ethernet.

7.46.1 Detailed Description

7.46.2 Macro Definition Documentation

NETIF_DETECTED

```
#define NETIF_DETECTED 0x00000002
```

Is it detected?

NETIF_INITIALIZED

```
#define NETIF_INITIALIZED 0x00000004
```

Has it been initialized?

NETIF_NEEDSPOLL

```
#define NETIF_NEEDSPOLL 0x01000000
```

Needs to be polled for input.

NETIF_NO_FLAGS

```
#define NETIF_NO_FLAGS 0x00000000
```

No flags set.

NETIF_NOETH

```
#define NETIF_NOETH 0x10000000
```

Does not use ethernet.

NETIF_PROMISC

```
#define NETIF_PROMISC 0x00010000
```

Promiscuous mode.

NETIF_REGISTERED

```
#define NETIF_REGISTERED 0x00000001
```

Is it registered?

NETIF_RUNNING

```
#define NETIF_RUNNING 0x00000008
```

Has start() been called?

7.47 Flags for the field in vid_mode_t.

Macros

- #define VID_INTERLACE 0x00000001
  Interlaced display.
- #define VID_LINEDOUBLE 0x00000002
  Display each scanline twice.
- #define VID_PIXELDOUBLE 0x00000004
  Display each pixel twice.
- #define VID_PAL 0x00000008
  50Hz refresh rate, if not VGA

7.47.1 Detailed Description

These flags indicate various things related to the modes for a vid_mode_t.

7.47.2 Macro Definition Documentation

VID_INTERLACE

```
#define VID_INTERLACE 0x00000001
```

Interlaced display.

VID_LINEDOUBLE

```
#define VID_LINEDOUBLE 0x00000002
```

Display each scanline twice.

VID_PAL

```
#define VID_PAL 0x00000008
```

50Hz refresh rate, if not VGA

VID_PIXELDOUBLE

```
#define VID_PIXELDOUBLE 0x00000004
```

Display each pixel twice.

7.48 Flags for the flashrom_ispcfg_t struct

Macros

- #define FLASHROM_ISP_DIAL_AREACODE (1 << 0)

  Dial area code before number.
- #define FLASHROM_ISP_USE_PROXY (1 << 1)

  Proxy enabled.
- #define FLASHROM_ISP_PULSE_DIAL (1 << 2)

  Pulse dialing (instead of tone)
- #define FLASHROM_ISP_BLIND_DIAL (1 << 3)

  Blind dial (don't wait for tone)

7.48.1 Detailed Description

The flags field of the flashrom_ispcfg_t will have some combination of these ORed together to represent what settings were set.

7.48.2 Macro Definition Documentation

FLASHROM_ISP_BLIND_DIAL

```
#define FLASHROM_ISP_BLIND_DIAL (1 << 3)
```

Blind dial (don't wait for tone)

FLASHROM_ISP_DIAL_AREACODE

```
#define FLASHROM_ISP_DIAL_AREACODE (1 << 0)
```

Dial area code before number.

FLASHROM_ISP_PULSE_DIAL

```
#define FLASHROM_ISP_PULSE_DIAL (1 << 2)
```

Pulse dialing (instead of tone)

FLASHROM_ISP_USE_PROXY

```
#define FLASHROM_ISP_USE_PROXY (1 << 1)
```

Proxy enabled.

7.49 IPv4 protocol level options

Macros

- #define IP_TTL 24

  TTL for unicast (get/set)

7.49.1 Detailed Description

These are the various socket-level optoins that can be accessed with the setsockopt() and getsockopt() fnctions for the IPPROTO_IP level value.

As there isn't really a full standard list of these defined in the SUS (apparently), only ones that we support are listed here.

See also

Socket-level options

IPv6 protocol level options

UDP protocol level options

UDP-Lite protocol level options

TCP protocol level options

7.49.2 Macro Definition Documentation

IP_TTL

```
#define IP_TTL 24
```

TTL for unicast (get/set)

7.50 IPv6 protocol level options

Macros

- #define IPV6_JOIN_GROUP 17
  Join a multicast group (set)
- #define IPV6_LEAVE_GROUP 18
  Leave a multicast group (set)
- #define IPV6_MULTICAST_HOPS 19
  Hop limit for multicast (get/set)
- #define IPV6_MULTICAST_IF 20
  Multicast interface (get/set)
- #define IPV6_MULTICAST_LOOP 21
  Multicasts loopback (get/set)
- #define IPV6_UNICAST_HOPS 22
  Hop limit for unicast (get/set)
- #define IPV6_V6ONLY 23
  IPv6 only – no IPv4 (get/set)

7.50.1 Detailed Description

These are the various socket-level options that can be accessed with the setsockopt() and getsockopt() functions for the IPPROTO_IPV6 level value.

Not all of these are currently supported, but they are listed for completeness.

See also

- Socket-level options
- IPv4 protocol level options
- UDP protocol level options
- UDP-Lite protocol level options
- TCP protocol level options

7.50.2 Macro Definition Documentation

IPV6_JOIN_GROUP

```
#define IPV6_JOIN_GROUP 17
```

Join a multicast group (set)

IPV6_LEAVE_GROUP

```
#define IPV6_LEAVE_GROUP 18
```

Leave a multicast group (set)

IPV6_MULTICAST_HOPS

```
#define IPV6_MULTICAST_HOPS 19
```

Hop limit for multicast (get/set)

IPV6_MULTICAST_IF

```
#define IPV6_MULTICAST_IF 20
```

Multicast interface (get/set)

IPV6_MULTICAST_LOOP

```
#define IPV6_MULTICAST_LOOP 21
```

Multicasts loopback (get/set)

IPV6_UNICAST_HOPS

```
#define IPV6_UNICAST_HOPS 22
```

Hop limit for unicast (get/set)

IPV6_V6ONLY

```
#define IPV6_V6ONLY 23
```

IPv6 only – no IPv4 (get/set)

7.51 Image format types

Macros

- #define KOS_IMG_FMT_NONE 0x00
  Undefined or uninitialized format.
- #define KOS_IMG_FMT_RGB888 0x01
  24-bpp interleaved R/G/B bytes.
- #define KOS_IMG_FMT_ARGB8888 0x02
  32-bpp interleaved A/R/G/B bytes.
- #define KOS_IMG_FMT_RGB565 0x03
  16-bpp interleaved R (5 bits), G (6 bits), B (5 bits).
- #define KOS_IMG_FMT_ARGB4444 0x04
  16-bpp interleaved A/R/G/B (4 bits each).
- #define KOS_IMG_FMT_ARGB1555 0x05
  16-bpp interleaved A (1 bit), R (5 bits), G (5 bits), B (5 bits).
- #define KOS_IMG_FMT_PAL4BPP 0x06
  Paletted, 4 bits per pixel (16 colors).
- #define KOS_IMG_FMT_PAL8BPP 0x07
  Paletted, 8 bits per pixel (256 colors).
- #define KOS_IMG_FMT_YUV422 0x08
  8-bit Y (4 bits), U (2 bits), V (2 bits).
- #define KOS_IMG_FMT_BGR565 0x09
  15-bpp interleaved B (5 bits), G (6 bits), R (5 bits).
- #define KOS_IMG_FMT_RGBA8888 0x10
  32-bpp interleaved R/G/B/A bytes.
- #define KOS_IMG_FMT_MASK 0xff
  Basic format mask (not an actual format value).
- #define KOS_IMG_INVERTED_X 0x0100
  X axis of image data is inverted (stored right to left).
- #define KOS_IMG_INVERTED_Y 0x0200
  Y axis of image data is inverted (stored bottom to top).
- #define KOS_IMG_NOT_OWNER 0x0400
  The image is not the owner of the image data buffer.

7.51.1 Detailed Description

This is the list of platform-independent image types that can be used as the lower-half of the fmt value for a kos_img_t.

7.51.2 Macro Definition Documentation

KOS_IMG_FMT_ARGB1555

```
#define KOS_IMG_FMT_ARGB1555 0x05
```

16-bpp interleaved A (1 bit), R (5 bits), G (5 bits), B (5 bits).

Note

This can also be used for RGB555 (with the top bit ignored).

KOS_IMG_FMT_ARGB4444

```
#define KOS_IMG_FMT_ARGB4444 0x04
```

16-bpp interleaved A/R/G/B (4 bits each).

KOS_IMG_FMT_ARGB8888

```
#define KOS_IMG_FMT_ARGB8888 0x02
```

32-bpp interleaved A/R/G/B bytes.

KOS_IMG_FMT_BGR565

```
#define KOS_IMG_FMT_BGR565 0x09
```

15-bpp interleaved B (5 bits), G (6 bits), R (5 bits).

KOS_IMG_FMT_MASK

```
#define KOS_IMG_FMT_MASK 0xff
```

Basic format mask (not an actual format value).

KOS_IMG_FMT_NONE

```
#define KOS_IMG_FMT_NONE 0x00
```

Undefined or uninitialized format.

KOS_IMG_FMT_PAL4BPP

```
#define KOS_IMG_FMT_PAL4BPP 0x06
```

Paletted, 4 bits per pixel (16 colors).

KOS_IMG_FMT_PAL8BPP

```
#define KOS_IMG_FMT_PAL8BPP 0x07
```

Paletted, 8 bits per pixel (256 colors).

KOS_IMG_FMT_RGB565

```
#define KOS_IMG_FMT_RGB565 0x03
```

16-bpp interleaved R (5 bits), G (6 bits), B (5 bits).

KOS_IMG_FMT_RGB888

```
#define KOS_IMG_FMT_RGB888 0x01
```

24-bpp interleaved R/G/B bytes.

KOS_IMG_FMT_RGBA8888

```
#define KOS_IMG_FMT_RGBA8888 0x10
```

32-bpp interleaved R/G/B/A bytes.

KOS_IMG_FMT_YUV422

```
#define KOS_IMG_FMT_YUV422 0x08
```

8-bit Y (4 bits), U (2 bits), V (2 bits).

KOS_IMG_INVERTED_X

```
#define KOS_IMG_INVERTED_X 0x0100
```

X axis of image data is inverted (stored right to left).

KOS_IMG_INVERTED_Y

```
#define KOS_IMG_INVERTED_Y 0x0200
```

Y axis of image data is inverted (stored bottom to top).

KOS_IMG_NOT_OWNER

```
#define KOS_IMG_NOT_OWNER 0x0400
```

The image is not the owner of the image data buffer.

This generally implies that the image data is stored in ROM and thus cannot be freed.

7.52 Keyboard LEDs

Macros

- #define KBD_LED_NUMLOCK (1<<0)
- #define KBD_LED_CAPSLOCK (1<<1)
- #define KBD_LED_SCRLOCK (1<<2)

7.52.1 Detailed Description

This is the LEDs that can be turned on and off on the keyboard. This list may not be exhaustive. Think of these sorta like an extension of the modifiers list.

7.52.2 Macro Definition Documentation

KBD_LED_CAPSLOCK

```
#define KBD_LED_CAPSLOCK (1<<1)
```

KBD_LED_NUMLOCK

```
#define KBD_LED_NUMLOCK (1<<0)
```

KBD_LED_SCRLOCK

```
#define KBD_LED_SCRLOCK (1<<2)
```

7.53 Keyboard keys

Macros

- #define KBD_KEY_NONE 0x00
- #define KBD_KEY_ERROR 0x01
- #define KBD_KEY_ERR2 0x02
- #define KBD_KEY_ERR3 0x03
- #define KBD_KEY_A 0x04
- #define KBD_KEY_B 0x05
- #define KBD_KEY_C 0x06
- #define KBD_KEY_D 0x07
- #define KBD_KEY_E 0x08
- #define KBD_KEY_F 0x09
- #define KBD_KEY_G 0x0a
- #define KBD_KEY_H 0x0b
- #define KBD_KEY_I 0x0c
- #define KBD_KEY_J 0x0d
- #define KBD_KEY_K 0x0e
- #define KBD_KEY_L 0x0f
- #define KBD_KEY_M 0x10
- #define KBD_KEY_N 0x11
- #define KBD_KEY_O 0x12
- #define KBD_KEY_P 0x13
- #define KBD_KEY_Q 0x14
- #define KBD_KEY_R 0x15
- #define KBD_KEY_S 0x16
- #define KBD_KEY_T 0x17
- #define KBD_KEY_U 0x18
- #define KBD_KEY_V 0x19
- #define KBD_KEY_W 0x1a
- #define KBD_KEY_X 0x1b
- #define KBD_KEY_Y 0x1c
- #define KBD_KEY_Z 0x1d
- #define KBD_KEY_1 0x1e
- #define KBD_KEY_2 0x1f
- #define KBD_KEY_3 0x20
- #define KBD_KEY_4 0x21
- #define KBD_KEY_5 0x22
- #define KBD_KEY_6 0x23
- #define KBD_KEY_7 0x24
- #define KBD_KEY_8 0x25
- #define KBD_KEY_9 0x26
- #define KBD_KEY_0 0x27
- #define KBD_KEY_ENTER 0x28

- #define KBD_KEY_ESCAPE 0x29
- #define KBD_KEY_BACKSPACE 0x2a
- #define KBD_KEY_TAB 0x2b
- #define KBD_KEY_SPACE 0x2c
- #define KBD_KEY_MINUS 0x2d
- #define KBD_KEY_PLUS 0x2e
- #define KBD_KEY_LBRACKET 0x2f
- #define KBD_KEY_RBRACKET 0x30
- #define KBD_KEY_BACKSLASH 0x31
- #define KBD_KEY_SEMICOLON 0x33
- #define KBD_KEY_QUOTE 0x34
- #define KBD_KEY_TILDE 0x35
- #define KBD_KEY_COMMA 0x36
- #define KBD_KEY_PERIOD 0x37
- #define KBD_KEY_SLASH 0x38
- #define KBD_KEY_CAPSLOCK 0x39
- #define KBD_KEY_F1 0x3a
- #define KBD_KEY_F2 0x3b
- #define KBD_KEY_F3 0x3c
- #define KBD_KEY_F4 0x3d
- #define KBD_KEY_F5 0x3e
- #define KBD_KEY_F6 0x3f
- #define KBD_KEY_F7 0x40
- #define KBD_KEY_F8 0x41
- #define KBD_KEY_F9 0x42
- #define KBD_KEY_F10 0x43
- #define KBD_KEY_F11 0x44
- #define KBD_KEY_F12 0x45
- #define KBD_KEY_PRINT 0x46
- #define KBD_KEY_SCRLOCK 0x47
- #define KBD_KEY_PAUSE 0x48
- #define KBD_KEY_INSERT 0x49
- #define KBD_KEY_HOME 0x4a
- #define KBD_KEY_PGUP 0x4b
- #define KBD_KEY_DEL 0x4c
- #define KBD_KEY_END 0x4d
- #define KBD_KEY_PGDOWN 0x4e
- #define KBD_KEY_RIGHT 0x4f
- #define KBD_KEY_LEFT 0x50
- #define KBD_KEY_DOWN 0x51
- #define KBD_KEY_UP 0x52
- #define KBD_KEY_PAD_NUMLOCK 0x53
- #define KBD_KEY_PAD_DIVIDE 0x54
- #define KBD_KEY_PAD_MULTIPLY 0x55
- #define KBD_KEY_PAD_MINUS 0x56
- #define KBD_KEY_PAD_PLUS 0x57
- #define KBD_KEY_PAD_ENTER 0x58
- #define KBD_KEY_PAD_1 0x59
- #define KBD_KEY_PAD_2 0x5a
- #define KBD_KEY_PAD_3 0x5b
- #define KBD_KEY_PAD_4 0x5c

- #define KBD_KEY_PAD_5 0x5d
- #define KBD_KEY_PAD_6 0x5e
- #define KBD_KEY_PAD_7 0x5f
- #define KBD_KEY_PAD_8 0x60
- #define KBD_KEY_PAD_9 0x61
- #define KBD_KEY_PAD_0 0x62
- #define KBD_KEY_PAD_PERIOD 0x63
- #define KBD_KEY_S3 0x65

7.53.1 Detailed Description

This is the list of keys that are on the keyboard that may be pressed. The keyboard returns keys in this format.

These are the raw keycodes returned by the US keyboard, and thus only cover the keys on US keyboards.

7.53.2 Macro Definition Documentation

KBD_KEY_0

```
#define KBD_KEY_0 0x27
```

KBD_KEY_1

```
#define KBD_KEY_1 0x1e
```

KBD_KEY_2

```
#define KBD_KEY_2 0x1f
```

KBD_KEY_3

```
#define KBD_KEY_3 0x20
```

KBD_KEY_4

```
#define KBD_KEY_4 0x21
```

KBD_KEY_5

```
#define KBD_KEY_5 0x22
```

KBD_KEY_6

```
#define KBD_KEY_6 0x23
```

KBD_KEY_7

```
#define KBD_KEY_7 0x24
```

KBD_KEY_8

```
#define KBD_KEY_8 0x25
```

KBD_KEY_9

```
#define KBD_KEY_9 0x26
```

KBD_KEY_A

```
#define KBD_KEY_A 0x04
```

KBD_KEY_B

```
#define KBD_KEY_B 0x05
```

KBD_KEY_BACKSLASH

```
#define KBD_KEY_BACKSLASH 0x31
```

KBD_KEY_BACKSPACE

```
#define KBD_KEY_BACKSPACE 0x2a
```

KBD_KEY_C

```
#define KBD_KEY_C 0x06
```

KBD_KEY_CAPSLOCK

```
#define KBD_KEY_CAPSLOCK 0x39
```

KBD_KEY_COMMA

```
#define KBD_KEY_COMMA 0x36
```

KBD_KEY_D

```
#define KBD_KEY_D 0x07
```

KBD_KEY_DEL

```
#define KBD_KEY_DEL 0x4c
```

KBD_KEY_DOWN

```
#define KBD_KEY_DOWN 0x51
```

KBD_KEY_E

```
#define KBD_KEY_E 0x08
```

KBD_KEY_END

```
#define KBD_KEY_END 0x4d
```

KBD_KEY_ENTER

```
#define KBD_KEY_ENTER 0x28
```

KBD_KEY_ERR2

```
#define KBD_KEY_ERR2 0x02
```

KBD_KEY_ERR3

```
#define KBD_KEY_ERR3 0x03
```

KBD_KEY_ERROR

```
#define KBD_KEY_ERROR 0x01
```

KBD_KEY_ESCAPE

```
#define KBD_KEY_ESCAPE 0x29
```

KBD_KEY_F

```
#define KBD_KEY_F 0x09
```

KBD_KEY_F1

```
#define KBD_KEY_F1 0x3a
```

KBD_KEY_F10

```
#define KBD_KEY_F10 0x43
```

KBD_KEY_F11

```
#define KBD_KEY_F11 0x44
```

KBD_KEY_F12

```
#define KBD_KEY_F12 0x45
```

KBD_KEY_F2

```
#define KBD_KEY_F2 0x3b
```

KBD_KEY_F3

```
#define KBD_KEY_F3 0x3c
```

KBD_KEY_F4

```
#define KBD_KEY_F4 0x3d
```

KBD_KEY_F5

```
#define KBD_KEY_F5 0x3e
```

KBD_KEY_F6

```
#define KBD_KEY_F6 0x3f
```

KBD_KEY_F7

```
#define KBD_KEY_F7 0x40
```

KBD_KEY_F8

```
#define KBD_KEY_F8 0x41
```

KBD_KEY_F9

```
#define KBD_KEY_F9 0x42
```

KBD_KEY_G

```
#define KBD_KEY_G 0x0a
```

KBD_KEY_H

```
#define KBD_KEY_H 0x0b
```

KBD_KEY_HOME

```
#define KBD_KEY_HOME 0x4a
```

KBD_KEY_I

```
#define KBD_KEY_I 0x0c
```

KBD_KEY_INSERT

```
#define KBD_KEY_INSERT 0x49
```

KBD_KEY_J

```
#define KBD_KEY_J 0x0d
```

KBD_KEY_K

```
#define KBD_KEY_K 0x0e
```

KBD_KEY_L

```
#define KBD_KEY_L 0x0f
```

KBD_KEY_LBRACKET

```
#define KBD_KEY_LBRACKET 0x2f
```

KBD_KEY_LEFT

```
#define KBD_KEY_LEFT 0x50
```

KBD_KEY_M

```
#define KBD_KEY_M 0x10
```

KBD_KEY_MINUS

```
#define KBD_KEY_MINUS 0x2d
```

KBD_KEY_N

```
#define KBD_KEY_N 0x11
```

KBD_KEY_NONE

```
#define KBD_KEY_NONE 0x00
```

KBD_KEY_O

```
#define KBD_KEY_O 0x12
```

KBD_KEY_P

```
#define KBD_KEY_P 0x13
```

KBD_KEY_PAD_0

```
#define KBD_KEY_PAD_0 0x62
```

KBD_KEY_PAD_1

```
#define KBD_KEY_PAD_1 0x59
```

KBD_KEY_PAD_2

```
#define KBD_KEY_PAD_2 0x5a
```

KBD_KEY_PAD_3

```
#define KBD_KEY_PAD_3 0x5b
```

KBD_KEY_PAD_4

```
#define KBD_KEY_PAD_4 0x5c
```

KBD_KEY_PAD_5

```
#define KBD_KEY_PAD_5 0x5d
```

KBD_KEY_PAD_6

```
#define KBD_KEY_PAD_6 0x5e
```

KBD_KEY_PAD_7

```
#define KBD_KEY_PAD_7 0x5f
```

KBD_KEY_PAD_8

```
#define KBD_KEY_PAD_8 0x60
```

KBD_KEY_PAD_9

```
#define KBD_KEY_PAD_9 0x61
```

KBD_KEY_PAD_DIVIDE

```
#define KBD_KEY_PAD_DIVIDE 0x54
```

KBD_KEY_PAD_ENTER

```
#define KBD_KEY_PAD_ENTER 0x58
```

KBD_KEY_PAD_MINUS

```
#define KBD_KEY_PAD_MINUS 0x56
```

KBD_KEY_PAD_MULTIPLY

```
#define KBD_KEY_PAD_MULTIPLY 0x55
```

KBD_KEY_PAD_NUMLOCK

```
#define KBD_KEY_PAD_NUMLOCK 0x53
```

KBD_KEY_PAD_PERIOD

```
#define KBD_KEY_PAD_PERIOD 0x63
```

KBD_KEY_PAD_PLUS

```
#define KBD_KEY_PAD_PLUS 0x57
```

KBD_KEY_PAUSE

```
#define KBD_KEY_PAUSE 0x48
```

KBD_KEY_PERIOD

```
#define KBD_KEY_PERIOD 0x37
```

KBD_KEY_PGDOWN

```
#define KBD_KEY_PGDOWN 0x4e
```

KBD_KEY_PGUP

```
#define KBD_KEY_PGUP 0x4b
```

KBD_KEY_PLUS

```
#define KBD_KEY_PLUS 0x2e
```

KBD_KEY_PRINT

```
#define KBD_KEY_PRINT 0x46
```

KBD_KEY_Q

```
#define KBD_KEY_Q 0x14
```

KBD_KEY_QUOTE

```
#define KBD_KEY_QUOTE 0x34
```

KBD_KEY_R

```
#define KBD_KEY_R 0x15
```

KBD_KEY_RBRACKET

```
#define KBD_KEY_RBRACKET 0x30
```

KBD_KEY_RIGHT

```
#define KBD_KEY_RIGHT 0x4f
```

KBD_KEY_S

```
#define KBD_KEY_S 0x16
```

KBD_KEY_S3

```
#define KBD_KEY_S3 0x65
```

KBD_KEY_SCRLOCK

```
#define KBD_KEY_SCRLOCK 0x47
```

KBD_KEY_SEMICOLON

```
#define KBD_KEY_SEMICOLON 0x33
```

KBD_KEY_SLASH

```
#define KBD_KEY_SLASH 0x38
```

KBD_KEY_SPACE

```
#define KBD_KEY_SPACE 0x2c
```

KBD_KEY_T

```
#define KBD_KEY_T 0x17
```

KBD_KEY_TAB

```
#define KBD_KEY_TAB 0x2b
```

KBD_KEY_TILDE

```
#define KBD_KEY_TILDE 0x35
```

KBD_KEY_U

```
#define KBD_KEY_U 0x18
```

KBD_KEY_UP

```
#define KBD_KEY_UP 0x52
```

KBD_KEY_V

```
#define KBD_KEY_V 0x19
```

KBD_KEY_W

```
#define KBD_KEY_W 0x1a
```

KBD_KEY_X

```
#define KBD_KEY_X 0x1b
```

KBD_KEY_Y

```
#define KBD_KEY_Y 0x1c
```

KBD_KEY_Z

```
#define KBD_KEY_Z 0x1d
```

7.54 Keyboard modifier keys

Macros

- #define KBD_MOD_LCTRL (1<<0)
- #define KBD_MOD_LSHIFT (1<<1)
- #define KBD_MOD_LALT (1<<2)
- #define KBD_MOD_S1 (1<<3)
- #define KBD_MOD_RCTRL (1<<4)
- #define KBD_MOD_RSHIFT (1<<5)
- #define KBD_MOD_RALT (1<<6)
- #define KBD_MOD_S2 (1<<7)

7.54.1 Detailed Description

These are the various modifiers that can be pressed on the keyboard, and are reflected in the modifiers field of kbd_cond_t.

7.54.2 Macro Definition Documentation

KBD_MOD_LALT

```
#define KBD_MOD_LALT (1<<2)
```

KBD_MOD_LCTRL

```
#define KBD_MOD_LCTRL (1<<0)
```

KBD_MOD_LSHIFT

```
#define KBD_MOD_LSHIFT (1<<1)
```

KBD_MOD_RALT

```
#define KBD_MOD_RALT (1<<6)
```

KBD_MOD_RCTRL

```
#define KBD_MOD_RCTRL (1<<4)
```

KBD_MOD_RSHIFT

```
#define KBD_MOD_RSHIFT (1<<5)
```

KBD_MOD_S1

```
#define KBD_MOD_S1 (1<<3)
```

KBD_MOD_S2

```
#define KBD_MOD_S2 (1<<7)
```

7.55 Keyboard region codes

Macros

- #define KBD_REGION_JP 1
  Japanese keyboard.
- #define KBD_REGION_US 2
  US keyboard.
- #define KBD_REGION_UK 3
  UK keyboard.
- #define KBD_REGION_DE 4
  German keyboard.
- #define KBD_REGION_FR 5
  French keyboard (not supported yet)
- #define KBD_REGION_IT 6
  Italian keyboard (not supported yet)
- #define KBD_REGION_ES 7
  Spanish keyboard.

7.55.1 Detailed Description

This is the list of possible values for the "region" field in the kbd_state_t structure.

7.55.2 Macro Definition Documentation

KBD_REGION_DE

```
#define KBD_REGION_DE 4
```

German keyboard.

KBD_REGION_ES

```
#define KBD_REGION_ES 7
```

Spanish keyboard.

KBD_REGION_FR

```
#define KBD_REGION_FR 5
```

French keyboard (not supported yet)

KBD_REGION_IT

```
#define KBD_REGION_IT 6
```

Italian keyboard (not supported yet)

KBD_REGION_JP

```
#define KBD_REGION_JP 1
```

Japanese keyboard.

KBD_REGION_UK

```
#define KBD_REGION_UK 3
```

UK keyboard.

KBD_REGION_US

```
#define KBD_REGION_US 2
```

US keyboard.

7.56 Language settings possible in the BIOS menu

Macros

- #define FLASHROM_LANG_JAPANESE 0
  Japanese language code.
- #define FLASHROM_LANG_ENGLISH 1
  English language code.
- #define FLASHROM_LANG_GERMAN 2
  German language code.
- #define FLASHROM_LANG_FRENCH 3
  French language code.
- #define FLASHROM_LANG_SPANISH 4
  Spanish language code.
- #define FLASHROM_LANG_ITALIAN 5
  Italian language code.

7.56.1 Detailed Description

This set of constants will be returned as the language value in the flashrom_syscfg_t structure.

7.56.2 Macro Definition Documentation

FLASHROM_LANG_ENGLISH

```
#define FLASHROM_LANG_ENGLISH 1
```

English language code.

FLASHROM_LANG_FRENCH

```
#define FLASHROM_LANG_FRENCH 3
```

French language code.

FLASHROM_LANG_GERMAN

```
#define FLASHROM_LANG_GERMAN 2
```

German language code.

FLASHROM_LANG_ITALIAN

```
#define FLASHROM_LANG_ITALIAN 5
```

Italian language code.

FLASHROM_LANG_JAPANESE

```
#define FLASHROM_LANG_JAPANESE 0
```

Japanese language code.

FLASHROM_LANG_SPANISH

```
#define FLASHROM_LANG_SPANISH 4
```

Spanish language code.

7.57 Log levels for dbglog

Macros

- #define DBG_DEAD 0
  The system is dead.
- #define DBG_CRITICAL 1
  A critical error message.
- #define DBG_ERROR 2
  A normal error message.
- #define DBG_WARNING 3
  Potential problem.
- #define DBG_NOTICE 4
  Normal but significant.
- #define DBG_INFO 5
  Informational messages.
- #define DBG_DEBUG 6
  User debug messages.
- #define DBG_KDEBUG 7
  Kernel debug messages.

7.57.1 Detailed Description

This is the list of levels that are allowed to be passed into the dbglog() function, representing different levels of importance.

7.57.2 Macro Definition Documentation

DBG_CRITICAL

```
#define DBG_CRITICAL 1
```

A critical error message.

DBG_DEAD

```
#define DBG_DEAD 0
```

The system is dead.

DBG_DEBUG

```
#define DBG_DEBUG 6
```

User debug messages.

DBG_ERROR

```
#define DBG_ERROR 2
```

A normal error message.

DBG_INFO

```
#define DBG_INFO 5
```

Informational messages.

DBG_KDEBUG

```
#define DBG_KDEBUG 7
```

Kernel debug messages.

DBG_NOTICE

```
#define DBG_NOTICE 4
```

Normal but significant.

DBG_WARNING

```
#define DBG_WARNING 3
```

Potential problem.

7.58 Logical blocks available in the flashrom

Macros

- #define FLASHROM_B1_SYSCFG 0x05
  System config (BLOCK_1)
- #define FLASHROM_B1_PW_SETTINGS_1 0x80
  PlanetWeb settings (BLOCK_1)
- #define FLASHROM_B1_PW_SETTINGS_2 0x81
  PlanetWeb settings (BLOCK_1)
- #define FLASHROM_B1_PW_SETTINGS_3 0x82
  PlanetWeb settings (BLOCK_1)
- #define FLASHROM_B1_PW_SETTINGS_4 0x83
  PlanetWeb settings (BLOCK_1)
- #define FLASHROM_B1_PW_SETTINGS_5 0x84
  PlanetWeb settings (BLOCK_1)
- #define FLASHROM_B1_PW_PPP1 0xC0
  PlanetWeb PPP settings (BLOCK_1)
- #define FLASHROM_B1_PW_PPP2 0xC1
  PlanetWeb PPP settings (BLOCK_1)
- #define FLASHROM_B1_PW_DNS 0xC2
  PlanetWeb DNS settings (BLOCK_1)
- #define FLASHROM_B1_PW_EMAIL1 0xC3
  PlanetWeb Email settings (BLOCK_1)
- #define FLASHROM_B1_PW_EMAIL2 0xC4
  PlanetWeb Email settings (BLOCK_1)
- #define FLASHROM_B1_PW_EMAIL_PROXY 0xC5
  PlanetWeb Email/Proxy settings (BLOCK_1)
- #define FLASHROM_B1_DK_PPP1 0xC6
  DreamKey PPP settings (also seen in PW)
- #define FLASHROM_B1_DK_PPP2 0xC7
  DreamKey PPP settings (also seen in PW)
- #define FLASHROM_B1_DK_DNS 0xC8

DreamKey PPP settings (also seen in PW)

- #define FLASHROM_B1_IP_SETTINGS 0xE0

  IP settings for BBA (BLOCK_1)
- #define FLASHROM_B1_EMAIL 0xE2

  Email address (BLOCK_1)
- #define FLASHROM_B1_SMTP 0xE4

  SMTP server setting (BLOCK_1)
- #define FLASHROM_B1_POP3 0xE5

  POP3 server setting (BLOCK_1)
- #define FLASHROM_B1_POP3LOGIN 0xE6

  POP3 login setting (BLOCK_1)
- #define FLASHROM_B1_POP3PASSWD 0xE7

  POP3 password setting + proxy (BLOCK_1)
- #define FLASHROM_B1_PPPLOGIN 0xE8

  PPP username + proxy (BLOCK_1)
- #define FLASHROM_B1_PPPPASSWD 0xE9

  PPP passwd (BLOCK_1)
- #define FLASHROM_B1_PPPMODEM 0xEB

  PPP modem settings.

7.58.1 Detailed Description

7.58.2 Macro Definition Documentation

FLASHROM_B1_DK_DNS

```
#define FLASHROM_B1_DK_DNS 0xC8
```

DreamKey PPP settings (also seen in PW)

FLASHROM_B1_DK_PPP1

```
#define FLASHROM_B1_DK_PPP1 0xC6
```

DreamKey PPP settings (also seen in PW)

FLASHROM_B1_DK_PPP2

```
#define FLASHROM_B1_DK_PPP2 0xC7
```

DreamKey PPP settings (also seen in PW)

FLASHROM_B1_EMAIL

```
#define FLASHROM_B1_EMAIL 0xE2
```

Email address (BLOCK_1)

FLASHROM_B1_IP_SETTINGS

```
#define FLASHROM_B1_IP_SETTINGS 0xE0
```

IP settings for BBA (BLOCK_1)

FLASHROM_B1_POP3

```
#define FLASHROM_B1_POP3 0xE5
```

POP3 server setting (BLOCK_1)

FLASHROM_B1_POP3LOGIN

```
#define FLASHROM_B1_POP3LOGIN 0xE6
```

POP3 login setting (BLOCK_1)

FLASHROM_B1_POP3PASSWD

```
#define FLASHROM_B1_POP3PASSWD 0xE7
```

POP3 password setting + proxy (BLOCK_1)

FLASHROM_B1_PPPLOGIN

```
#define FLASHROM_B1_PPPLOGIN 0xE8
```

PPP username + proxy (BLOCK_1)

FLASHROM_B1_PPPMODEM

```
#define FLASHROM_B1_PPPMODEM 0xEB
```

PPP modem settings.

FLASHROM_B1_PPPPASSWD

```
#define FLASHROM_B1_PPPPASSWD 0xE9
```

PPP passwd (BLOCK_1)

FLASHROM_B1_PW_DNS

```
#define FLASHROM_B1_PW_DNS 0xC2
```

PlanetWeb DNS settings (BLOCK_1)

FLASHROM_B1_PW_EMAIL1

```
#define FLASHROM_B1_PW_EMAIL1 0xC3
```

PlanetWeb Email settings (BLOCK_1)

FLASHROM_B1_PW_EMAIL2

```
#define FLASHROM_B1_PW_EMAIL2 0xC4
```

PlanetWeb Email settings (BLOCK_1)

FLASHROM_B1_PW_EMAIL_PROXY

```
#define FLASHROM_B1_PW_EMAIL_PROXY 0xC5
```

PlanetWeb Email/Proxy settings (BLOCK_1)

FLASHROM_B1_PW_PPP1

```
#define FLASHROM_B1_PW_PPP1 0xC0
```

PlanetWeb PPP settings (BLOCK_1)

FLASHROM_B1_PW_PPP2

```
#define FLASHROM_B1_PW_PPP2 0xC1
```

PlanetWeb PPP settings (BLOCK_1)

FLASHROM_B1_PW_SETTINGS_1

```
#define FLASHROM_B1_PW_SETTINGS_1 0x80
```

PlanetWeb settings (BLOCK_1)

FLASHROM_B1_PW_SETTINGS_2

```
#define FLASHROM_B1_PW_SETTINGS_2 0x81
```

PlanetWeb settings (BLOCK_1)

FLASHROM_B1_PW_SETTINGS_3

```
#define FLASHROM_B1_PW_SETTINGS_3 0x82
```

PlanetWeb settings (BLOCK_1)

FLASHROM_B1_PW_SETTINGS_4

```
#define FLASHROM_B1_PW_SETTINGS_4 0x83
```

PlanetWeb settings (BLOCK_1)

FLASHROM_B1_PW_SETTINGS_5

```
#define FLASHROM_B1_PW_SETTINGS_5 0x84
```

PlanetWeb settings (BLOCK_1)

FLASHROM_B1_SMTP

```
#define FLASHROM_B1_SMTP 0xE4
```

SMTP server setting (BLOCK_1)

FLASHROM_B1_SYSCFG

```
#define FLASHROM_B1_SYSCFG 0x05
```

System config (BLOCK_1)

7.59 MMU address bit definitions

Macros

- #define MMU_TOP_SHIFT 21

  Top-level shift.
- #define MMU_TOP_BITS 10

  Top-level bits.
- #define MMU_TOP_MASK ((1 << MMU_TOP_BITS) - 1)

  Top-level mask.
- #define MMU_BOT_SHIFT 12

  Bottom shift.
- #define MMU_BOT_BITS 9

  Bottom bits.
- #define MMU_BOT_MASK ((1 << MMU_BOT_BITS) - 1)

  Bottom mask.
- #define MMU_IND_SHIFT 0

  Index shift.
- #define MMU_IND_BITS 12

  Index bits.
- #define MMU_IND_MASK ((1 << MMU_IND_BITS) - 1)

  Index mask.

7.59.1 Detailed Description

The MMU code uses these to determine the page of a request.

7.59.2 Macro Definition Documentation

MMU_BOT_BITS

```
#define MMU_BOT_BITS 9
```

Bottom bits.

MMU_BOT_MASK

```
#define MMU_BOT_MASK ((1 << MMU_BOT_BITS) - 1)
```

Bottom mask.

MMU_BOT_SHIFT

```
#define MMU_BOT_SHIFT 12
```

Bottom shift.

MMU_IND_BITS

```
#define MMU_IND_BITS 12
```

Index bits.

MMU_IND_MASK

```
#define MMU_IND_MASK ((1 << MMU_IND_BITS) - 1)
```

Index mask.

MMU_IND_SHIFT

```
#define MMU_IND_SHIFT 0
```

Index shift.

MMU_TOP_BITS

```
#define MMU_TOP_BITS 10
```

Top-level bits.

MMU_TOP_MASK

```
#define MMU_TOP_MASK ((1 << MMU_TOP_BITS) - 1)
```

Top-level mask.

MMU_TOP_SHIFT

```
#define MMU_TOP_SHIFT 21
```

Top-level shift.

7.60 MMU cacheability settings

Macros

- #define MMU_NO_CACHE 1

  Cache disabled.
- #define MMU_CACHE_BACK 2

  Write-back cacheing.
- #define MMU_CACHE_WT 3

  Write-through cacheing.
- #define MMU_CACHEABLE MMU_CACHE_BACK

  Default cacheing.

7.60.1 Detailed Description

Each page mapped via the MMU can have its cacheability set individually.

7.60.2 Macro Definition Documentation

MMU_CACHE_BACK

```
#define MMU_CACHE_BACK 2
```

Write-back cacheing.

MMU_CACHE_WT

```
#define MMU_CACHE_WT 3
```

Write-through cacheing.

MMU_CACHEABLE

```
#define MMU_CACHEABLE MMU_CACHE_BACK
```

Default cacheing.

MMU_NO_CACHE

```
#define MMU_NO_CACHE 1
```

Cache disabled.

7.61 MMU protection settings

Macros

- #define MMU_KERNEL_RDONLY 0

  No user access, kernel read-only.
- #define MMU_KERNEL_RDWR 1

  No user access, kernel full.
- #define MMU_ALL_RDONLY 2

  Read-only user and kernel.
- #define MMU_ALL_RDWR 3

  Full access, user and kernel.

7.61.1 Detailed Description

Each page mapped via the MMU can be protected in a couple of different ways, as specified here.

7.61.2 Macro Definition Documentation

MMU_ALL_RDONLY

```
#define MMU_ALL_RDONLY 2
```

Read-only user and kernel.

MMU_ALL_RDWR

```
#define MMU_ALL_RDWR 3
```

Full access, user and kernel.

MMU_KERNEL_RDONLY

```
#define MMU_KERNEL_RDONLY 0
```

No user access, kernel read-only.

MMU_KERNEL_RDWR

```
#define MMU_KERNEL_RDWR 1
```

No user access, kernel full.

7.62 Macros for accessing the format of an image

Macros

- #define KOS_IMG_FMT_I(x) ((x) & 0xffff)

  Read the platform-independent half of the format.
- #define KOS_IMG_FMT_D(x) (((x) >> 16) & 0xffff)

  Read the platform-specific half of the format.
- #define KOS_IMG_FMT(i, d) (((i) & 0xffff) | (((d) & 0xffff) << 16))

  Build a format value from a platform-independent half and a platform-specific half of the value.

7.62.1 Detailed Description

These macros provide easy access to the fmt field of a kos_img_t object.

7.62.2 Macro Definition Documentation

KOS_IMG_FMT

```
#define KOS_IMG_FMT(
              i,
              d ) ( ((i) & 0xffff) | (((d) & 0xffff) << 16) )
```

Build a format value from a platform-independent half and a platform-specific half of the value.

This macro combines the platform-independent and platform-specific portions of an image format into a value suitable for storing as the fmt field of a kos_img_t object.

Parameters

i	The platform-independent half of the format.
d	The platform-specific half of the format. This should not be pre-shifted.

Returns

A complete image format value, suitable for placing in the fmt variable of a kos_img_t.

KOS_IMG_FMT_D

```
#define KOS_IMG_FMT_D(
              x ) (((x) >> 16) & 0xffff)
```

Read the platform-specific half of the format.

This macro masks the format of a kos_img_t to give you just the upper half of the value, which contains the platform-specific half of the format.

Parameters

x	An image format (fmt field of a kos_img_t).

Returns

The platform-specific half of the format.

KOS_IMG_FMT_I

```
#define KOS_IMG_FMT_I(
              x ) ((x) & 0xffff)
```

Read the platform-independent half of the format.

This macro masks the format of a kos_img_t to give you just the lower half of the value, which contains the platform-independent half of the format.

Parameters

x	An image format (fmt field of a kos_img_t).

Returns

The platform-independent half of the format.

7.63 Maple Bus register locations

Macros

- #define MAPLE_BASE 0xa05f6c00

  Maple register base.
- #define MAPLE_DMAADDR (MAPLE_BASE+0x04)

  DMA address register.
- #define MAPLE_RESET2 (MAPLE_BASE+0x10)

  Reset register #2.
- #define MAPLE_ENABLE (MAPLE_BASE+0x14)

  Enable register.
- #define MAPLE_STATE (MAPLE_BASE+0x18)

  Status register.
- #define MAPLE_SPEED (MAPLE_BASE+0x80)

  Speed register.
- #define MAPLE_RESET1 (MAPLE_BASE+0x8c)

  Reset register #1.

7.63.1 Detailed Description

These are various registers related to the Maple Bus. In general, you probably won't ever need to mess with these directly.

7.63.2 Macro Definition Documentation

MAPLE_BASE

```
#define MAPLE_BASE 0xa05f6c00
```

Maple register base.

MAPLE_DMAADDR

```
#define MAPLE_DMAADDR (MAPLE_BASE+0x04)
```

DMA address register.

MAPLE_ENABLE

```
#define MAPLE_ENABLE (MAPLE_BASE+0x14)
```

Enable register.

MAPLE_RESET1

```
#define MAPLE_RESET1 (MAPLE_BASE+0x8c)
```

Reset register #1.

MAPLE_RESET2

```
#define MAPLE_RESET2 (MAPLE_BASE+0x10)
```

Reset register #2.

MAPLE_SPEED

```
#define MAPLE_SPEED (MAPLE_BASE+0x80)
```

Speed register.

MAPLE_STATE

```
#define MAPLE_STATE (MAPLE_BASE+0x18)
```

Status register.

7.64 Maple commands and responses

Macros

- #define MAPLE_RESPONSE_FILEERR -5

  File error.
- #define MAPLE_RESPONSE_AGAIN -4

  Try again later.
- #define MAPLE_RESPONSE_BADCMD -3

  Bad command sent.
- #define MAPLE_RESPONSE_BADFUNC -2

  Bad function code.
- #define MAPLE_RESPONSE_NONE -1

  No response.
- #define MAPLE_COMMAND_DEVINFO 1

  Device info request.
- #define MAPLE_COMMAND_ALLINFO 2

  All info request.
- #define MAPLE_COMMAND_RESET 3

  Reset device request.
- #define MAPLE_COMMAND_KILL 4

  Kill device request.
- #define MAPLE_RESPONSE_DEVINFO 5

  Device info response.
- #define MAPLE_RESPONSE_ALLINFO 6

  All info response.
- #define MAPLE_RESPONSE_OK 7

  Command completed ok.
- #define MAPLE_RESPONSE_DATATRF 8

  Data transfer.
- #define MAPLE_COMMAND_GETCOND 9

  Get condition request.
- #define MAPLE_COMMAND_GETMINFO 10

  Get memory information.
- #define MAPLE_COMMAND_BREAD 11

  Block read.
- #define MAPLE_COMMAND_BWRITE 12

  Block write.
- #define MAPLE_COMMAND_BSYNC 13

  Block sync.

- #define MAPLE_COMMAND_SETCOND 14

  Set condition request.
- #define MAPLE_COMMAND_MICCONTROL 15

  Microphone control.
- #define MAPLE_COMMAND_CAMCONTROL 17

  Camera control.

7.64.1 Detailed Description

These are all either commands or responses to commands sent to or from Maple in normal operation.

7.64.2 Macro Definition Documentation

MAPLE_COMMAND_ALLINFO

```
#define MAPLE_COMMAND_ALLINFO 2
```

All info request.

MAPLE_COMMAND_BREAD

```
#define MAPLE_COMMAND_BREAD 11
```

Block read.

MAPLE_COMMAND_BSYNC

```
#define MAPLE_COMMAND_BSYNC 13
```

Block sync.

MAPLE_COMMAND_BWRITE

```
#define MAPLE_COMMAND_BWRITE 12
```

Block write.

MAPLE_COMMAND_CAMCONTROL

```
#define MAPLE_COMMAND_CAMCONTROL 17
```

Camera control.

MAPLE_COMMAND_DEVINFO

```
#define MAPLE_COMMAND_DEVINFO 1
```

Device info request.

MAPLE_COMMAND_GETCOND

```
#define MAPLE_COMMAND_GETCOND 9
```

Get condition request.

MAPLE_COMMAND_GETMINFO

```
#define MAPLE_COMMAND_GETMINFO 10
```

Get memory information.

MAPLE_COMMAND_KILL

```
#define MAPLE_COMMAND_KILL 4
```

Kill device request.

MAPLE_COMMAND_MICCONTROL

```
#define MAPLE_COMMAND_MICCONTROL 15
```

Microphone control.

MAPLE_COMMAND_RESET

```
#define MAPLE_COMMAND_RESET 3
```

Reset device request.

MAPLE_COMMAND_SETCOND

```
#define MAPLE_COMMAND_SETCOND 14
```

Set condition request.

MAPLE_RESPONSE_AGAIN

```
#define MAPLE_RESPONSE_AGAIN -4
```

Try again later.

MAPLE_RESPONSE_ALLINFO

```
#define MAPLE_RESPONSE_ALLINFO 6
```

All info response.

MAPLE_RESPONSE_BADCMD

```
#define MAPLE_RESPONSE_BADCMD -3
```

Bad command sent.

MAPLE_RESPONSE_BADFUNC

```
#define MAPLE_RESPONSE_BADFUNC -2
```

Bad function code.

MAPLE_RESPONSE_DATATRF

```
#define MAPLE_RESPONSE_DATATRF 8
```

Data transfer.

MAPLE_RESPONSE_DEVINFO

```
#define MAPLE_RESPONSE_DEVINFO 5
```

Device info response.

MAPLE_RESPONSE_FILEERR

```
#define MAPLE_RESPONSE_FILEERR -5
```

File error.

MAPLE_RESPONSE_NONE

```
#define MAPLE_RESPONSE_NONE -1
```

No response.

MAPLE_RESPONSE_OK

```
#define MAPLE_RESPONSE_OK 7
```

Command completed ok.

7.65 Maple device function codes

Macros

- #define MAPLE_FUNC_PURUPURU 0x00010000
  Jump pack.
- #define MAPLE_FUNC_MOUSE 0x00020000
  Mouse.
- #define MAPLE_FUNC_CAMERA 0x00080000
  Camera (Dreameye)
- #define MAPLE_FUNC_CONTROLLER 0x01000000
  Controller.
- #define MAPLE_FUNC_MEMCARD 0x02000000
  Memory card.
- #define MAPLE_FUNC_LCD 0x04000000
  LCD screen.
- #define MAPLE_FUNC_CLOCK 0x08000000
  Clock.
- #define MAPLE_FUNC_MICROPHONE 0x10000000
  Microphone.
- #define MAPLE_FUNC_ARGUN 0x20000000
  AR gun?
- #define MAPLE_FUNC_KEYBOARD 0x40000000
  Keyboard.
- #define MAPLE_FUNC_LIGHTGUN 0x80000000
  Lightgun.

7.65.1 Detailed Description

This is the list of maple device types (function codes). Each device must have at least one function to actually do anything.

7.65.2 Macro Definition Documentation

MAPLE_FUNC_ARGUN

```
#define MAPLE_FUNC_ARGUN 0x20000000
```

AR gun?

MAPLE_FUNC_CAMERA

```
#define MAPLE_FUNC_CAMERA 0x00080000
```

Camera (Dreameye)

MAPLE_FUNC_CLOCK

```
#define MAPLE_FUNC_CLOCK 0x08000000
```

Clock.

MAPLE_FUNC_CONTROLLER

```
#define MAPLE_FUNC_CONTROLLER 0x01000000
```

Controller.

MAPLE_FUNC_KEYBOARD

```
#define MAPLE_FUNC_KEYBOARD 0x40000000
```

Keyboard.

MAPLE_FUNC_LCD

```
#define MAPLE_FUNC_LCD 0x04000000
```

LCD screen.

MAPLE_FUNC_LIGHTGUN

```
#define MAPLE_FUNC_LIGHTGUN 0x80000000
```

Lightgun.

MAPLE_FUNC_MEMCARD

```
#define MAPLE_FUNC_MEMCARD 0x02000000
```

Memory card.

MAPLE_FUNC_MICROPHONE

```
#define MAPLE_FUNC_MICROPHONE 0x10000000
```

Microphone.

MAPLE_FUNC_MOUSE

```
#define MAPLE_FUNC_MOUSE 0x00020000
```

Mouse.

MAPLE_FUNC_PURUPURU

```
#define MAPLE_FUNC_PURUPURU 0x00010000
```

Jump pack.

7.66 Memory

Basics of the SH4 Memory Map.

Modules

- P4 memory region

  P4 SH-internal memory region (non-cachable).

Macros

- #define MEM_AREA_CACHE_MASK 0x1fffffff

  Mask a cache-agnostic address.
- #define MEM_AREA_U0_BASE 0x00000000

  U0 memory region (cachable).
- #define MEM_AREA_P0_BASE 0x00000000

  P0 memory region (cachable).
- #define MEM_AREA_P1_BASE 0x80000000

  P1 memory region (cachable).
- #define MEM_AREA_P2_BASE 0xa0000000

  P2 memory region (non-cachable).
- #define MEM_AREA_P3_BASE 0xc0000000

  P3 memory region (cachable).

7.66.1 Detailed Description

Basics of the SH4 Memory Map.

The SH7750 Series physical address space is mapped onto a 29-bit external memory space, with the upper 3 bits of the address indicating which memory region will be used. The P0/U0 memory region spans a 2GB space with the bottom 512MB mirrored to the P1, P2, and P3 regions.

7.66.2 Macro Definition Documentation

MEM_AREA_CACHE_MASK

```
#define MEM_AREA_CACHE_MASK 0x1fffffff
```

Mask a cache-agnostic address.

This masks out the upper 3 bits of an address. This is used when it is necssary to access memory with a specified caching mode. This is needed for DMA and SQ usage as well as various MMU functions.

MEM_AREA_P0_BASE

```
#define MEM_AREA_P0_BASE 0x00000000
```

P0 memory region (cachable).

This is the base privileged mode memory address. It is cacheable as determined by the WT bit of the cache control register. By default KOS sets this to copy-back mode.

MEM_AREA_P1_BASE

```
#define MEM_AREA_P1_BASE 0x80000000
```

P1 memory region (cachable).

This is a modularly cachable memory region. It is cacheable as determined by the CB bit of the cache control register. That allows it to function in a different caching mode (copy-back v write-through) than the U0, P0, and P3 regions, whose cache mode are governed by the WT bit. By default KOS sets this to the same copy-back mode as the other cachable regions.

MEM_AREA_P2_BASE

```
#define MEM_AREA_P2_BASE 0xa0000000
```

P2 memory region (non-cachable).

This is the non-cachable memory region. It is most frequently for DMA transactions to ensure reads are not cached.

MEM_AREA_P3_BASE

```
#define MEM_AREA_P3_BASE 0xc0000000
```

P3 memory region (cachable).

This functions as the lower 512MB of P0.

MEM_AREA_U0_BASE

```
#define MEM_AREA_U0_BASE 0x00000000
```

U0 memory region (cachable).

This is the base user mode memory address. It is cacheable as determined by the WT bit of the cache control register. By default KOS sets this to copy-back mode.

KOS runs in privileged mode, so this is here merely for completeness.

7.66.3 P4 memory region

P4 SH-internal memory region (non-cachable).

Macros

- #define MEM_AREA_SQ_BASE 0xe0000000
  Store Queue (SQ) memory base.
- #define MEM_AREA_ICACHE_ADDRESS_ARRAY_BASE 0xf0000000
  Instruction cache address array base.
- #define MEM_AREA_ICACHE_DATA_ARRAY_BASE 0xf1000000
  Instruction cache data array base.
- #define MEM_AREA_ITLB_ADDRESS_ARRAY_BASE 0xf2000000
  Instruction TLB address array base.
- #define MEM_AREA_ITLB_DATA_ARRAY1_BASE 0xf3000000
  Instruction TLB data array 1 base.
- #define MEM_AREA_ITLB_DATA_ARRAY2_BASE 0xf3800000
  Instruction TLB data array 2 base.
- #define MEM_AREA_OCACHE_ADDRESS_ARRAY_BASE 0xf4000000
  Operand cache address array base.
- #define MEM_AREA_OCACHE_DATA_ARRAY_BASE 0xf5000000
  Instruction cache data array base.
- #define MEM_AREA_UTLB_ADDRESS_ARRAY_BASE 0xf6000000
  Unified TLB address array base.
- #define MEM_AREA_UTLB_DATA_ARRAY1_BASE 0xf7000000
  Unified TLB data array 1 base.
- #define MEM_AREA_UTLB_DATA_ARRAY2_BASE 0xf7800000
  Unified TLB data array 2 base.
- #define MEM_AREA_CTRL_REG_BASE 0xff000000
  Control Register base.

7.66.3.1 Detailed Description

P4 SH-internal memory region (non-cachable).

This offset maps to on-chip I/O channels.

7.66.3.2 Macro Definition Documentation

MEM_AREA_CTRL_REG_BASE

```
#define MEM_AREA_CTRL_REG_BASE 0xff000000
```

Control Register base.

This is the base address of all control registers

MEM_AREA_ICACHE_ADDRESS_ARRAY_BASE

```
#define MEM_AREA_ICACHE_ADDRESS_ARRAY_BASE 0xf0000000
```

Instruction cache address array base.

This offset is used for direct access to the instruction cache address array.

MEM_AREA_ICACHE_DATA_ARRAY_BASE

```
#define MEM_AREA_ICACHE_DATA_ARRAY_BASE 0xf1000000
```

Instruction cache data array base.

This offset is used for direct access to the instruction cache data array.

MEM_AREA_ITLB_ADDRESS_ARRAY_BASE

```
#define MEM_AREA_ITLB_ADDRESS_ARRAY_BASE 0xf2000000
```

Instruction TLB address array base.

This offset is used for direct access to the instruction TLB address array.

MEM_AREA_ITLB_DATA_ARRAY1_BASE

```
#define MEM_AREA_ITLB_DATA_ARRAY1_BASE 0xf3000000
```

Instruction TLB data array 1 base.

This offset is used for direct access to the instruction TLB data array 1.

MEM_AREA_ITLB_DATA_ARRAY2_BASE

```
#define MEM_AREA_ITLB_DATA_ARRAY2_BASE 0xf3800000
```

Instruction TLB data array 2 base.

This offset is used for direct access to the instruction TLB data array 2.

MEM_AREA_OCACHE_ADDRESS_ARRAY_BASE

```
#define MEM_AREA_OCACHE_ADDRESS_ARRAY_BASE 0xf4000000
```

Operand cache address array base.

This offset is used for direct access to the operand cache address array.

MEM_AREA_OCACHE_DATA_ARRAY_BASE

```
#define MEM_AREA_OCACHE_DATA_ARRAY_BASE 0xf5000000
```

Instruction cache data array base.

This offset is used for direct access to the operand cache data array.

MEM_AREA_SQ_BASE

```
#define MEM_AREA_SQ_BASE 0xe0000000
```

Store Queue (SQ) memory base.

This offset maps to the SQ memory region. RW to addresses from 0xe0000000-0xe3ffffff follow SQ rules.

See also

dc\sq.h

MEM_AREA_UTLB_ADDRESS_ARRAY_BASE

```
#define MEM_AREA_UTLB_ADDRESS_ARRAY_BASE 0xf6000000
```

Unified TLB address array base.

This offset is used for direct access to the unified TLB address array.

MEM_AREA_UTLB_DATA_ARRAY1_BASE

```
#define MEM_AREA_UTLB_DATA_ARRAY1_BASE 0xf7000000
```

Unified TLB data array 1 base.

This offset is used for direct access to the unified TLB data array 1.

MEM_AREA_UTLB_DATA_ARRAY2_BASE

```
#define MEM_AREA_UTLB_DATA_ARRAY2_BASE 0xf7800000
```

Unified TLB data array 2 base.

This offset is used for direct access to the unified TLB data array 2.

7.67 Modem V.22 modes

Macros

- #define MODEM_SPEED_V22_1200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V22, MODEM_SPEED_1200)
  1200bps, V.22

7.67.1 Detailed Description

7.67.2 Macro Definition Documentation

MODEM_SPEED_V22_1200

```
#define MODEM_SPEED_V22_1200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V22, MODEM_SPEED_1200)
```

1200bps, V.22

7.68 Modem V.22bis modes

Macros

- #define MODEM_SPEED_V22BIS_1200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V22BIS, MODEM_SPEED_1200)
  1200bps, V.22bis
- #define MODEM_SPEED_V22BIS_2400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V22BIS, MODEM_SPEED_2400)
  2400bps, V.22bis

7.68.1 Detailed Description

7.68.2 Macro Definition Documentation

MODEM_SPEED_V22BIS_1200

```
#define MODEM_SPEED_V22BIS_1200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V22BIS, MODEM_SPEED_1200)
```

1200bps, V.22bis

MODEM_SPEED_V22BIS_2400

```
#define MODEM_SPEED_V22BIS_2400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V22BIS, MODEM_SPEED_2400)
```

2400bps, V.22bis

7.69 Modem V.32 bis modes

Macros

- #define MODEM_SPEED_V32BIS_7200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32BIS, MODEM_SPEED_7200)
  7200bps, V.32bis
- #define MODEM_SPEED_V32BIS_12000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32BIS, MODEM_SPEED_12000)
  12000bps, V.32bis
- #define MODEM_SPEED_V32BIS_14400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32BIS, MODEM_SPEED_14400)
  14400bps, V.32bis

7.69.1 Detailed Description

7.69.2 Macro Definition Documentation

MODEM_SPEED_V32BIS_12000

```
#define MODEM_SPEED_V32BIS_12000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32BIS, MODEM_SPEED_12000)
```

12000bps, V.32bis

MODEM_SPEED_V32BIS_14400

```
#define MODEM_SPEED_V32BIS_14400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32BIS, MODEM_SPEED_14400)
```

14400bps, V.32bis

MODEM_SPEED_V32BIS_7200

```
#define MODEM_SPEED_V32BIS_7200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32BIS, MODEM_SPEED_7200)
```

7200bps, V.32bis

7.70 Modem V.32 modes

Macros

- #define MODEM_SPEED_V32_4800 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32, MODEM_SPEED_4800)
  4800bps, V.32
- #define MODEM_SPEED_V32_9600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32, MODEM_SPEED_9600)
  9600bps, V.32

7.70.1 Detailed Description

7.70.2 Macro Definition Documentation

MODEM_SPEED_V32_4800

```
#define MODEM_SPEED_V32_4800 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32, MODEM_SPEED_4800)
```

4800bps, V.32

MODEM_SPEED_V32_9600

```
#define MODEM_SPEED_V32_9600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32, MODEM_SPEED_9600)
```

9600bps, V.32

7.71 Modem V.8 modes

Macros

- #define MODEM_SPEED_V8_2400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_2400)
  2400bps, V.8
- #define MODEM_SPEED_V8_4800 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_4800)
  4800bps, V.8
- #define MODEM_SPEED_V8_7200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_7200)
  7200bps, V.8
- #define MODEM_SPEED_V8_9600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_9600)
  9600bps, V.8
- #define MODEM_SPEED_V8_12000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_12000)
  12000bps, V.8
- #define MODEM_SPEED_V8_14400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_14400)
  14400bps, V.8
- #define MODEM_SPEED_V8_16800 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_16800)
  16800bps, V.8
- #define MODEM_SPEED_V8_19200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_19200)
  19200bps, V.8
- #define MODEM_SPEED_V8_21600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_21600)
  21600bps, V.8
- #define MODEM_SPEED_V8_24000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_24000)
  24000bps, V.8
- #define MODEM_SPEED_V8_26400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_26400)
  26400bps, V.8
- #define MODEM_SPEED_V8_28000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_28000)
  28000bps, V.8
- #define MODEM_SPEED_V8_31200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_31200)
  31200bps, V.8
- #define MODEM_SPEED_V8_33600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_33600)
  33600bps, V.8
- #define MODEM_SPEED_V8_AUTO MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_1200)
  Automatically set speed, V.8.

7.71.1 Detailed Description

7.71.2 Macro Definition Documentation

MODEM_SPEED_V8_12000

```
#define MODEM_SPEED_V8_12000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_12000)
```

12000bps, V.8

MODEM_SPEED_V8_14400

```
#define MODEM_SPEED_V8_14400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_14400)
```

14400bps, V.8

MODEM_SPEED_V8_16800

```
#define MODEM_SPEED_V8_16800 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_16800)
```

16800bps, V.8

MODEM_SPEED_V8_19200

```
#define MODEM_SPEED_V8_19200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_19200)
```

19200bps, V.8

MODEM_SPEED_V8_21600

```
#define MODEM_SPEED_V8_21600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_21600)
```

21600bps, V.8

MODEM_SPEED_V8_2400

```
#define MODEM_SPEED_V8_2400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_2400)
```

2400bps, V.8

MODEM_SPEED_V8_24000

```
#define MODEM_SPEED_V8_24000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_24000)
```

24000bps, V.8

MODEM_SPEED_V8_26400

```
#define MODEM_SPEED_V8_26400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_26400)
```

26400bps, V.8

MODEM_SPEED_V8_28000

```
#define MODEM_SPEED_V8_28000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_28000)
```

28000bps, V.8

MODEM_SPEED_V8_31200

```
#define MODEM_SPEED_V8_31200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_31200)
```

31200bps, V.8

MODEM_SPEED_V8_33600

```
#define MODEM_SPEED_V8_33600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_33600)
```

33600bps, V.8

MODEM_SPEED_V8_4800

```
#define MODEM_SPEED_V8_4800 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_4800)
```

4800bps, V.8

MODEM_SPEED_V8_7200

```
#define MODEM_SPEED_V8_7200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_7200)
```

7200bps, V.8

MODEM_SPEED_V8_9600

```
#define MODEM_SPEED_V8_9600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_9600)
```

9600bps, V.8

MODEM_SPEED_V8_AUTO

```
#define MODEM_SPEED_V8_AUTO MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_1200)
```

Automatically set speed, V.8.

7.72 Modem protocol values

Macros

- #define MODEM_PROTOCOL_V17 0x0
- #define MODEM_PROTOCOL_V22 0x1
- #define MODEM_PROTOCOL_V22BIS 0x2
- #define MODEM_PROTOCOL_V32 0x3
- #define MODEM_PROTOCOL_V32BIS 0x4
- #define MODEM_PROTOCOL_V34 0x5
- #define MODEM_PROTOCOL_V8 0x6

7.72.1 Detailed Description

This group defines the available protocol values that are able to be used with the Dreamcast's modem. The actual speed value consists of one of these in the upper 4 bits and one of the speeds in the lower 4 bits. Don't try to use any protocols not defined here, as bad things may happen.

It should be fairly obvious from the names what the protocols that will be used are.

7.72.2 Macro Definition Documentation

MODEM_PROTOCOL_V17

```
#define MODEM_PROTOCOL_V17 0x0
```

MODEM_PROTOCOL_V22

```
#define MODEM_PROTOCOL_V22 0x1
```

MODEM_PROTOCOL_V22BIS

```
#define MODEM_PROTOCOL_V22BIS 0x2
```

MODEM_PROTOCOL_V32

```
#define MODEM_PROTOCOL_V32 0x3
```

MODEM_PROTOCOL_V32BIS

```
#define MODEM_PROTOCOL_V32BIS 0x4
```

MODEM_PROTOCOL_V34

```
#define MODEM_PROTOCOL_V34 0x5
```

MODEM_PROTOCOL_V8

```
#define MODEM_PROTOCOL_V8 0x6
```

7.73 Modem speed values

Macros

- #define MODEM_SPEED_AUTO 0x0
- #define MODEM_SPEED_1200 0x0
- #define MODEM_SPEED_2400 0x1
- #define MODEM_SPEED_4800 0x2
- #define MODEM_SPEED_7200 0x3
- #define MODEM_SPEED_9600 0x4
- #define MODEM_SPEED_12000 0x5
- #define MODEM_SPEED_14400 0x6
- #define MODEM_SPEED_16800 0x7
- #define MODEM_SPEED_19200 0x8
- #define MODEM_SPEED_21600 0x9
- #define MODEM_SPEED_24000 0xA
- #define MODEM_SPEED_26400 0xB
- #define MODEM_SPEED_28000 0xC
- #define MODEM_SPEED_31200 0xD
- #define MODEM_SPEED_33600 0xE

7.73.1 Detailed Description

This group defines the available speed values that are able to be used with the Dreamcast's modem. The actual speed value consists of one of these in the lower 4 bits and one of the protocols in the upper 4 bits. Don't try to use any speeds not defined here, as bad things may happen.

It should be fairly obvious from the names what the speeds are (they're all expressed in bits per second).

7.73.2 Macro Definition Documentation

MODEM_SPEED_1200

```
#define MODEM_SPEED_1200 0x0
```

MODEM_SPEED_12000

```
#define MODEM_SPEED_12000 0x5
```

MODEM_SPEED_14400

```
#define MODEM_SPEED_14400 0x6
```

MODEM_SPEED_16800

```
#define MODEM_SPEED_16800 0x7
```

MODEM_SPEED_19200

```
#define MODEM_SPEED_19200 0x8
```

MODEM_SPEED_21600

```
#define MODEM_SPEED_21600 0x9
```

MODEM_SPEED_2400

```
#define MODEM_SPEED_2400 0x1
```

MODEM_SPEED_24000

```
#define MODEM_SPEED_24000 0xA
```

MODEM_SPEED_26400

```
#define MODEM_SPEED_26400 0xB
```

MODEM_SPEED_28000

```
#define MODEM_SPEED_28000 0xC
```

MODEM_SPEED_31200

```
#define MODEM_SPEED_31200 0xD
```

MODEM_SPEED_33600

```
#define MODEM_SPEED_33600 0xE
```

MODEM_SPEED_4800

```
#define MODEM_SPEED_4800 0x2
```

MODEM_SPEED_7200

```
#define MODEM_SPEED_7200 0x3
```

MODEM_SPEED_9600

```
#define MODEM_SPEED_9600 0x4
```

MODEM_SPEED_AUTO

```
#define MODEM_SPEED_AUTO 0x0
```

7.74 Modes of operation of the Dreamcast modem.

Macros

- #define MODEM_MODE_REMOTE 0

  Connect to a remote modem.
- #define MODEM_MODE_ANSWER 1

  Answer a call when a ring is detected.
- #define MODEM_MODE_NULL 255

  Modem not in use. Do not attempt to set this mode yourself!

7.74.1 Detailed Description

This group defines the modes that the Dreamcast modem can be in at any given point in time.

7.74.2 Macro Definition Documentation

MODEM_MODE_ANSWER

```
#define MODEM_MODE_ANSWER 1
```

Answer a call when a ring is detected.

MODEM_MODE_NULL

```
#define MODEM_MODE_NULL 255
```

Modem not in use. Do not attempt to set this mode yourself!

MODEM_MODE_REMOTE

```
#define MODEM_MODE_REMOTE 0
```

Connect to a remote modem.

7.75 Modifier volume mode parameters

Macros

- #define PVR_MODIFIER_OTHER_POLY 0

  Not the last polygon in the volume.
- #define PVR_MODIFIER_INCLUDE_LAST_POLY 1

  Last polygon, inclusion volume.
- #define PVR_MODIFIER_EXCLUDE_LAST_POLY 2

  Last polygon, exclusion volume.

7.75.1 Detailed Description

All triangles in a single modifier volume should be of the other poly type, except for the last one. That should be either of the other two types, depending on whether you want an inclusion or exclusion volume.

7.75.2 Macro Definition Documentation

PVR_MODIFIER_EXCLUDE_LAST_POLY

```
#define PVR_MODIFIER_EXCLUDE_LAST_POLY 2
```

Last polygon, exclusion volume.

PVR_MODIFIER_INCLUDE_LAST_POLY

```
#define PVR_MODIFIER_INCLUDE_LAST_POLY 1
```

Last polygon, inclusion volume.

PVR_MODIFIER_OTHER_POLY

```
#define PVR_MODIFIER_OTHER_POLY 0
```

Not the last polygon in the volume.

7.76 Mount flags for fs_ext2

Macros

- #define FS_EXT2_MOUNT_READONLY 0x00000000

  Mount read-only.
- #define FS_EXT2_MOUNT_READWRITE 0x00000001

  Mount read-write.

7.76.1 Detailed Description

These values are the valid flags that can be passed for the flags parameter to the fs_ext2_mount() function. Note that these can be combined, except for the read-only flag.

Also, it is not possible to mount some filesystems as read-write. For instance, if the filesystem was marked as not cleanly unmounted (from Linux itself), the driver will fail to mount the device as read-write. Also, if the block device does not support writing, then the filesystem will not be mounted as read-write (for obvious reasons).

These should stay synchronized with the ones in ext2fs.h.

7.76.2 Macro Definition Documentation

FS_EXT2_MOUNT_READONLY

```
#define FS_EXT2_MOUNT_READONLY 0x00000000
```

Mount read-only.

FS_EXT2_MOUNT_READWRITE

```
#define FS_EXT2_MOUNT_READWRITE 0x00000001
```

Mount read-write.

7.77 Mount flags for fs_fat

Macros

- #define FS_FAT_MOUNT_READONLY 0x00000000

  Mount read-only.
- #define FS_FAT_MOUNT_READWRITE 0x00000001

  Mount read-write.

7.77.1 Detailed Description

These values are the valid flags that can be passed for the flags parameter to the fs_fat_mount() function. Note that these can be combined, except for the read-only flag.

Also, it is not possible to mount some filesystems as read-write. For instance, if the filesystem was marked as not cleanly unmounted the driver will fail to mount the device as read-write. Also, if the block device does not support writing, then the filesystem will not be mounted as read-write (for obvious reasons).

These should stay synchronized with the ones in fatfs.h.

7.77.2 Macro Definition Documentation

FS_FAT_MOUNT_READONLY

```
#define FS_FAT_MOUNT_READONLY 0x00000000
```

Mount read-only.

FS_FAT_MOUNT_READWRITE

```
#define FS_FAT_MOUNT_READWRITE 0x00000001
```

Mount read-write.

7.78 Mouse button codes

Macros

- #define MOUSE_RIGHTBUTTON (1<<1)

  Right mouse button.
- #define MOUSE_LEFTBUTTON (1<<2)

  Left mouse button.
- #define MOUSE_SIDEBUTTON (1<<3)

  Side mouse button.

7.78.1 Detailed Description

These are the possible buttons to press on a maple bus mouse.

7.78.2 Macro Definition Documentation

MOUSE_LEFTBUTTON

```
#define MOUSE_LEFTBUTTON (1<<2)
```

Left mouse button.

MOUSE_RIGHTBUTTON

```
#define MOUSE_RIGHTBUTTON (1<<1)
```

Right mouse button.

MOUSE_SIDEBUTTON

```
#define MOUSE_SIDEBUTTON (1<<3)
```

Side mouse button.

7.79 Name handler types

Macros

- #define NMMGR_TYPE_UNKNOWN 0x0000 /* ? */
  Unknown nmmgr type.
- #define NMMGR_TYPE_VFS 0x0010 /* Mounted file system */
  A mounted filesystem.
- #define NMMGR_TYPE_BLOCKDEV 0x0020 /* Block device */
  A block device.
- #define NMMGR_TYPE_SINGLETON 0x0030 /* Singleton service (e.g., /dev/irq) */
  A singleton service (e.g., /dev/irq)
- #define NMMGR_TYPE_SYMTAB 0x0040 /* Symbol table */
  A symbol table.
- #define NMMGR_SYS_MAX 0x10000 /* Here and above are user types */
  Everything this and above is a user type.

7.79.1 Detailed Description

This is the set of all defined types of name manager handlers. All system types are defined below NMMGR_SYS_MAX.

7.79.2 Macro Definition Documentation

NMMGR_SYS_MAX

```
#define NMMGR_SYS_MAX 0x10000 /* Here and above are user types */
```

Everything this and above is a user type.

NMMGR_TYPE_BLOCKDEV

```
#define NMMGR_TYPE_BLOCKDEV 0x0020 /* Block device */
```

A block device.

NMMGR_TYPE_SINGLETON

```
#define NMMGR_TYPE_SINGLETON 0x0030 /* Singleton service (e.g., /dev/irq) */
```

A singleton service (e.g., /dev/irq)

NMMGR_TYPE_SYMTAB

```
#define NMMGR_TYPE_SYMTAB 0x0040 /* Symbol table */
```

A symbol table.

NMMGR_TYPE_UNKNOWN

```
#define NMMGR_TYPE_UNKNOWN 0x0000 /* ?  */
```

Unknown nmmgr type.

NMMGR_TYPE_VFS

```
#define NMMGR_TYPE_VFS 0x0010 /* Mounted file system */
```

A mounted filesystem.

7.80 Network configuration sources

Macros

- #define NETCFG_SRC_VMU 0

  Read from a VMU.
- #define NETCFG_SRC_FLASH 1

  Read from the flashrom.
- #define NETCFG_SRC_CWD 2

  Read from the working directory.
- #define NETCFG_SRC_CDROOT 3

  Read from the root of the CD.

7.80.1 Detailed Description

These constants give the list of places that the network configuration might be read from. One of these constants should be in the src field of objects of type netcfg_t.

7.80.2 Macro Definition Documentation

NETCFG_SRC_CDROOT

```
#define NETCFG_SRC_CDROOT 3
```

Read from the root of the CD.

NETCFG_SRC_CWD

```
#define NETCFG_SRC_CWD 2
```

Read from the working directory.

NETCFG_SRC_FLASH

```
#define NETCFG_SRC_FLASH 1
```

Read from the flashrom.

NETCFG_SRC_VMU

```
#define NETCFG_SRC_VMU 0
```

Read from a VMU.

7.81 Network connection methods

Macros

- #define NETCFG_METHOD_DHCP 0

  Use DHCP to configure.
- #define NETCFG_METHOD_STATIC 1

  Static network configuration.
- #define NETCFG_METHOD_PPPOE 4

  Use PPPoE.

7.81.1 Detailed Description

These constants give the list of network connection methods that are supported by the netcfg_t type. One of these will be in the method field of objects of that type.

7.81.2 Macro Definition Documentation

NETCFG_METHOD_DHCP

```
#define NETCFG_METHOD_DHCP 0
```

Use DHCP to configure.

NETCFG_METHOD_PPPOE

```
#define NETCFG_METHOD_PPPOE 4
```

Use PPPoE.

NETCFG_METHOD_STATIC

```
#define NETCFG_METHOD_STATIC 1
```

Static network configuration.

7.82 Offsets to registers of the PVR

Macros

- #define PVR_ID 0x0000

  Chip ID.
- #define PVR_REVISION 0x0004

  Chip revision.
- #define PVR_RESET 0x0008

  Reset pins.
- #define PVR_ISP_START 0x0014

  Start the ISP/TSP.
- #define PVR_UNK_0018 0x0018

  ??
- #define PVR_ISP_VERTBUF_ADDR 0x0020

  Vertex buffer address for scene rendering.
- #define PVR_ISP_TILEMAT_ADDR 0x002c

  Tile matrix address for scene rendering.
- #define PVR_SPANSORT_CFG 0x0030

  ?? – write 0x101 for now
- #define PVR_BORDER_COLOR 0x0040

  Border Color in RGB888.
- #define PVR_FB_CFG_1 0x0044

  Framebuffer config 1.
- #define PVR_FB_CFG_2 0x0048

  Framebuffer config 2.
- #define PVR_RENDER_MODULO 0x004c

  Render modulo.
- #define PVR_FB_ADDR 0x0050

  Framebuffer start address.
- #define PVR_FB_IL_ADDR 0x0054

  Framebuffer odd-field start address for interlace.
- #define PVR_FB_SIZE 0x005c

  Framebuffer display size.
- #define PVR_RENDER_ADDR 0x0060

  Render output address.
- #define PVR_RENDER_ADDR_2 0x0064

  Output for strip-buffering.
- #define PVR_PCLIP_X 0x0068

  Horizontal clipping area.
- #define PVR_PCLIP_Y 0x006c

  Vertical clipping area.
- #define PVR_CHEAP_SHADOW 0x0074

  Cheap shadow control.
- #define PVR_OBJECT_CLIP 0x0078

  Distance for polygon culling.
- #define PVR_UNK_007C 0x007c

?? – write 0x0027df77 for now

- #define PVR_UNK_0080 0x0080

  ?? – write 7 for now
- #define PVR_TEXTURE_CLIP 0x0084

  Distance for texture clipping.
- #define PVR_BGPLANE_Z 0x0088

  Distance for background plane.
- #define PVR_BGPLANE_CFG 0x008c

  Background plane config.
- #define PVR_UNK_0098 0x0098

  ?? – write 0x00800408 for now
- #define PVR_UNK_00A0 0x00a0

  ?? – write 0x20 for now
- #define PVR_UNK_00A8 0x00a8

  ?? – write 0x15d1c951 for now
- #define PVR_FOG_TABLE_COLOR 0x00b0

  Table fog color.
- #define PVR_FOG_VERTEX_COLOR 0x00b4

  Vertex fog color.
- #define PVR_FOG_DENSITY 0x00b8

  Fog density coefficient.
- #define PVR_COLOR_CLAMP_MAX 0x00bc

  RGB Color clamp max.
- #define PVR_COLOR_CLAMP_MIN 0x00c0

  RGB Color clamp min.
- #define PVR_GUN_POS 0x00c4

  Light gun position.
- #define PVR_HPOS_IRQ 0x00c8

  Horizontal position IRQ.
- #define PVR_VPOS_IRQ 0x00cc

  Vertical position IRQ.
- #define PVR_IL_CFG 0x00d0

  Interlacing config.
- #define PVR_BORDER_X 0x00d4

  Window border X position.
- #define PVR_SCAN_CLK 0x00d8

  Clock and scanline values.
- #define PVR_BORDER_Y 0x00dc

  Window border Y position.
- #define PVR_TEXTURE_MODULO 0x00e4

  Output texture width modulo.
- #define PVR_VIDEO_CFG 0x00e8

  Misc video config.
- #define PVR_BITMAP_X 0x00ec

  Bitmap window X position.
- #define PVR_BITMAP_Y 0x00f0

  Bitmap window Y position.

- #define PVR_SCALER_CFG 0x00f4

  Smoothing scaler.
- #define PVR_PALETTE_CFG 0x0108

  Palette format.
- #define PVR_SYNC_STATUS 0x010c

  V/H blank status.
- #define PVR_UNK_0110 0x0110

  ?? – write 0x93f39 for now
- #define PVR_UNK_0114 0x0114

  ?? – write 0x200000 for now
- #define PVR_UNK_0118 0x0118

  ?? – write 0x8040 for now
- #define PVR_TA_OPB_START 0x0124

  Object Pointer Buffer start for TA usage.
- #define PVR_TA_VERTBUF_START 0x0128

  Vertex buffer start for TA usage.
- #define PVR_TA_OPB_END 0x012c

  OPB end for TA usage.
- #define PVR_TA_VERTBUF_END 0x0130

  Vertex buffer end for TA usage.
- #define PVR_TA_OPB_POS 0x0134

  Top used memory location in OPB for TA usage.
- #define PVR_TA_VERTBUF_POS 0x0138

  Top used memory location in vertbuf for TA usage.
- #define PVR_TILEMAT_CFG 0x013c

  Tile matrix size config.
- #define PVR_OPB_CFG 0x0140

  Active lists / list size.
- #define PVR_TA_INIT 0x0144

  Initialize vertex reg. params.
- #define PVR_YUV_ADDR 0x0148

  YUV conversion destination.
- #define PVR_YUV_CFG 0x014c

  YUV configuration.
- #define PVR_YUV_STAT 0x0150

  The number of YUV macroblocks converted.
- #define PVR_UNK_0160 0x0160

  ??
- #define PVR_TA_OPB_INIT 0x0164

  Object pointer buffer position init.
- #define PVR_FOG_TABLE_BASE 0x0200

  Base of the fog table.
- #define PVR_PALETTE_TABLE_BASE 0x1000

  Base of the palette table.

7.82.1 Detailed Description

7.82.2 Macro Definition Documentation

PVR_BGPLANE_CFG

```
#define PVR_BGPLANE_CFG 0x008c
```

Background plane config.

PVR_BGPLANE_Z

```
#define PVR_BGPLANE_Z 0x0088
```

Distance for background plane.

PVR_BITMAP_X

```
#define PVR_BITMAP_X 0x00ec
```

Bitmap window X position.

PVR_BITMAP_Y

```
#define PVR_BITMAP_Y 0x00f0
```

Bitmap window Y position.

PVR_BORDER_COLOR

```
#define PVR_BORDER_COLOR 0x0040
```

Border Color in RGB888.

PVR_BORDER_X

```
#define PVR_BORDER_X 0x00d4
```

Window border X position.

PVR_BORDER_Y

```
#define PVR_BORDER_Y 0x00dc
```

Window border Y position.

PVR_CHEAP_SHADOW

```
#define PVR_CHEAP_SHADOW 0x0074
```

Cheap shadow control.

PVR_COLOR_CLAMP_MAX

```
#define PVR_COLOR_CLAMP_MAX 0x00bc
```

RGB Color clamp max.

PVR_COLOR_CLAMP_MIN

```
#define PVR_COLOR_CLAMP_MIN 0x00c0
```

RGB Color clamp min.

PVR_FB_ADDR

```
#define PVR_FB_ADDR 0x0050
```

Framebuffer start address.

PVR_FB_CFG_1

```
#define PVR_FB_CFG_1 0x0044
```

Framebuffer config 1.

PVR_FB_CFG_2

```
#define PVR_FB_CFG_2 0x0048
```

Framebuffer config 2.

PVR_FB_IL_ADDR

```
#define PVR_FB_IL_ADDR 0x0054
```

Framebuffer odd-field start address for interlace.

PVR_FB_SIZE

```
#define PVR_FB_SIZE 0x005c
```

Framebuffer display size.

PVR_FOG_DENSITY

```
#define PVR_FOG_DENSITY 0x00b8
```

Fog density coefficient.

PVR_FOG_TABLE_BASE

```
#define PVR_FOG_TABLE_BASE 0x0200
```

Base of the fog table.

PVR_FOG_TABLE_COLOR

```
#define PVR_FOG_TABLE_COLOR 0x00b0
```

Table fog color.

PVR_FOG_VERTEX_COLOR

```
#define PVR_FOG_VERTEX_COLOR 0x00b4
```

Vertex fog color.

PVR_GUN_POS

```
#define PVR_GUN_POS 0x00c4
```

Light gun position.

PVR_HPOS_IRQ

```
#define PVR_HPOS_IRQ 0x00c8
```

Horizontal position IRQ.

PVR_ID

```
#define PVR_ID 0x0000
```

Chip ID.

PVR_IL_CFG

```
#define PVR_IL_CFG 0x00d0
```

Interlacing config.

PVR_ISP_START

```
#define PVR_ISP_START 0x0014
```

Start the ISP/TSP.

PVR_ISP_TILEMAT_ADDR

```
#define PVR_ISP_TILEMAT_ADDR 0x002c
```

Tile matrix address for scene rendering.

PVR_ISP_VERTBUF_ADDR

```
#define PVR_ISP_VERTBUF_ADDR 0x0020
```

Vertex buffer address for scene rendering.

PVR_OBJECT_CLIP

```
#define PVR_OBJECT_CLIP 0x0078
```

Distance for polygon culling.

PVR_OPB_CFG

```
#define PVR_OPB_CFG 0x0140
```

Active lists / list size.

PVR_PALETTE_CFG

```
#define PVR_PALETTE_CFG 0x0108
```

Palette format.

PVR_PALETTE_TABLE_BASE

```
#define PVR_PALETTE_TABLE_BASE 0x1000
```

Base of the palette table.

PVR_PCLIP_X

```
#define PVR_PCLIP_X 0x0068
```

Horizontal clipping area.

PVR_PCLIP_Y

```
#define PVR_PCLIP_Y 0x006c
```

Vertical clipping area.

PVR_RENDER_ADDR

```
#define PVR_RENDER_ADDR 0x0060
```

Render output address.

PVR_RENDER_ADDR_2

```
#define PVR_RENDER_ADDR_2 0x0064
```

Output for strip-buffering.

PVR_RENDER_MODULO

```
#define PVR_RENDER_MODULO 0x004c
```

Render modulo.

PVR_RESET

```
#define PVR_RESET 0x0008
```

Reset pins.

PVR_REVISION

```
#define PVR_REVISION 0x0004
```

Chip revision.

PVR_SCALER_CFG

```
#define PVR_SCALER_CFG 0x00f4
```

Smoothing scaler.

PVR_SCAN_CLK

```
#define PVR_SCAN_CLK 0x00d8
```

Clock and scanline values.

PVR_SPANSORT_CFG

```
#define PVR_SPANSORT_CFG 0x0030
```

?? – write 0x101 for now

PVR_SYNC_STATUS

```
#define PVR_SYNC_STATUS 0x010c
```

V/H blank status.

PVR_TA_INIT

```
#define PVR_TA_INIT 0x0144
```

Initialize vertex reg. params.

PVR_TA_OPB_END

```
#define PVR_TA_OPB_END 0x012c
```

OPB end for TA usage.

PVR_TA_OPB_INIT

```
#define PVR_TA_OPB_INIT 0x0164
```

Object pointer buffer position init.

PVR_TA_OPB_POS

```
#define PVR_TA_OPB_POS 0x0134
```

Top used memory location in OPB for TA usage.

PVR_TA_OPB_START

```
#define PVR_TA_OPB_START 0x0124
```

Object Pointer Buffer start for TA usage.

PVR_TA_VERTBUF_END

```
#define PVR_TA_VERTBUF_END 0x0130
```

Vertex buffer end for TA usage.

PVR_TA_VERTBUF_POS

```
#define PVR_TA_VERTBUF_POS 0x0138
```

Top used memory location in vertbuf for TA usage.

PVR_TA_VERTBUF_START

```
#define PVR_TA_VERTBUF_START 0x0128
```

Vertex buffer start for TA usage.

PVR_TEXTURE_CLIP

```
#define PVR_TEXTURE_CLIP 0x0084
```

Distance for texture clipping.

PVR_TEXTURE_MODULO

```
#define PVR_TEXTURE_MODULO 0x00e4
```

Output texture width modulo.

PVR_TILEMAT_CFG

```
#define PVR_TILEMAT_CFG 0x013c
```

Tile matrix size config.

PVR_UNK_0018

```
#define PVR_UNK_0018 0x0018
```

??

PVR_UNK_007C

```
#define PVR_UNK_007C 0x007c
```

?? – write 0x0027df77 for now

PVR_UNK_0080

```
#define PVR_UNK_0080 0x0080
```

?? – write 7 for now

PVR_UNK_0098

```
#define PVR_UNK_0098 0x0098
```

?? – write 0x00800408 for now

PVR_UNK_00A0

```
#define PVR_UNK_00A0 0x00a0
```

?? – write 0x20 for now

PVR_UNK_00A8

```
#define PVR_UNK_00A8 0x00a8
```

?? – write 0x15d1c951 for now

PVR_UNK_0110

```
#define PVR_UNK_0110 0x0110
```

?? – write 0x93f39 for now

PVR_UNK_0114

```
#define PVR_UNK_0114 0x0114
```

?? – write 0x200000 for now

PVR_UNK_0118

```
#define PVR_UNK_0118 0x0118
```

?? – write 0x8040 for now

PVR_UNK_0160

```
#define PVR_UNK_0160 0x0160
```

??

PVR_VIDEO_CFG

```
#define PVR_VIDEO_CFG 0x00e8
```

Misc video config.

PVR_VPOS_IRQ

```
#define PVR_VPOS_IRQ 0x00cc
```

Vertical position IRQ.

PVR_YUV_ADDR

```
#define PVR_YUV_ADDR 0x0148
```

YUV conversion destination.

PVR_YUV_CFG

```
#define PVR_YUV_CFG 0x014c
```

YUV configuration.

PVR_YUV_STAT

```
#define PVR_YUV_STAT 0x0150
```

The number of YUV macroblocks converted.

7.83 PPP automaton phases

Macros

- #define PPP_PHASE_DEAD 0x01
  Pre-connection.
- #define PPP_PHASE_ESTABLISH 0x02
  Establishing connection.
- #define PPP_PHASE_AUTHENTICATE 0x03
  Authentication to peer.
- #define PPP_PHASE_NETWORK 0x04
  Established and working.
- #define PPP_PHASE_TERMINATE 0x05
  Tearing down the link.

7.83.1 Detailed Description

This list defines the phases of the PPP automaton, as described in Section 3.2 of RFC 1661.

7.83.2 Macro Definition Documentation

PPP_PHASE_AUTHENTICATE

```
#define PPP_PHASE_AUTHENTICATE 0x03
```

Authentication to peer.

PPP_PHASE_DEAD

```
#define PPP_PHASE_DEAD 0x01
```

Pre-connection.

PPP_PHASE_ESTABLISH

```
#define PPP_PHASE_ESTABLISH 0x02
```

Establishing connection.

PPP_PHASE_NETWORK

```
#define PPP_PHASE_NETWORK 0x04
```

Established and working.

PPP_PHASE_TERMINATE

```
#define PPP_PHASE_TERMINATE 0x05
```

Tearing down the link.

7.84 PPP link configuration flags

Macros

- #define PPP_FLAG_AUTH_PAP 0x00000001
  PAP authentication.
- #define PPP_FLAG_AUTH_CHAP 0x00000002
  CHAP authentication.
- #define PPP_FLAG_PCOMP 0x00000004
  Protocol compression.
- #define PPP_FLAG_ACCOMP 0x00000008
  Addr/ctrl compression.
- #define PPP_FLAG_MAGIC_NUMBER 0x00000010
  Use magic numbers.
- #define PPP_FLAG_WANT_MRU 0x00000020
  Specify MRU.
- #define PPP_FLAG_NO_ACCM 0x00000040
  No ctl character map.

7.84.1 Detailed Description

This list defines the flags we can negotiate during link establishment.

7.84.2 Macro Definition Documentation

PPP_FLAG_ACCOMP

```
#define PPP_FLAG_ACCOMP 0x00000008
```

Addr/ctrl compression.

PPP_FLAG_AUTH_CHAP

```
#define PPP_FLAG_AUTH_CHAP 0x00000002
```

CHAP authentication.

PPP_FLAG_AUTH_PAP

```
#define PPP_FLAG_AUTH_PAP 0x00000001
```

PAP authentication.

PPP_FLAG_MAGIC_NUMBER

```
#define PPP_FLAG_MAGIC_NUMBER 0x00000010
```

Use magic numbers.

PPP_FLAG_NO_ACCM

```
#define PPP_FLAG_NO_ACCM 0x00000040
```

No ctl character map.

PPP_FLAG_PCOMP

```
#define PPP_FLAG_PCOMP 0x00000004
```

Protocol compression.

PPP_FLAG_WANT_MRU

```
#define PPP_FLAG_WANT_MRU 0x00000020
```

Specify MRU.

7.85 PVR U/V data format control

Macros

- #define PVR_UVFMT_32BIT 0
  32-bit floating point U/V
- #define PVR_UVFMT_16BIT 1
  16-bit floating point U/V

7.85.1 Detailed Description

7.85.2 Macro Definition Documentation

PVR_UVFMT_16BIT

```
#define PVR_UVFMT_16BIT 1
```

16-bit floating point U/V

PVR_UVFMT_32BIT

```
#define PVR_UVFMT_32BIT 0
```

32-bit floating point U/V

7.86 PVR blending modes

Macros

- #define PVR_BLEND_ZERO 0
  None of this color.
- #define PVR_BLEND_ONE 1
  All of this color.
- #define PVR_BLEND_DESTCOLOR 2
  Destination color.
- #define PVR_BLEND_INVDESTCOLOR 3
  Inverse of destination color.
- #define PVR_BLEND_SRCALPHA 4
  Blend with source alpha.
- #define PVR_BLEND_INVSRCALPHA 5
  Blend with inverse source alpha.
- #define PVR_BLEND_DESTALPHA 6
  Blend with destination alpha.
- #define PVR_BLEND_INVDESTALPHA 7
  Blend with inverse destination alpha.

7.86.1 Detailed Description

These are all the blending modes that can be done with regard to alpha blending on the PVR.

7.86.2 Macro Definition Documentation

PVR_BLEND_DESTALPHA

```
#define PVR_BLEND_DESTALPHA 6
```

Blend with destination alpha.

PVR_BLEND_DESTCOLOR

```
#define PVR_BLEND_DESTCOLOR 2
```

Destination color.

PVR_BLEND_INVDESTALPHA

```
#define PVR_BLEND_INVDESTALPHA 7
```

Blend with inverse destination alpha.

PVR_BLEND_INVDESTCOLOR

```
#define PVR_BLEND_INVDESTCOLOR 3
```

Inverse of destination color.

PVR_BLEND_INVSRCALPHA

```
#define PVR_BLEND_INVSRCALPHA 5
```

Blend with inverse source alpha.

PVR_BLEND_ONE

```
#define PVR_BLEND_ONE 1
```

All of this color.

PVR_BLEND_SRCALPHA

```
#define PVR_BLEND_SRCALPHA 4
```

Blend with source alpha.

PVR_BLEND_ZERO

```
#define PVR_BLEND_ZERO 0
```

None of this color.

7.87 PVR clipping modes

Macros

- #define PVR_USERCLIP_DISABLE 0
  Disable clipping.
- #define PVR_USERCLIP_INSIDE 2
  Enable clipping inside area.
- #define PVR_USERCLIP_OUTSIDE 3
  Enable clipping outside area.

7.87.1 Detailed Description

These control how primitives are clipped against the user clipping area.

7.87.2 Macro Definition Documentation

PVR_USERCLIP_DISABLE

```
#define PVR_USERCLIP_DISABLE 0
```

Disable clipping.

PVR_USERCLIP_INSIDE

```
#define PVR_USERCLIP_INSIDE 2
```

Enable clipping inside area.

PVR_USERCLIP_OUTSIDE

```
#define PVR_USERCLIP_OUTSIDE 3
```

Enable clipping outside area.

7.88 PVR culling modes

Macros

- #define PVR_CULLING_NONE 0
  Disable culling.
- #define PVR_CULLING_SMALL 1
  Cull if small.
- #define PVR_CULLING_CCW 2
  Cull if counterclockwise.
- #define PVR_CULLING_CW 3
  Cull if clockwise.

7.88.1 Detailed Description

These culling modes can be set by polygons to determine when they are culled. They work pretty much as you'd expect them to if you've ever used any 3D hardware before.

7.88.2 Macro Definition Documentation

PVR_CULLING_CCW

```
#define PVR_CULLING_CCW 2
```

Cull if counterclockwise.

PVR_CULLING_CW

```
#define PVR_CULLING_CW 3
```

Cull if clockwise.

PVR_CULLING_NONE

```
#define PVR_CULLING_NONE 0
```

Disable culling.

PVR_CULLING_SMALL

```
#define PVR_CULLING_SMALL 1
```

Cull if small.

7.89 PVR depth comparison modes

Macros

- #define PVR_DEPTHCMP_NEVER 0
  Never pass.
- #define PVR_DEPTHCMP_LESS 1
  Less than.
- #define PVR_DEPTHCMP_EQUAL 2
  Equal to.
- #define PVR_DEPTHCMP_LEQUAL 3
  Less than or equal to.
- #define PVR_DEPTHCMP_GREATER 4
  Greater than.
- #define PVR_DEPTHCMP_NOTEQUAL 5
  Not equal to.
- #define PVR_DEPTHCMP_GEQUAL 6
  Greater than or equal to.
- #define PVR_DEPTHCMP_ALWAYS 7
  Always pass.

7.89.1 Detailed Description

These set the depth function used for comparisons.

7.89.2 Macro Definition Documentation

PVR_DEPTHCMP_ALWAYS

```
#define PVR_DEPTHCMP_ALWAYS 7
```

Always pass.

PVR_DEPTHCMP_EQUAL

```
#define PVR_DEPTHCMP_EQUAL 2
```

Equal to.

PVR_DEPTHCMP_GEQUAL

```
#define PVR_DEPTHCMP_GEQUAL 6
```

Greater than or equal to.

PVR_DEPTHCMP_GREATER

```
#define PVR_DEPTHCMP_GREATER 4
```

Greater than.

PVR_DEPTHCMP_LEQUAL

```
#define PVR_DEPTHCMP_LEQUAL 3
```

Less than or equal to.

PVR_DEPTHCMP_LESS

```
#define PVR_DEPTHCMP_LESS 1
```

Less than.

PVR_DEPTHCMP_NEVER

```
#define PVR_DEPTHCMP_NEVER 0
```

Never pass.

PVR_DEPTHCMP_NOTEQUAL

```
#define PVR_DEPTHCMP_NOTEQUAL 5
```

Not equal to.

7.90 PVR fog modes

Macros

- #define PVR_FOG_TABLE 0
  Table fog.
- #define PVR_FOG_VERTEX 1
  Vertex fog.
- #define PVR_FOG_DISABLE 2
  Disable fog.
- #define PVR_FOG_TABLE2 3
  Table fog mode 2.

7.90.1 Detailed Description

Each polygon can decide what fog type is used with regard to it using these constants in its pvr_poly_cxt_t.

7.90.2 Macro Definition Documentation

PVR_FOG_DISABLE

```
#define PVR_FOG_DISABLE 2
```

Disable fog.

PVR_FOG_TABLE

```
#define PVR_FOG_TABLE 0
```

Table fog.

PVR_FOG_TABLE2

```
#define PVR_FOG_TABLE2 3
```

Table fog mode 2.

PVR_FOG_VERTEX

```
#define PVR_FOG_VERTEX 1
```

Vertex fog.

7.91 PVR mipmap bias modes

Macros

- #define PVR_MIPBIAS_NORMAL PVR_MIPBIAS_1_00 /* txr_mipmap_bias */
- #define PVR_MIPBIAS_0_25 1
- #define PVR_MIPBIAS_0_50 2
- #define PVR_MIPBIAS_0_75 3
- #define PVR_MIPBIAS_1_00 4
- #define PVR_MIPBIAS_1_25 5
- #define PVR_MIPBIAS_1_50 6
- #define PVR_MIPBIAS_1_75 7
- #define PVR_MIPBIAS_2_00 8
- #define PVR_MIPBIAS_2_25 9
- #define PVR_MIPBIAS_2_50 10
- #define PVR_MIPBIAS_2_75 11
- #define PVR_MIPBIAS_3_00 12
- #define PVR_MIPBIAS_3_25 13
- #define PVR_MIPBIAS_3_50 14
- #define PVR_MIPBIAS_3_75 15

7.91.1 Detailed Description

7.91.2 Macro Definition Documentation

PVR_MIPBIAS_0_25

```
#define PVR_MIPBIAS_0_25 1
```

PVR_MIPBIAS_0_50

```
#define PVR_MIPBIAS_0_50 2
```

PVR_MIPBIAS_0_75

```
#define PVR_MIPBIAS_0_75 3
```

PVR_MIPBIAS_1_00

```
#define PVR_MIPBIAS_1_00 4
```

PVR_MIPBIAS_1_25

```
#define PVR_MIPBIAS_1_25 5
```

PVR_MIPBIAS_1_50

```
#define PVR_MIPBIAS_1_50 6
```

PVR_MIPBIAS_1_75

```
#define PVR_MIPBIAS_1_75 7
```

PVR_MIPBIAS_2_00

```
#define PVR_MIPBIAS_2_00 8
```

PVR_MIPBIAS_2_25

```
#define PVR_MIPBIAS_2_25 9
```

PVR_MIPBIAS_2_50

```
#define PVR_MIPBIAS_2_50 10
```

PVR_MIPBIAS_2_75

```
#define PVR_MIPBIAS_2_75 11
```

PVR_MIPBIAS_3_00

```
#define PVR_MIPBIAS_3_00 12
```

PVR_MIPBIAS_3_25

```
#define PVR_MIPBIAS_3_25 13
```

PVR_MIPBIAS_3_50

```
#define PVR_MIPBIAS_3_50 14
```

PVR_MIPBIAS_3_75

```
#define PVR_MIPBIAS_3_75 15
```

PVR_MIPBIAS_NORMAL

```
#define PVR_MIPBIAS_NORMAL PVR_MIPBIAS_1_00 /* txr_mipmap_bias */
```

7.92 PVR palette formats

Macros

- #define PVR_PAL_ARGB1555 0
  16-bit ARGB1555 palette format
- #define PVR_PAL_RGB565 1
  16-bit RGB565 palette format
- #define PVR_PAL_ARGB4444 2
  16-bit ARGB4444 palette format
- #define PVR_PAL_ARGB8888 3
  32-bit ARGB8888 palette format

7.92.1 Detailed Description

Entries in the PVR's palettes can be of any of these formats. Note that you can only have one format active at a time.

7.92.2 Macro Definition Documentation

PVR_PAL_ARGB1555

```
#define PVR_PAL_ARGB1555 0
```

16-bit ARGB1555 palette format

PVR_PAL_ARGB4444

```
#define PVR_PAL_ARGB4444 2
```

16-bit ARGB4444 palette format

PVR_PAL_ARGB8888

```
#define PVR_PAL_ARGB8888 3
```

32-bit ARGB8888 palette format

PVR_PAL_RGB565

```
#define PVR_PAL_RGB565 1
```

16-bit RGB565 palette format

7.93 PVR primitive list types

Macros

- #define PVR_LIST_OP_POLY 0
  Opaque polygon list.
- #define PVR_LIST_OP_MOD 1
  Opaque modifier list.
- #define PVR_LIST_TR_POLY 2
  Translucent polygon list.
- #define PVR_LIST_TR_MOD 3
  Translucent modifier list.
- #define PVR_LIST_PT_POLY 4
  Punch-thru polygon list.

7.93.1 Detailed Description

Each primitive submitted to the PVR must be placed in one of these lists, depending on its characteristics.

7.93.2 Macro Definition Documentation

PVR_LIST_OP_MOD

```
#define PVR_LIST_OP_MOD 1
```

Opaque modifier list.

PVR_LIST_OP_POLY

```
#define PVR_LIST_OP_POLY 0
```

Opaque polygon list.

PVR_LIST_PT_POLY

```
#define PVR_LIST_PT_POLY 4
```

Punch-thru polygon list.

PVR_LIST_TR_MOD

```
#define PVR_LIST_TR_MOD 3
```

Translucent modifier list.

PVR_LIST_TR_POLY

```
#define PVR_LIST_TR_POLY 2
```

Translucent polygon list.

7.94 PVR shading modes

Macros

- #define PVR_SHADE_FLAT 0

  Use flat shading.
- #define PVR_SHADE_GOURAUD 1

  Use Gouraud shading.

7.94.1 Detailed Description

Each polygon can define how it wants to be shaded, be it with flat or Gouraud shading using these constants in the appropriate place in its pvr_poly_cxt_t.

7.94.2 Macro Definition Documentation

PVR_SHADE_FLAT

```
#define PVR_SHADE_FLAT 0
```

Use flat shading.

PVR_SHADE_GOURAUD

```
#define PVR_SHADE_GOURAUD 1
```

Use Gouraud shading.

7.95 PVR texture formats

Macros

- #define PVR_TXRFMT_NONE 0
  No texture.
- #define PVR_TXRFMT_VQ_DISABLE (0 << 30)
  Not VQ encoded.
- #define PVR_TXRFMT_VQ_ENABLE (1 << 30)
  VQ encoded.
- #define PVR_TXRFMT_ARGB1555 (0 << 27)
  16-bit ARGB1555
- #define PVR_TXRFMT_RGB565 (1 << 27)
  16-bit RGB565
- #define PVR_TXRFMT_ARGB4444 (2 << 27)
  16-bit ARGB4444
- #define PVR_TXRFMT_YUV422 (3 << 27)
  YUV422 format.
- #define PVR_TXRFMT_BUMP (4 << 27)
  Bumpmap format.
- #define PVR_TXRFMT_PAL4BPP (5 << 27)
  4BPP paletted format
- #define PVR_TXRFMT_PAL8BPP (6 << 27)
  8BPP paletted format
- #define PVR_TXRFMT_TWIDDLED (0 << 26)
  Texture is twiddled.
- #define PVR_TXRFMT_NONTWIDDLED (1 << 26)
  Texture is not twiddled.
- #define PVR_TXRFMT_NOSTRIDE (0 << 21)
  Texture is not strided.
- #define PVR_TXRFMT_STRIDE (1 << 21)
  Texture is strided.
- #define PVR_TXRFMT_8BPP_PAL(x) ((x) << 25)
  8BPP palette selector
- #define PVR_TXRFMT_4BPP_PAL(x) ((x) << 21)
  4BPP palette selector

7.95.1 Detailed Description

These are the texture formats that the PVR supports. Note that some of these, you can OR together with other values.

7.95.2 Macro Definition Documentation

PVR_TXRFMT_4BPP_PAL

```
#define PVR_TXRFMT_4BPP_PAL(
              x ) ((x) << 21)
```

4BPP palette selector

Parameters

x	The palette index

PVR_TXRFMT_8BPP_PAL

```
#define PVR_TXRFMT_8BPP_PAL(
              x ) ((x) << 25)
```

8BPP palette selector

Parameters

x	The palette index

PVR_TXRFMT_ARGB1555

```
#define PVR_TXRFMT_ARGB1555 (0 << 27)
```

16-bit ARGB1555

PVR_TXRFMT_ARGB4444

```
#define PVR_TXRFMT_ARGB4444 (2 << 27)
```

16-bit ARGB4444

PVR_TXRFMT_BUMP

```
#define PVR_TXRFMT_BUMP (4 << 27)
```

Bumpmap format.

PVR_TXRFMT_NONE

```
#define PVR_TXRFMT_NONE 0
```

No texture.

PVR_TXRFMT_NONTWIDDLED

```
#define PVR_TXRFMT_NONTWIDDLED (1 << 26)
```

Texture is not twiddled.

PVR_TXRFMT_NOSTRIDE

```
#define PVR_TXRFMT_NOSTRIDE (0 << 21)
```

Texture is not strided.

PVR_TXRFMT_PAL4BPP

```
#define PVR_TXRFMT_PAL4BPP (5 << 27)
```

4BPP paletted format

PVR_TXRFMT_PAL8BPP

```
#define PVR_TXRFMT_PAL8BPP (6 << 27)
```

8BPP paletted format

PVR_TXRFMT_RGB565

```
#define PVR_TXRFMT_RGB565 (1 << 27)
```

16-bit RGB565

PVR_TXRFMT_STRIDE

```
#define PVR_TXRFMT_STRIDE (1 << 21)
```

Texture is strided.

PVR_TXRFMT_TWIDDLED

```
#define PVR_TXRFMT_TWIDDLED (0 << 26)
```

Texture is twiddled.

PVR_TXRFMT_VQ_DISABLE

```
#define PVR_TXRFMT_VQ_DISABLE (0 << 30)
```

Not VQ encoded.

PVR_TXRFMT_VQ_ENABLE

```
#define PVR_TXRFMT_VQ_ENABLE (1 << 30)
```

VQ encoded.

PVR_TXRFMT_YUV422

```
#define PVR_TXRFMT_YUV422 (3 << 27)
```

YUV422 format.

7.96 PVR texture sampling modes

Macros

- #define PVR_FILTER_NONE 0
  No filtering (point sample)
- #define PVR_FILTER_NEAREST 0
  No filtering (point sample)
- #define PVR_FILTER_BILINEAR 2
  Bilinear interpolation.
- #define PVR_FILTER_TRILINEAR1 4
  Trilinear interpolation pass 1.
- #define PVR_FILTER_TRILINEAR2 6
  Trilinear interpolation pass 2.

7.96.1 Detailed Description

7.96.2 Macro Definition Documentation

PVR_FILTER_BILINEAR

```
#define PVR_FILTER_BILINEAR 2
```

Bilinear interpolation.

PVR_FILTER_NEAREST

```
#define PVR_FILTER_NEAREST 0
```

No filtering (point sample)

PVR_FILTER_NONE

```
#define PVR_FILTER_NONE 0
```

No filtering (point sample)

PVR_FILTER_TRILINEAR1

```
#define PVR_FILTER_TRILINEAR1 4
```

Trilinear interpolation pass 1.

PVR_FILTER_TRILINEAR2

```
#define PVR_FILTER_TRILINEAR2 6
```

Trilinear interpolation pass 2.

7.97 PVR vertex color formats

Macros

- #define PVR_CLRFMT_ARGBPACKED 0
  32-bit integer ARGB
- #define PVR_CLRFMT_4FLOATS 1
  4 floating point values
- #define PVR_CLRFMT_INTENSITY 2
  Intensity color.
- #define PVR_CLRFMT_INTENSITY_PREV 3
  Use last intensity.

7.97.1 Detailed Description

These control how colors are represented in polygon data.

7.97.2 Macro Definition Documentation

PVR_CLRFMT_4FLOATS

```
#define PVR_CLRFMT_4FLOATS 1
```

4 floating point values

PVR_CLRFMT_ARGBPACKED

```
#define PVR_CLRFMT_ARGBPACKED 0
```

32-bit integer ARGB

PVR_CLRFMT_INTENSITY

```
#define PVR_CLRFMT_INTENSITY 2
```

Intensity color.

PVR_CLRFMT_INTENSITY_PREV

```
#define PVR_CLRFMT_INTENSITY_PREV 3
```

Use last intensity.

7.98 Partitions available in the flashrom

Macros

- #define FLASHROM_PT_SYSTEM 0
  Factory settings (read-only, 8K)
- #define FLASHROM_PT_RESERVED 1
  reserved (all 0s, 8K)
- #define FLASHROM_PT_BLOCK_1 2
  Block allocated (16K)
- #define FLASHROM_PT_SETTINGS 3
  Game settings (block allocated, 32K)
- #define FLASHROM_PT_BLOCK_2 4
  Block allocated (64K)

7.98.1 Detailed Description

7.98.2 Macro Definition Documentation

FLASHROM_PT_BLOCK_1

```
#define FLASHROM_PT_BLOCK_1 2
```

Block allocated (16K)

FLASHROM_PT_BLOCK_2

```
#define FLASHROM_PT_BLOCK_2 4
```

Block allocated (64K)

FLASHROM_PT_RESERVED

```
#define FLASHROM_PT_RESERVED 1
```

reserved (all 0s, 8K)

FLASHROM_PT_SETTINGS

```
#define FLASHROM_PT_SETTINGS 3
```

Game settings (block allocated, 32K)

FLASHROM_PT_SYSTEM

```
#define FLASHROM_PT_SYSTEM 0
```

Factory settings (read-only, 8K)

7.99 Performance Counters

Modules

- Performance Counter Modes

Macros

- #define PRFC0 0

  SH4 Performance Counter.
- #define PRFC1 1

  SH4 Performance Counter.
- #define PMCR_COUNT_CPU_CYCLES 0

  CPU Cycles Count Type.
- #define PMCR_COUNT_RATIO_CYCLES 1

  Ratio Cycles Count Type.

Functions

- uint16 perf_cntr_get_config (int which)

  Get a performance counter's settings.
- int perf_cntr_start (int which, int mode, int count_type)

  Start a performance counter.
- int perf_cntr_stop (int which)

  Stop a performance counter.
- int perf_cntr_clear (int which)

  Clear a performance counter.
- uint64 perf_cntr_count (int which)

  Obtain the count of a performance counter.
- void timer_ns_enable (void)

  Enable the nanosecond timer.
- void timer_ns_disable (void)

  Disable the nanosecond timer.
- uint64 timer_ns_gettime64 (void)

  Get the current uptime of the system (in nanoseconds).

7.99.1 Detailed Description

The performance counter API exposes the SH4's hardware profiling registers, which consist of two different sets of independently operable 64-bit counters.

7.99.2 Macro Definition Documentation

PMCR_COUNT_CPU_CYCLES

```
#define PMCR_COUNT_CPU_CYCLES 0
```

CPU Cycles Count Type.

Count cycles. At 5 ns increments, a 48-bit cycle counter can run continuously for 16.33 days.

PMCR_COUNT_RATIO_CYCLES

```
#define PMCR_COUNT_RATIO_CYCLES 1
```

Ratio Cycles Count Type.

CPU/bus ratio mode where cycles (where T = C x B / 24 and T is time, C is count, and B is time of one bus cycle).

PRFC0

```
#define PRFC0 0
```

SH4 Performance Counter.

This counter is used by the ns_gettime function in this header.

PRFC1

```
#define PRFC1 1
```

SH4 Performance Counter.

A counter that is not used by KOS.

7.99.3 Function Documentation

perf_cntr_clear()

```
int perf_cntr_clear (
            int which )
```

Clear a performance counter.

This function clears a performance counter. It resets its count to zero. This function stops the counter before clearing it because you cant clear a running counter.

Parameters

which	The counter to clear (i.e, PRFC0 or PRFC1).

Return values

0	On success.

perf_cntr_count()

```
uint64 perf_cntr_count (
            int which )
```

Obtain the count of a performance counter.

This function simply returns the count of the counter.

Parameters

which	The counter to read (i.e, PRFC0 or PRFC1).

Returns

The counter's count.

perf_cntr_get_config()

```
uint16 perf_cntr_get_config (
            int which )
```

Get a performance counter's settings.

This function returns a performance counter's settings.

Parameters

which	The performance counter (i.e, PRFC0 or PRFC1).

Return values

0	On success.

perf_cntr_start()

```
int perf_cntr_start (
            int which,
            int mode,
            int count_type )
```

Start a performance counter.

This function starts a performance counter

Parameters

which	The counter to start (i.e, PRFC0 or PRFC1).
mode	Use one of the 33 modes listed above.
count_type	PMCR_COUNT_CPU_CYCLES or PMCR_COUNT_RATIO_CYCLES.

Return values

0	On success.

perf_cntr_stop()

```
int perf_cntr_stop (
            int which )
```

Stop a performance counter.

This function stops a performance counter that was started with perf_cntr_start(). Stopping a counter retains its count. To clear the count use perf_cntr_clear().

Parameters

which	The counter to stop (i.e, PRFC0 or PRFC1).

Return values

0	On success.

timer_ns_disable()

```
void timer_ns_disable (
            void  )
```

Disable the nanosecond timer.

This function disables the performance counter used for the timer_ns_gettime64() function. Generally, you will not want to do this, unless you have some need to use the counter PRFC0 for something else.

timer_ns_enable()

```
void timer_ns_enable (
            void  )
```

Enable the nanosecond timer.

This function enables the performance counter used for the timer_ns_gettime64() function. This is on by default. The function uses PRFC0 to do the work.

timer_ns_gettime64()

```
uint64 timer_ns_gettime64 (
            void  )
```

Get the current uptime of the system (in nanoseconds).

This function retrieves the number of nanoseconds since KOS was started.

Returns

The number of nanoseconds since KOS started.

7.99.4 Performance Counter Modes

Macros

- #define PMCR_INIT_NO_MODE 0x00

  None; Just here to be complete.
- #define PMCR_OPERAND_READ_ACCESS_MODE 0x01

  Quantity; With cache.
- #define PMCR_OPERAND_WRITE_ACCESS_MODE 0x02

  Quantity; With cache.
- #define PMCR_UTLB_MISS_MODE 0x03

  Quantity.
- #define PMCR_OPERAND_CACHE_READ_MISS_MODE 0x04

  Quantity.
- #define PMCR_OPERAND_CACHE_WRITE_MISS_MODE 0x05

  Quantity.
- #define PMCR_INSTRUCTION_FETCH_MODE 0x06

  Quantity; With cache.
- #define PMCR_INSTRUCTION_TLB_MISS_MODE 0x07

  Quantity.
- #define PMCR_INSTRUCTION_CACHE_MISS_MODE 0x08

  Quantity.
- #define PMCR_ALL_OPERAND_ACCESS_MODE 0x09

  Quantity.
- #define PMCR_ALL_INSTRUCTION_FETCH_MODE 0x0a

  Quantity.
- #define PMCR_ON_CHIP_RAM_OPERAND_ACCESS_MODE 0x0b

  Quantity.
- #define PMCR_ON_CHIP_IO_ACCESS_MODE 0x0d

  Quantity.
- #define PMCR_OPERAND_ACCESS_MODE 0x0e

  Quantity; With cache, counts both reads and writes.
- #define PMCR_OPERAND_CACHE_MISS_MODE 0x0f

  Quantity.

- #define PMCR_BRANCH_ISSUED_MODE 0x10

  Quantity; Not the same as branch taken!
- #define PMCR_BRANCH_TAKEN_MODE 0x11

  Quantity.
- #define PMCR_SUBROUTINE_ISSUED_MODE 0x12

  Quantity; Issued a BSR, BSRF, JSR, JSR/N.
- #define PMCR_INSTRUCTION_ISSUED_MODE 0x13

  Quantity.
- #define PMCR_PARALLEL_INSTRUCTION_ISSUED_MODE 0x14

  Quantity.
- #define PMCR_FPU_INSTRUCTION_ISSUED_MODE 0x15

  Quantity.
- #define PMCR_INTERRUPT_COUNTER_MODE 0x16

  Quantity.
- #define PMCR_NMI_COUNTER_MODE 0x17

  Quantity.
- #define PMCR_TRAPA_INSTRUCTION_COUNTER_MODE 0x18

  Quantity.
- #define PMCR_UBC_A_MATCH_MODE 0x19

  Quantity.
- #define PMCR_UBC_B_MATCH_MODE 0x1a

  Quantity.
- #define PMCR_INSTRUCTION_CACHE_FILL_MODE 0x21

  Cycles.
- #define PMCR_OPERAND_CACHE_FILL_MODE 0x22

  Cycles.
- #define PMCR_ELAPSED_TIME_MODE 0x23

  Cycles; For 200MHz CPU: 5ns per count in 1 cycle = 1 count mode, or around 417.715ps per count (increments by 12) in CPU/bus ratio mode.
- #define PMCR_PIPELINE_FREEZE_BY_ICACHE_MISS_MODE 0x24

  Cycles.
- #define PMCR_PIPELINE_FREEZE_BY_DCACHE_MISS_MODE 0x25

  Cycles.
- #define PMCR_PIPELINE_FREEZE_BY_BRANCH_MODE 0x27

  Cycles.
- #define PMCR_PIPELINE_FREEZE_BY_CPU_REGISTER_MODE 0x28

  Cycles.
- #define PMCR_PIPELINE_FREEZE_BY_FPU_MODE 0x29

  Cycles.

7.99.4.1 Detailed Description

This is the list of modes that are allowed to be passed into the perf_cntr_start() function, representing different things you want to count.

7.99.4.2 Macro Definition Documentation

PMCR_ALL_INSTRUCTION_FETCH_MODE

```
#define PMCR_ALL_INSTRUCTION_FETCH_MODE 0x0a
```

Quantity.

PMCR_ALL_OPERAND_ACCESS_MODE

```
#define PMCR_ALL_OPERAND_ACCESS_MODE 0x09
```

Quantity.

PMCR_BRANCH_ISSUED_MODE

```
#define PMCR_BRANCH_ISSUED_MODE 0x10
```

Quantity; Not the same as branch taken!

PMCR_BRANCH_TAKEN_MODE

```
#define PMCR_BRANCH_TAKEN_MODE 0x11
```

Quantity.

PMCR_ELAPSED_TIME_MODE

```
#define PMCR_ELAPSED_TIME_MODE 0x23
```

Cycles; For 200MHz CPU: 5ns per count in 1 cycle = 1 count mode, or around 417.715ps per count (increments by 12) in CPU/bus ratio mode.

PMCR_FPU_INSTRUCTION_ISSUED_MODE

```
#define PMCR_FPU_INSTRUCTION_ISSUED_MODE 0x15
```

Quantity.

PMCR_INIT_NO_MODE

```
#define PMCR_INIT_NO_MODE 0x00
```

None; Just here to be complete.

PMCR_INSTRUCTION_CACHE_FILL_MODE

```
#define PMCR_INSTRUCTION_CACHE_FILL_MODE 0x21
```

Cycles.

PMCR_INSTRUCTION_CACHE_MISS_MODE

```
#define PMCR_INSTRUCTION_CACHE_MISS_MODE 0x08
```

Quantity.

PMCR_INSTRUCTION_FETCH_MODE

```
#define PMCR_INSTRUCTION_FETCH_MODE 0x06
```

Quantity; With cache.

PMCR_INSTRUCTION_ISSUED_MODE

```
#define PMCR_INSTRUCTION_ISSUED_MODE 0x13
```

Quantity.

PMCR_INSTRUCTION_TLB_MISS_MODE

```
#define PMCR_INSTRUCTION_TLB_MISS_MODE 0x07
```

Quantity.

PMCR_INTERRUPT_COUNTER_MODE

```
#define PMCR_INTERRUPT_COUNTER_MODE 0x16
```

Quantity.

PMCR_NMI_COUNTER_MODE

```
#define PMCR_NMI_COUNTER_MODE 0x17
```

Quantity.

PMCR_ON_CHIP_IO_ACCESS_MODE

```
#define PMCR_ON_CHIP_IO_ACCESS_MODE 0x0d
```

Quantity.

PMCR_ON_CHIP_RAM_OPERAND_ACCESS_MODE

```
#define PMCR_ON_CHIP_RAM_OPERAND_ACCESS_MODE 0x0b
```

Quantity.

PMCR_OPERAND_ACCESS_MODE

```
#define PMCR_OPERAND_ACCESS_MODE 0x0e
```

Quantity; With cache, counts both reads and writes.

PMCR_OPERAND_CACHE_FILL_MODE

```
#define PMCR_OPERAND_CACHE_FILL_MODE 0x22
```

Cycles.

PMCR_OPERAND_CACHE_MISS_MODE

```
#define PMCR_OPERAND_CACHE_MISS_MODE 0x0f
```

Quantity.

PMCR_OPERAND_CACHE_READ_MISS_MODE

```
#define PMCR_OPERAND_CACHE_READ_MISS_MODE 0x04
```

Quantity.

PMCR_OPERAND_CACHE_WRITE_MISS_MODE

```
#define PMCR_OPERAND_CACHE_WRITE_MISS_MODE 0x05
```

Quantity.

PMCR_OPERAND_READ_ACCESS_MODE

```
#define PMCR_OPERAND_READ_ACCESS_MODE 0x01
```

Quantity; With cache.

PMCR_OPERAND_WRITE_ACCESS_MODE

```
#define PMCR_OPERAND_WRITE_ACCESS_MODE 0x02
```

Quantity; With cache.

PMCR_PARALLEL_INSTRUCTION_ISSUED_MODE

```
#define PMCR_PARALLEL_INSTRUCTION_ISSUED_MODE 0x14
```

Quantity.

PMCR_PIPELINE_FREEZE_BY_BRANCH_MODE

```
#define PMCR_PIPELINE_FREEZE_BY_BRANCH_MODE 0x27
```

Cycles.

PMCR_PIPELINE_FREEZE_BY_CPU_REGISTER_MODE

```
#define PMCR_PIPELINE_FREEZE_BY_CPU_REGISTER_MODE 0x28
```

Cycles.

PMCR_PIPELINE_FREEZE_BY_DCACHE_MISS_MODE

```
#define PMCR_PIPELINE_FREEZE_BY_DCACHE_MISS_MODE 0x25
```

Cycles.

PMCR_PIPELINE_FREEZE_BY_FPU_MODE

```
#define PMCR_PIPELINE_FREEZE_BY_FPU_MODE 0x29
```

Cycles.

PMCR_PIPELINE_FREEZE_BY_ICACHE_MISS_MODE

```
#define PMCR_PIPELINE_FREEZE_BY_ICACHE_MISS_MODE 0x24
```

Cycles.

PMCR_SUBROUTINE_ISSUED_MODE

```
#define PMCR_SUBROUTINE_ISSUED_MODE 0x12
```

Quantity; Issued a BSR, BSRF, JSR, JSR/N.

PMCR_TRAPA_INSTRUCTION_COUNTER_MODE

```
#define PMCR_TRAPA_INSTRUCTION_COUNTER_MODE 0x18
```

Quantity.

PMCR_UBC_A_MATCH_MODE

```
#define PMCR_UBC_A_MATCH_MODE 0x19
```

Quantity.

PMCR_UBC_B_MATCH_MODE

```
#define PMCR_UBC_B_MATCH_MODE 0x1a
```

Quantity.

PMCR_UTLB_MISS_MODE

```
#define PMCR_UTLB_MISS_MODE 0x03
```

Quantity.

7.100 Potential exit paths from the kernel on

Macros

- #define ARCH_EXIT_RETURN 1

  Return to loader.
- #define ARCH_EXIT_MENU 2

  Return to system menu.
- #define ARCH_EXIT_REBOOT 3

  Reboot the machine.

7.100.1 Detailed Description

arch_exit()

7.100.2 Macro Definition Documentation

ARCH_EXIT_MENU

```
#define ARCH_EXIT_MENU 2
```

Return to system menu.

ARCH_EXIT_REBOOT

```
#define ARCH_EXIT_REBOOT 3
```

Reboot the machine.

ARCH_EXIT_RETURN

```
#define ARCH_EXIT_RETURN 1
```

Return to loader.

7.101 RTL8139C Command Bits

Macros

- #define RT_CMD_RESET 0x10
  Reset the RTL8139C.
- #define RT_CMD_RX_ENABLE 0x08
  Enable Rx.
- #define RT_CMD_TX_ENABLE 0x04
  Enable Tx.
- #define RT_CMD_RX_BUF_EMPTY 0x01
  Empty the Rx buffer.

7.101.1 Detailed Description

OR appropriate bit values together and write into the RT_CHIPCMD register to execute the command.

7.101.2 Macro Definition Documentation

RT_CMD_RESET

```
#define RT_CMD_RESET 0x10
```

Reset the RTL8139C.

RT_CMD_RX_BUF_EMPTY

```
#define RT_CMD_RX_BUF_EMPTY 0x01
```

Empty the Rx buffer.

RT_CMD_RX_ENABLE

```
#define RT_CMD_RX_ENABLE 0x08
```

Enable Rx.

RT_CMD_TX_ENABLE

```
#define RT_CMD_TX_ENABLE 0x04
```

Enable Tx.

7.102 RTL8139C Config Register 1 bits

Macros

- #define RT_CONFIG1_LED1 0x80
  XXX DC bba has no LED, maybe repurposed.
- #define RT_CONFIG1_LED0 0x40
  XXX DC bba has no LED, maybe repurposed.
- #define RT_CONFIG1_DVRLOAD 0x20
  Sets the Driver as loaded.
- #define RT_CONFIG1_LWACT 0x10
  LWAKE active mode. Default 0.
- #define RT_CONFIG1_MEMMAP 0x08
  Registers mapped to PCI mem space. Read Only.
- #define RT_CONFIG1_IOMAP 0x04
  Registers mapped to PCI I/O space. Read Only.
- #define RT_CONFIG1_VPD 0x02
  Enable Vital Product Data.
- #define RT_CONFIG1_PMEn 0x01
  Power Management Enable.

7.102.1 Detailed Description

From RTL8139C(L) datasheet v1.4

7.102.2 Macro Definition Documentation

RT_CONFIG1_DVRLOAD

```
#define RT_CONFIG1_DVRLOAD 0x20
```

Sets the Driver as loaded.

RT_CONFIG1_IOMAP

```
#define RT_CONFIG1_IOMAP 0x04
```

Registers mapped to PCI I/O space. Read Only.

RT_CONFIG1_LED0

```
#define RT_CONFIG1_LED0 0x40
```

XXX DC bba has no LED, maybe repurposed.

RT_CONFIG1_LED1

```
#define RT_CONFIG1_LED1 0x80
```

XXX DC bba has no LED, maybe repurposed.

RT_CONFIG1_LWACT

```
#define RT_CONFIG1_LWACT 0x10
```

LWAKE active mode. Default 0.

RT_CONFIG1_MEMMAP

```
#define RT_CONFIG1_MEMMAP 0x08
```

Registers mapped to PCI mem space. Read Only.

RT_CONFIG1_PMEn

```
#define RT_CONFIG1_PMEn 0x01
```

Power Management Enable.

RT_CONFIG1_VPD

```
#define RT_CONFIG1_VPD 0x02
```

Enable Vital Product Data.

7.103 RTL8139C Config Register 4 bits

Macros

- #define RT_CONFIG4_RxFIFIOAC 0x80
  Auto-clear the Rx FIFO overflow.
- #define RT_CONFIG4_AnaOff 0x40
  Turn off analog power. Default 0.
- #define RT_CONFIG4_LongWF 0x20
  Long Wake-up Frames.
- #define RT_CONFIG4_LWPME 0x10
  LWake vs PMEB.
- #define RT_CONFIG4_RES08 0x08
  Reserved.
- #define RT_CONFIG4_LWPTN 0x04
  LWAKE Pattern.
- #define RT_CONFIG4_RES02 0x02
  Reserved.
- #define RT_CONFIG4_PBWake 0x01
  Disable pre-Boot Wakeup.

7.103.1 Detailed Description

From RTL8139C(L) datasheet v1.4. Only RT_CONFIG4_RxFIFIOAC is used.

7.103.2 Macro Definition Documentation

RT_CONFIG4_AnaOff

```
#define RT_CONFIG4_AnaOff 0x40
```

Turn off analog power. Default 0.

RT_CONFIG4_LongWF

```
#define RT_CONFIG4_LongWF 0x20
```

Long Wake-up Frames.

RT_CONFIG4_LWPME

```
#define RT_CONFIG4_LWPME 0x10
```

LWake vs PMEB.

RT_CONFIG4_LWPTN

```
#define RT_CONFIG4_LWPTN 0x04
```

LWAKE Pattern.

RT_CONFIG4_PBWake

```
#define RT_CONFIG4_PBWake 0x01
```

Disable pre-Boot Wakeup.

RT_CONFIG4_RES02

```
#define RT_CONFIG4_RES02 0x02
```

Reserved.

RT_CONFIG4_RES08

```
#define RT_CONFIG4_RES08 0x08
```

Reserved.

RT_CONFIG4_RxFIFIOAC

```
#define RT_CONFIG4_RxFIFIOAC 0x80
```

Auto-clear the Rx FIFO overflow.

7.104 RTL8139C Config Register 5 bits

Macros

- #define RT_CONFIG5_RES80 0x80
  Reserved.
- #define RT_CONFIG5_BWF 0x40
  Enable Broadcast Wakeup Frame. Default 0.
- #define RT_CONFIG5_MWF 0x20
  Enable Multicast Wakeup Frame. Default 0.
- #define RT_CONFIG5_UWF 0x10
  Enable Unicast Wakeup Frame. Default 0.
- #define RT_CONFIG5_FIFOAddr 0x08
  Set FIFO address pointer. For testing only.
- #define RT_CONFIG5_LDPS 0x04
  Disable Link Down Power Saving mode.
- #define RT_CONFIG5_LANW 0x02
  Enable LANWake signal.
- #define RT_CONFIG5_PME_STS 0x01
  Allow PCI reset to set PME_Status bit.

7.104.1 Detailed Description

From RTL8139C(L) datasheet v1.4. Only RT_CONFIG5_LDPS is used.

7.104.2 Macro Definition Documentation

RT_CONFIG5_BWF

```
#define RT_CONFIG5_BWF 0x40
```

Enable Broadcast Wakeup Frame. Default 0.

RT_CONFIG5_FIFOAddr

```
#define RT_CONFIG5_FIFOAddr 0x08
```

Set FIFO address pointer. For testing only.

RT_CONFIG5_LANW

```
#define RT_CONFIG5_LANW 0x02
```

Enable LANWake signal.

RT_CONFIG5_LDPS

```
#define RT_CONFIG5_LDPS 0x04
```

Disable Link Down Power Saving mode.

RT_CONFIG5_MWF

```
#define RT_CONFIG5_MWF 0x20
```

Enable Multicast Wakeup Frame. Default 0.

RT_CONFIG5_PME_STS

```
#define RT_CONFIG5_PME_STS 0x01
```

Allow PCI reset to set PME_Status bit.

RT_CONFIG5_RES80

```
#define RT_CONFIG5_RES80 0x80
```

Reserved.

RT_CONFIG5_UWF

```
#define RT_CONFIG5_UWF 0x10
```

Enable Unicast Wakeup Frame. Default 0.

7.105 RTL8139C Interrupt Status bits

Macros

- #define RT_INT_PCIERR 0x8000

  PCI Bus error.
- #define RT_INT_TIMEOUT 0x4000

  Set when TCTR reaches TimerInt value.
- #define RT_INT_RXFIFO_OVERFLOW 0x0040

  Rx FIFO overflow.
- #define RT_INT_RXFIFO_UNDERRUN 0x0020

  Packet underrun / link change.
- #define RT_INT_LINK_CHANGE 0x0020

Packet underrun / link change.

- #define RT_INT_RXBUF_OVERFLOW 0x0010

  Rx BUFFER overflow.
- #define RT_INT_TX_ERR 0x0008

  Tx error.
- #define RT_INT_TX_OK 0x0004

  Tx OK.
- #define RT_INT_RX_ERR 0x0002

  Rx error.
- #define RT_INT_RX_OK 0x0001

  Rx OK.
- #define RT_INT_RX_ACK (RT_INT_RXFIFO_OVERFLOW | RT_INT_RXBUF_OVERFLOW | RT_INT_RX_OK)

  Composite RX bits we check for while doing an RX interrupt.

7.105.1 Detailed Description

7.105.2 Macro Definition Documentation

RT_INT_LINK_CHANGE

```
#define RT_INT_LINK_CHANGE 0x0020
```

Packet underrun / link change.

RT_INT_PCIERR

```
#define RT_INT_PCIERR 0x8000
```

PCI Bus error.

RT_INT_RX_ACK

```
#define RT_INT_RX_ACK (RT_INT_RXFIFO_OVERFLOW | RT_INT_RXBUF_OVERFLOW | RT_INT_RX_OK)
```

Composite RX bits we check for while doing an RX interrupt.

RT_INT_RX_ERR

```
#define RT_INT_RX_ERR 0x0002
```

Rx error.

RT_INT_RX_OK

```
#define RT_INT_RX_OK 0x0001
```

Rx OK.

RT_INT_RXBUF_OVERFLOW

```
#define RT_INT_RXBUF_OVERFLOW 0x0010
```

Rx BUFFER overflow.

RT_INT_RXFIFO_OVERFLOW

```
#define RT_INT_RXFIFO_OVERFLOW 0x0040
```

Rx FIFO overflow.

RT_INT_RXFIFO_UNDERRUN

```
#define RT_INT_RXFIFO_UNDERRUN 0x0020
```

Packet underrun / link change.

RT_INT_TIMEOUT

```
#define RT_INT_TIMEOUT 0x4000
```

Set when TCTR reaches TimerInt value.

RT_INT_TX_ERR

```
#define RT_INT_TX_ERR 0x0008
```

Tx error.

RT_INT_TX_OK

```
#define RT_INT_TX_OK 0x0004
```

Tx OK.

7.106 RTL8139C MII (media independent interface) control bits

Macros

- #define RT_MII_RESET 0x8000

  Reset the MII chip.
- #define RT_MII_RES4000 0x4000

  Reserved.
- #define RT_MII_SPD_SET 0x2000

  1 for 100 0 for 10. Ignored if AN enabled.
- #define RT_MII_AN_ENABLE 0x1000

  Enable auto-negotiation.
- #define RT_MII_RES0800 0x0800

  Reserved.
- #define RT_MII_RES0400 0x0400

  Reserved.
- #define RT_MII_AN_START 0x0200

  Start auto-negotiation.
- #define RT_MII_DUPLEX 0x0100

  1 for full 0 for half. Ignored if AN enabled.

7.106.1 Detailed Description

7.106.2 Macro Definition Documentation

RT_MII_AN_ENABLE

```
#define RT_MII_AN_ENABLE 0x1000
```

Enable auto-negotiation.

RT_MII_AN_START

```
#define RT_MII_AN_START 0x0200
```

Start auto-negotiation.

RT_MII_DUPLEX

```
#define RT_MII_DUPLEX 0x0100
```

1 for full 0 for half. Ignored if AN enabled.

RT_MII_RES0400

```
#define RT_MII_RES0400 0x0400
```

Reserved.

RT_MII_RES0800

```
#define RT_MII_RES0800 0x0800
```

Reserved.

RT_MII_RES4000

```
#define RT_MII_RES4000 0x4000
```

Reserved.

RT_MII_RESET

```
#define RT_MII_RESET 0x8000
```

Reset the MII chip.

RT_MII_SPD_SET

```
#define RT_MII_SPD_SET 0x2000
```

1 for 100 0 for 10. Ignored if AN enabled.

7.107 RTL8139C MII (media independent interface) status bits

Macros

- #define RT_MII_LINK 0x0004
  Link is present.
- #define RT_MII_AN_CAPABLE 0x0008
  Can do auto negotiation.
- #define RT_MII_AN_COMPLETE 0x0020
  Auto-negotiation complete.
- #define RT_MII_10_HALF 0x0800
  Can do 10Mbit half duplex.
- #define RT_MII_10_FULL 0x1000
  Can do 10Mbit full.
- #define RT_MII_100_HALF 0x2000
  Can do 100Mbit half.
- #define RT_MII_100_FULL 0x4000
  Can do 100Mbit full.

7.107.1 Detailed Description

7.107.2 Macro Definition Documentation

RT_MII_100_FULL

```
#define RT_MII_100_FULL 0x4000
```

Can do 100Mbit full.

RT_MII_100_HALF

```
#define RT_MII_100_HALF 0x2000
```

Can do 100Mbit half.

RT_MII_10_FULL

```
#define RT_MII_10_FULL 0x1000
```

Can do 10Mbit full.

RT_MII_10_HALF

```
#define RT_MII_10_HALF 0x0800
```

Can do 10Mbit half duplex.

RT_MII_AN_CAPABLE

```
#define RT_MII_AN_CAPABLE 0x0008
```

Can do auto negotiation.

RT_MII_AN_COMPLETE

```
#define RT_MII_AN_COMPLETE 0x0020
```

Auto-negotiation complete.

RT_MII_LINK

```
#define RT_MII_LINK 0x0004
```

Link is present.

7.108 RTL8139C Register Definitions

Macros

- #define RT_IDR0 0x00

  MAC address 0 (RW 32bit, RO 16/8)
- #define RT_IDR1 0x01

  MAC address 1 (Read-only)
- #define RT_IDR2 0x02

  MAC address 2 (Read-only)
- #define RT_IDR3 0x03

  MAC address 3 (Read-only)
- #define RT_IDR4 0x04

  MAC address 4 (RW 32bit, RO 16/8)
- #define RT_IDR5 0x05

  MAC address 5 (Read-only)
- #define RT_RES06 0x06

  Reserved.
- #define RT_RES07 0x07

  Reserved.
- #define RT_MAR0 0x08

  Multicast filter 0 (RW 32bit, RO 16/8)
- #define RT_MAR1 0x09

  Multicast filter 1 (Read-only)
- #define RT_MAR2 0x0A

  Multicast filter 2 (Read-only)
- #define RT_MAR3 0x0B

  Multicast filter 3 (Read-only)
- #define RT_MAR4 0x0C

  Multicast filter 4 (RW 32bit, RO 16/8)
- #define RT_MAR5 0x0D

  Multicast filter 5 (Read-only)
- #define RT_MAR6 0x0E

  Multicast filter 6 (Read-only)
- #define RT_MAR7 0x0F

  Multicast filter 7 (Read-only)
- #define RT_TXSTATUS0 0x10

  Transmit status 0 (32bit only)
- #define RT_TXSTATUS1 0x14

  Transmit status 1 (32bit only)

- #define RT_TXSTATUS2 0x18

  Transmit status 2 (32bit only)
- #define RT_TXSTATUS3 0x1C

  Transmit status 3 (32bit only)
- #define RT_TXADDR0 0x20

  Tx descriptor 0 (32bit only)
- #define RT_TXADDR1 0x24

  Tx descriptor 1 (32bit only)
- #define RT_TXADDR2 0x28

  Tx descriptor 2 (32bit only)
- #define RT_TXADDR3 0x2C

  Tx descriptor 3 (32bit only)
- #define RT_RXBUF 0x30

  Receive buffer start address (32bit only)
- #define RT_RXEARLYCNT 0x34

  Early Rx byte count (RO 16bit)
- #define RT_RXEARLYSTATUS 0x36

  Early Rx status (RO)
- #define RT_CHIPCMD 0x37

  Command register.
- #define RT_RXBUFTAIL 0x38

  Current address of packet read (queue tail) (16bit only)
- #define RT_RXBUFHEAD 0x3A

  Current buffer address (queue head) (RO 16bit)
- #define RT_INTRMASK 0x3C

  Interrupt mask (16bit only)
- #define RT_INTRSTATUS 0x3E

  Interrupt status (16bit only)
- #define RT_TXCONFIG 0x40

  Tx config (32bit only)
- #define RT_RXCONFIG 0x44

  Rx config (32bit only)
- #define RT_TIMER 0x48

  A general purpose counter, any write clears (32bit only)
- #define RT_RXMISSED 0x4C

  24 bits valid, write clears (32bit only)
- #define RT_CFG9346 0x50

  93C46 command register
- #define RT_CONFIG0 0x51

  Configuration reg 0.
- #define RT_CONFIG1 0x52

  Configuration reg 1.
- #define RT_RES53 0x53

  Reserved.
- #define RT_TIMERINT 0x54

  Timer interrupt register (32bit only)
- #define RT_MEDIASTATUS 0x58

Media status register.

- #define RT_CONFIG3 0x59

  Config register 3.
- #define RT_CONFIG4 0x5A

  Config register 4.
- #define RT_RES5B 0x5B

  Reserved.
- #define RT_MULTIINTR 0x5C

  Multiple interrupt select (32bit only)
- #define RT_RERID 0x5E

  PCI Revision ID (10h) (Read-only)
- #define RT_RES5F 0x5F

  Reserved.
- #define RT_MII_TSAD 0x60

  Transmit status of all descriptors (RO 16bit)
- #define RT_MII_BMCR 0x62

  Basic Mode Control Register (16bit only)
- #define RT_MII_BMSR 0x64

  Basic Mode Status Register (RO 16bit)
- #define RT_AS_ADVERT 0x66

  Auto-negotiation advertisement reg (16bit only)
- #define RT_AS_LPAR 0x68

  Auto-negotiation link partner reg (RO 16bit)
- #define RT_AS_EXPANSION 0x6A

  Auto-negotiation expansion reg (RO 16bit)
- #define RT_CONFIG5 0xD8

  Config register 5.

7.108.1 Detailed Description

The default assumption is that these are all RW at any aligned size unless otherwise noted. ex (RW 32bit, RO 16/8) indicates read/write at 32bit and read-only at 16 or 8bits.

7.108.2 Macro Definition Documentation

RT_AS_ADVERT

```
#define RT_AS_ADVERT 0x66
```

Auto-negotiation advertisement reg (16bit only)

RT_AS_EXPANSION

```
#define RT_AS_EXPANSION 0x6A
```

Auto-negotiation expansion reg (RO 16bit)

RT_AS_LPAR

```
#define RT_AS_LPAR 0x68
```

Auto-negotiation link partner reg (RO 16bit)

RT_CFG9346

```
#define RT_CFG9346 0x50
```

93C46 command register

RT_CHIPCMD

```
#define RT_CHIPCMD 0x37
```

Command register.

RT_CONFIG0

```
#define RT_CONFIG0 0x51
```

Configuration reg 0.

RT_CONFIG1

```
#define RT_CONFIG1 0x52
```

Configuration reg 1.

RT_CONFIG3

```
#define RT_CONFIG3 0x59
```

Config register 3.

RT_CONFIG4

```
#define RT_CONFIG4 0x5A
```

Config register 4.

RT_CONFIG5

```
#define RT_CONFIG5 0xD8
```

Config register 5.

RT_IDR0

```
#define RT_IDR0 0x00
```

MAC address 0 (RW 32bit, RO 16/8)

RT_IDR1

```
#define RT_IDR1 0x01
```

MAC address 1 (Read-only)

RT_IDR2

```
#define RT_IDR2 0x02
```

MAC address 2 (Read-only)

RT_IDR3

```
#define RT_IDR3 0x03
```

MAC address 3 (Read-only)

RT_IDR4

```
#define RT_IDR4 0x04
```

MAC address 4 (RW 32bit, RO 16/8)

RT_IDR5

```
#define RT_IDR5 0x05
```

MAC address 5 (Read-only)

RT_INTRMASK

```
#define RT_INTRMASK 0x3C
```

Interrupt mask (16bit only)

RT_INTRSTATUS

```
#define RT_INTRSTATUS 0x3E
```

Interrupt status (16bit only)

RT_MAR0

```
#define RT_MAR0 0x08
```

Multicast filter 0 (RW 32bit, RO 16/8)

RT_MAR1

```
#define RT_MAR1 0x09
```

Multicast filter 1 (Read-only)

RT_MAR2

```
#define RT_MAR2 0x0A
```

Multicast filter 2 (Read-only)

RT_MAR3

```
#define RT_MAR3 0x0B
```

Multicast filter 3 (Read-only)

RT_MAR4

```
#define RT_MAR4 0x0C
```

Multicast filter 4 (RW 32bit, RO 16/8)

RT_MAR5

```
#define RT_MAR5 0x0D
```

Multicast filter 5 (Read-only)

RT_MAR6

```
#define RT_MAR6 0x0E
```

Multicast filter 6 (Read-only)

RT_MAR7

```
#define RT_MAR7 0x0F
```

Multicast filter 7 (Read-only)

RT_MEDIASTATUS

```
#define RT_MEDIASTATUS 0x58
```

Media status register.

RT_MII_BMCR

```
#define RT_MII_BMCR 0x62
```

Basic Mode Control Register (16bit only)

RT_MII_BMSR

```
#define RT_MII_BMSR 0x64
```

Basic Mode Status Register (RO 16bit)

RT_MII_TSAD

```
#define RT_MII_TSAD 0x60
```

Transmit status of all descriptors (RO 16bit)

RT_MULTIINTR

```
#define RT_MULTIINTR 0x5C
```

Multiple interrupt select (32bit only)

RT_RERID

```
#define RT_RERID 0x5E
```

PCI Revision ID (10h) (Read-only)

RT_RES06

```
#define RT_RES06 0x06
```

Reserved.

RT_RES07

```
#define RT_RES07 0x07
```

Reserved.

RT_RES53

```
#define RT_RES53 0x53
```

Reserved.

RT_RES5B

```
#define RT_RES5B 0x5B
```

Reserved.

RT_RES5F

```
#define RT_RES5F 0x5F
```

Reserved.

RT_RXBUF

```
#define RT_RXBUF 0x30
```

Receive buffer start address (32bit only)

RT_RXBUFHEAD

```
#define RT_RXBUFHEAD 0x3A
```

Current buffer address (queue head) (RO 16bit)

RT_RXBUFTAIL

```
#define RT_RXBUFTAIL 0x38
```

Current address of packet read (queue tail) (16bit only)

RT_RXCONFIG

```
#define RT_RXCONFIG 0x44
```

Rx config (32bit only)

RT_RXEARLYCNT

```
#define RT_RXEARLYCNT 0x34
```

Early Rx byte count (RO 16bit)

RT_RXEARLYSTATUS

```
#define RT_RXEARLYSTATUS 0x36
```

Early Rx status (RO)

RT_RXMISSED

```
#define RT_RXMISSED 0x4C
```

24 bits valid, write clears (32bit only)

RT_TIMER

```
#define RT_TIMER 0x48
```

A general purpose counter, any write clears (32bit only)

RT_TIMERINT

```
#define RT_TIMERINT 0x54
```

Timer interrupt register (32bit only)

RT_TXADDR0

```
#define RT_TXADDR0 0x20
```

Tx descriptor 0 (32bit only)

RT_TXADDR1

```
#define RT_TXADDR1 0x24
```

Tx descriptor 1 (32bit only)

RT_TXADDR2

```
#define RT_TXADDR2 0x28
```

Tx descriptor 2 (32bit only)

RT_TXADDR3

```
#define RT_TXADDR3 0x2C
```

Tx descriptor 3 (32bit only)

RT_TXCONFIG

```
#define RT_TXCONFIG 0x40
```

Tx config (32bit only)

RT_TXSTATUS0

```
#define RT_TXSTATUS0 0x10
```

Transmit status 0 (32bit only)

RT_TXSTATUS1

```
#define RT_TXSTATUS1 0x14
```

Transmit status 1 (32bit only)

RT_TXSTATUS2

```
#define RT_TXSTATUS2 0x18
```

Transmit status 2 (32bit only)

RT_TXSTATUS3

```
#define RT_TXSTATUS3 0x1C
```

Transmit status 3 (32bit only)

7.109 RTL8139C receive status bits

Macros

- #define RT_RX_MULTICAST 0x00008000
  Multicast packet.
- #define RT_RX_PAM 0x00004000
  Physical address matched.
- #define RT_RX_BROADCAST 0x00002000
  Broadcast address matched.
- #define RT_RX_BAD_SYMBOL 0x00000020
  Invalid symbol in 100TX packet.
- #define RT_RX_RUNT 0x00000010
  Packet size is <64 bytes.
- #define RT_RX_TOO_LONG 0x00000008
  Packet size is >4K bytes.
- #define RT_RX_CRC_ERR 0x00000004
  CRC error.
- #define RT_RX_FRAME_ALIGN 0x00000002
  Frame alignment error.
- #define RT_RX_STATUS_OK 0x00000001
  Status ok: a good packet was received.

7.109.1 Detailed Description

7.109.2 Macro Definition Documentation

RT_RX_BAD_SYMBOL

```
#define RT_RX_BAD_SYMBOL 0x00000020
```

Invalid symbol in 100TX packet.

RT_RX_BROADCAST

```
#define RT_RX_BROADCAST 0x00002000
```

Broadcast address matched.

RT_RX_CRC_ERR

```
#define RT_RX_CRC_ERR 0x00000004
```

CRC error.

RT_RX_FRAME_ALIGN

```
#define RT_RX_FRAME_ALIGN 0x00000002
```

Frame alignment error.

RT_RX_MULTICAST

```
#define RT_RX_MULTICAST 0x00008000
```

Multicast packet.

RT_RX_PAM

```
#define RT_RX_PAM 0x00004000
```

Physical address matched.

RT_RX_RUNT

```
#define RT_RX_RUNT 0x00000010
```

Packet size is <64 bytes.

RT_RX_STATUS_OK

```
#define RT_RX_STATUS_OK 0x00000001
```

Status ok: a good packet was received.

RT_RX_TOO_LONG

```
#define RT_RX_TOO_LONG 0x00000008
```

Packet size is >4K bytes.

7.110 RTL8139C transmit status bits

Macros

- #define RT_TX_CARRIER_LOST 0x80000000
  Carrier sense lost.
- #define RT_TX_ABORTED 0x40000000
  Transmission aborted.
- #define RT_TX_OUT_OF_WINDOW 0x20000000
  Out of window collision.
- #define RT_TX_STATUS_OK 0x00008000
  Status ok: a good packet was transmitted.
- #define RT_TX_UNDERRUN 0x00004000
  Transmit FIFO underrun.
- #define RT_TX_HOST_OWNS 0x00002000
  Set to 1 when DMA operation is completed.
- #define RT_TX_SIZE_MASK 0x00001fff
  Descriptor size mask.

7.110.1 Detailed Description

7.110.2 Macro Definition Documentation

RT_TX_ABORTED

```
#define RT_TX_ABORTED 0x40000000
```

Transmission aborted.

RT_TX_CARRIER_LOST

```
#define RT_TX_CARRIER_LOST 0x80000000
```

Carrier sense lost.

RT_TX_HOST_OWNS

```
#define RT_TX_HOST_OWNS 0x00002000
```

Set to 1 when DMA operation is completed.

RT_TX_OUT_OF_WINDOW

```
#define RT_TX_OUT_OF_WINDOW 0x20000000
```

Out of window collision.

RT_TX_SIZE_MASK

```
#define RT_TX_SIZE_MASK 0x00001fff
```

Descriptor size mask.

RT_TX_STATUS_OK

```
#define RT_TX_STATUS_OK 0x00008000
```

Status ok: a good packet was transmitted.

RT_TX_UNDERRUN

```
#define RT_TX_UNDERRUN 0x00004000
```

Transmit FIFO underrun.

7.111 Real-Time Clock

Real-Time Clock (RTC) Management.

Modules

- Registers

  RTC registers.

Files

- file rtc.h

  Low-level real-time clock functionality.

Functions

- time_t rtc_unix_secs (void)

  Get the current date/time.
- int rtc_set_unix_secs (time_t time)

  Get the current date/time.
- time_t rtc_boot_time (void)

  Get the time since the sytem was booted.

7.111.1 Detailed Description

Real-Time Clock (RTC) Management.

Provides an API for fetching and managing the date/time using the Dreamcast's real-time clock. All timestamps are in standard Unix format, with an epoch of January 1, 1970. Due to the fact that there is no time zone data on the RTC, all times are expected to be in the local time zone.

Note

The RTC that is used by the DC is located on the AICA rather than SH4, presumably for power-efficiency reasons. Because of this, accessing it requires a trip over the G2 BUS, which is notoriously slow.

For reading the current date/time, you should favor the standard C, C++, or POSIX functions, as they are platform-indpendent and are calculating current time based on a cached boot time plus a delta that is maintained by the timer subsystem, rather than actually having to requery the RTC over the G2 BUS, so they are faster.

Warning

Internally, the RTC's date/time is maintained using a 32-bit counter with an epoch of January 1, 1950 00:00. Because of this, the Dreamcast's Y2K and the last timestamp it can represent before rolling over is February 06 2086 06:28:15.

7.111.2 Function Documentation

rtc_boot_time()

```
time_t rtc_boot_time (
            void  )
```

Get the time since the sytem was booted.

This function retrieves the cached RTC value from when KallistiOS was started. As with rtc_unix_secs(), this is a UNIX-style timestamp in local time.

Returns

The boot time as a UNIX-style timestamp.

See also

rtc_unix_secs()

rtc_set_unix_secs()

```
int rtc_set_unix_secs (
            time_t time )
```

Get the current date/time.

This function sets the current RTC value as a standard UNIX timestamp (with an epoch of January 1, 1970 00:00). This is assumed to be in the timezone of the user (as the RTC does not support timezones).

Parameters

time	Unix timestamp to set the current time to

Returns

0 for success or -1 for failure

See also

rtc_unix_secs()

rtc_unix_secs()

```
time_t rtc_unix_secs (
            void  )
```

Get the current date/time.

This function retrieves the current RTC value as a standard UNIX timestamp (with an epoch of January 1, 1970 00:00). This is assumed to be in the timezone of the user (as the RTC does not support timezones).

Returns

The current UNIX-style timestamp (local time).

See also

rtc_set_unix_secs(), rtc_boot_time()

7.111.3 Registers

RTC registers.

Macros

- #define RTC_TIMESTAMP_HIGH_ADDR 0xa0710000

  High 16-bit timestamp value.
- #define RTC_TIMESTAMP_LOW_ADDR 0xa0710004

  Low 16-bit timestamp value.
- #define RTC_CTRL_ADDR 0xa0710008

  Timestamp control register.

7.111.3.1 Detailed Description

RTC registers.

All registers are located on the G2 BUS and must be read and written to as full 32-byte values.

7.111.3.2 Macro Definition Documentation

RTC_CTRL_ADDR

```
#define RTC_CTRL_ADDR 0xa0710008
```

Timestamp control register.

All fields are reserved except for RTC_CTRL_WRITE_EN, which is write-only.

RTC_TIMESTAMP_HIGH_ADDR

```
#define RTC_TIMESTAMP_HIGH_ADDR 0xa0710000
```

High 16-bit timestamp value.

32-bit register containing the upper 16-bits of the 32-bit timestamp in seconds. Only the lower 16-bits are valid.

Note

Writing to this register will lock the timestamp registers.

RTC_TIMESTAMP_LOW_ADDR

```
#define RTC_TIMESTAMP_LOW_ADDR 0xa0710004
```

Low 16-bit timestamp value.

32-bit register containing the lower 16-bits of the 32-bit timestamp in seconds. Only the lower 16-bits are valid.

7.112 Region codes

Macros

- #define HW_REGION_UNKNOWN 0x0
  Unknown region.
- #define HW_REGION_ASIA 0x1
  Japan/Asia (NTSC)
- #define HW_REGION_US 0x4
  North America.
- #define HW_REGION_EUROPE 0xC
  Europe (PAL)

7.112.1 Detailed Description

These are the various region codes that can be returned by the hardware_sys_mode() function. Note that a retail Dreamcast will always return 0 for the region code. You must read the region of a retail device from the flashrom.

See also

Region settings possible in the system

flashrom_get_region()

7.112.2 Macro Definition Documentation

HW_REGION_ASIA

```
#define HW_REGION_ASIA 0x1
```

Japan/Asia (NTSC)

HW_REGION_EUROPE

```
#define HW_REGION_EUROPE 0xC
```

Europe (PAL)

HW_REGION_UNKNOWN

```
#define HW_REGION_UNKNOWN 0x0
```

Unknown region.

HW_REGION_US

```
#define HW_REGION_US 0x4
```

North America.

7.113 Region settings possible in the system

Macros

- #define FLASHROM_REGION_UNKNOWN 0
  Unknown region.
- #define FLASHROM_REGION_JAPAN 1
  Japanese region.
- #define FLASHROM_REGION_US 2
  US/Canada region.
- #define FLASHROM_REGION_EUROPE 3
  European region.

7.113.1 Detailed Description

One of these values should be returned by flashrom_get_region().

7.113.2 Macro Definition Documentation

FLASHROM_REGION_EUROPE

```
#define FLASHROM_REGION_EUROPE 3
```

European region.

FLASHROM_REGION_JAPAN

```
#define FLASHROM_REGION_JAPAN 1
```

Japanese region.

FLASHROM_REGION_UNKNOWN

```
#define FLASHROM_REGION_UNKNOWN 0
```

Unknown region.

FLASHROM_REGION_US

```
#define FLASHROM_REGION_US 2
```

US/Canada region.

7.114 Return values from Maple functions

Macros

- #define MAPLE_EOK 0

  No error.
- #define MAPLE_EFAIL -1

  Command failed.
- #define MAPLE_EAGAIN -2

  Try again later.
- #define MAPLE_EINVALID -3

  Invalid command.
- #define MAPLE_ENOTSUPP -4

  Command not suppoted by device.
- #define MAPLE_ETIMEOUT -5

  Command timed out.

7.114.1 Detailed Description

7.114.2 Macro Definition Documentation

MAPLE_EAGAIN

```
#define MAPLE_EAGAIN -2
```

Try again later.

MAPLE_EFAIL

```
#define MAPLE_EFAIL -1
```

Command failed.

MAPLE_EINVALID

```
#define MAPLE_EINVALID -3
```

Invalid command.

MAPLE_ENOTSUPP

```
#define MAPLE_ENOTSUPP -4
```

Command not suppoted by device.

MAPLE_EOK

```
#define MAPLE_EOK 0
```

No error.

MAPLE_ETIMEOUT

```
#define MAPLE_ETIMEOUT -5
```

Command timed out.

7.115 Return values from bba_tx().

Macros

- #define BBA_TX_OK 0

  Transmit success.
- #define BBA_TX_ERROR -1

  Transmit error.
- #define BBA_TX_AGAIN -2

  Retry transmit again.

7.115.1 Detailed Description

7.115.2 Macro Definition Documentation

BBA_TX_AGAIN

```
#define BBA_TX_AGAIN -2
```

Retry transmit again.

BBA_TX_ERROR

```
#define BBA_TX_ERROR -1
```

Transmit error.

BBA_TX_OK

```
#define BBA_TX_OK 0
```

Transmit success.

7.116 SH4 exception codes

Modules

- Completion type
- Interrupt (completion type)
- Re-Execution type
- Reset type

Macros

- #define EXC_DOUBLE_FAULT 0x0ff0

  Double fault.
- #define EXC_UNHANDLED_EXC 0x0fe0

  Unhandled exception.

7.116.1 Detailed Description

These are all of the exceptions that can be raised on the SH4, and their codes. They're divided into several logical groups.

7.116.2 Macro Definition Documentation

EXC_DOUBLE_FAULT

```
#define EXC_DOUBLE_FAULT 0x0ff0
```

Double fault.

This exception is completely done in software (not represented on the CPU at all). Its used for when an exception occurs during an IRQ service routine.

EXC_UNHANDLED_EXC

```
#define EXC_UNHANDLED_EXC 0x0fe0
```

Unhandled exception.

This exception is a software-generated exception for a generic unhandled exception.

7.116.3 Completion type

Macros

- #define EXC_TRAPA 0x0160

  Unconditional trap (trapa)
- #define EXC_USER_BREAK_POST 0x01e0

  User break after instruction.

7.116.3.1 Detailed Description

These exceptions are actually handled in-between instructions, allowing the instruction that causes them to finish completely. The saved PC thus is the value of the next instruction.

7.116.3.2 Macro Definition Documentation

EXC_TRAPA

```
#define EXC_TRAPA 0x0160
```

Unconditional trap (trapa)

EXC_USER_BREAK_POST

```
#define EXC_USER_BREAK_POST 0x01e0
```

User break after instruction.

7.116.4 Interrupt (completion type)

Macros

- #define EXC_NMI 0x01c0
  Nonmaskable interrupt.
- #define EXC_IRQ0 0x0200
  External IRQ request (level 0)
- #define EXC_IRQ1 0x0220
  External IRQ request (level 1)
- #define EXC_IRQ2 0x0240
  External IRQ request (level 2)
- #define EXC_IRQ3 0x0260
  External IRQ request (level 3)
- #define EXC_IRQ4 0x0280
  External IRQ request (level 4)
- #define EXC_IRQ5 0x02a0
  External IRQ request (level 5)
- #define EXC_IRQ6 0x02c0
  External IRQ request (level 6)
- #define EXC_IRQ7 0x02e0
  External IRQ request (level 7)
- #define EXC_IRQ8 0x0300
  External IRQ request (level 8)
- #define EXC_IRQ9 0x0320
  External IRQ request (level 9)
- #define EXC_IRQA 0x0340
  External IRQ request (level 10)
- #define EXC_IRQB 0x0360
  External IRQ request (level 11)
- #define EXC_IRQC 0x0380

External IRQ request (level 12)

- #define EXC_IRQD 0x03a0

  External IRQ request (level 13)

- #define EXC_IRQE 0x03c0

  External IRQ request (level 14)

- #define EXC_TMU0_TUNI0 0x0400

  TMU0 underflow.

- #define EXC_TMU1_TUNI1 0x0420

  TMU1 underflow.

- #define EXC_TMU2_TUNI2 0x0440

  TMU2 underflow.

- #define EXC_TMU2_TICPI2 0x0460

  TMU2 input capture.

- #define EXC_RTC_ATI 0x0480

  RTC alarm interrupt.

- #define EXC_RTC_PRI 0x04a0

  RTC periodic interrupt.

- #define EXC_RTC_CUI 0x04c0

  RTC carry interrupt.

- #define EXC_SCI_ERI 0x04e0

  SCI Error receive.

- #define EXC_SCI_RXI 0x0500

  SCI Receive ready.

- #define EXC_SCI_TXI 0x0520

  SCI Transmit ready.

- #define EXC_SCI_TEI 0x0540

  SCI Transmit error.

- #define EXC_WDT_ITI 0x0560

  Watchdog timer.

- #define EXC_REF_RCMI 0x0580

  Memory refresh compare-match interrupt.

- #define EXC_REF_ROVI 0x05a0

  Memory refresh counter overflow interrupt.

- #define EXC_UDI 0x0600

  Hitachi UDI.

- #define EXC_GPIO_GPIOI 0x0620

  I/O port interrupt.

- #define EXC_DMAC_DMTE0 0x0640

  DMAC transfer end (channel 0)

- #define EXC_DMAC_DMTE1 0x0660

  DMAC transfer end (channel 1)

- #define EXC_DMAC_DMTE2 0x0680

  DMAC transfer end (channel 2)

- #define EXC_DMAC_DMTE3 0x06a0

  DMAC transfer end (channel 3)

- #define EXC_DMA_DMAE 0x06c0

  DMAC address error.

- #define EXC_SCIF_ERI 0x0700

  SCIF Error receive.
- #define EXC_SCIF_RXI 0x0720

  SCIF Receive ready.
- #define EXC_SCIF_BRI 0x0740

  SCIF break.
- #define EXC_SCIF_TXI 0x0760

  SCIF Transmit ready.

7.116.4.1 Detailed Description

These exceptions are caused by interrupt requests. These generally are from peripheral devices, but NMIs, timer interrupts, and DMAC interrupts are also included here.

Note

Not all of these have any meaning on the Dreamcast. Those that have no meaning are only included for completeness.

7.116.4.2 Macro Definition Documentation

EXC_DMA_DMAE

```
#define EXC_DMA_DMAE 0x06c0
```

DMAC address error.

EXC_DMAC_DMTE0

```
#define EXC_DMAC_DMTE0 0x0640
```

DMAC transfer end (channel 0)

EXC_DMAC_DMTE1

```
#define EXC_DMAC_DMTE1 0x0660
```

DMAC transfer end (channel 1)

EXC_DMAC_DMTE2

```
#define EXC_DMAC_DMTE2 0x0680
```

DMAC transfer end (channel 2)

EXC_DMAC_DMTE3

```
#define EXC_DMAC_DMTE3 0x06a0
```

DMAC transfer end (channel 3)

EXC_GPIO_GPIOI

```
#define EXC_GPIO_GPIOI 0x0620
```

I/O port interrupt.

EXC_IRQ0

```
#define EXC_IRQ0 0x0200
```

External IRQ request (level 0)

EXC_IRQ1

```
#define EXC_IRQ1 0x0220
```

External IRQ request (level 1)

EXC_IRQ2

```
#define EXC_IRQ2 0x0240
```

External IRQ request (level 2)

EXC_IRQ3

```
#define EXC_IRQ3 0x0260
```

External IRQ request (level 3)

EXC_IRQ4

```
#define EXC_IRQ4 0x0280
```

External IRQ request (level 4)

EXC_IRQ5

```
#define EXC_IRQ5 0x02a0
```

External IRQ request (level 5)

EXC_IRQ6

```
#define EXC_IRQ6 0x02c0
```

External IRQ request (level 6)

EXC_IRQ7

```
#define EXC_IRQ7 0x02e0
```

External IRQ request (level 7)

EXC_IRQ8

```
#define EXC_IRQ8 0x0300
```

External IRQ request (level 8)

EXC_IRQ9

```
#define EXC_IRQ9 0x0320
```

External IRQ request (level 9)

EXC_IRQA

```
#define EXC_IRQA 0x0340
```

External IRQ request (level 10)

EXC_IRQB

```
#define EXC_IRQB 0x0360
```

External IRQ request (level 11)

EXC_IRQC

```
#define EXC_IRQC 0x0380
```

External IRQ request (level 12)

EXC_IRQD

```
#define EXC_IRQD 0x03a0
```

External IRQ request (level 13)

EXC_IRQE

```
#define EXC_IRQE 0x03c0
```

External IRQ request (level 14)

EXC_NMI

```
#define EXC_NMI 0x01c0
```

Nonmaskable interrupt.

EXC_REF_RCMI

```
#define EXC_REF_RCMI 0x0580
```

Memory refresh compare-match interrupt.

EXC_REF_ROVI

```
#define EXC_REF_ROVI 0x05a0
```

Memory refresh counter overflow interrupt.

EXC_RTC_ATI

```
#define EXC_RTC_ATI 0x0480
```

RTC alarm interrupt.

EXC_RTC_CUI

```
#define EXC_RTC_CUI 0x04c0
```

RTC carry interrupt.

EXC_RTC_PRI

```
#define EXC_RTC_PRI 0x04a0
```

RTC periodic interrupt.

EXC_SCI_ERI

```
#define EXC_SCI_ERI 0x04e0
```

SCI Error receive.

EXC_SCI_RXI

```
#define EXC_SCI_RXI 0x0500
```

SCI Receive ready.

EXC_SCI_TEI

```
#define EXC_SCI_TEI 0x0540
```

SCI Transmit error.

EXC_SCI_TXI

```
#define EXC_SCI_TXI 0x0520
```

SCI Transmit ready.

EXC_SCIF_BRI

```
#define EXC_SCIF_BRI 0x0740
```

SCIF break.

EXC_SCIF_ERI

```
#define EXC_SCIF_ERI 0x0700
```

SCIF Error receive.

EXC_SCIF_RXI

```
#define EXC_SCIF_RXI 0x0720
```

SCIF Receive ready.

EXC_SCIF_TXI

```
#define EXC_SCIF_TXI 0x0760
```

SCIF Transmit ready.

EXC_TMU0_TUNI0

```
#define EXC_TMU0_TUNI0 0x0400
```

TMU0 underflow.

EXC_TMU1_TUNI1

```
#define EXC_TMU1_TUNI1 0x0420
```

TMU1 underflow.

EXC_TMU2_TICPI2

```
#define EXC_TMU2_TICPI2 0x0460
```

TMU2 input capture.

EXC_TMU2_TUNI2

```
#define EXC_TMU2_TUNI2 0x0440
```

TMU2 underflow.

EXC_UDI

```
#define EXC_UDI 0x0600
```

Hitachi UDI.

EXC_WDT_ITI

```
#define EXC_WDT_ITI 0x0560
```

Watchdog timer.

7.116.5 Re-Execution type

Macros

- #define EXC_USER_BREAK_PRE 0x01e0
  User break before instruction.
- #define EXC_INSTR_ADDRESS 0x00e0
  Instruction address.
- #define EXC_ITLB_MISS 0x0040
  Instruction TLB miss.
- #define EXC_ITLB_PV 0x00a0
  Instruction TLB protection violation.
- #define EXC_ILLEGAL_INSTR 0x0180
  Illegal instruction.
- #define EXC_SLOT_ILLEGAL_INSTR 0x01a0
  Slot illegal instruction.
- #define EXC_GENERAL_FPU 0x0800
  General FPU exception.
- #define EXC_SLOT_FPU 0x0820
  Slot FPU exception.
- #define EXC_DATA_ADDRESS_READ 0x00e0
  Data address (read)
- #define EXC_DATA_ADDRESS_WRITE 0x0100
  Data address (write)
- #define EXC_DTLB_MISS_READ 0x0040
  Data TLB miss (read)
- #define EXC_DTLB_MISS_WRITE 0x0060
  Data TLB miss (write)
- #define EXC_DTLB_PV_READ 0x00a0
  Data TLB protection violation (read)
- #define EXC_DTLB_PV_WRITE 0x00c0
  Data TLB protection violation (write)
- #define EXC_FPU 0x0120
  FPU exception.
- #define EXC_INITIAL_PAGE_WRITE 0x0080
  Initial page write exception.

7.116.5.1 Detailed Description

These exceptions will stop the currently processing instruction, and transition into exception processing. After handling the exception (assuming that it can be handled by the code), the offending instruction will be re-executed from the start.

7.116.5.2 Macro Definition Documentation

EXC_DATA_ADDRESS_READ

```
#define EXC_DATA_ADDRESS_READ 0x00e0
```

Data address (read)

EXC_DATA_ADDRESS_WRITE

```
#define EXC_DATA_ADDRESS_WRITE 0x0100
```

Data address (write)

EXC_DTLB_MISS_READ

```
#define EXC_DTLB_MISS_READ 0x0040
```

Data TLB miss (read)

EXC_DTLB_MISS_WRITE

```
#define EXC_DTLB_MISS_WRITE 0x0060
```

Data TLB miss (write)

EXC_DTLB_PV_READ

```
#define EXC_DTLB_PV_READ 0x00a0
```

Data TLB protection violation (read)

EXC_DTLB_PV_WRITE

```
#define EXC_DTLB_PV_WRITE 0x00c0
```

Data TLB protection violation (write)

EXC_FPU

```
#define EXC_FPU 0x0120
```

FPU exception.

EXC_GENERAL_FPU

```
#define EXC_GENERAL_FPU 0x0800
```

General FPU exception.

EXC_ILLEGAL_INSTR

```
#define EXC_ILLEGAL_INSTR 0x0180
```

Illegal instruction.

EXC_INITIAL_PAGE_WRITE

```
#define EXC_INITIAL_PAGE_WRITE 0x0080
```

Initial page write exception.

EXC_INSTR_ADDRESS

```
#define EXC_INSTR_ADDRESS 0x00e0
```

Instruction address.

EXC_ITLB_MISS

```
#define EXC_ITLB_MISS 0x0040
```

Instruction TLB miss.

EXC_ITLB_PV

```
#define EXC_ITLB_PV 0x00a0
```

Instruction TLB protection violation.

EXC_SLOT_FPU

```
#define EXC_SLOT_FPU 0x0820
```

Slot FPU exception.

EXC_SLOT_ILLEGAL_INSTR

```
#define EXC_SLOT_ILLEGAL_INSTR 0x01a0
```

Slot illegal instruction.

EXC_USER_BREAK_PRE

```
#define EXC_USER_BREAK_PRE 0x01e0
```

User break before instruction.

7.116.6 Reset type

Macros

- #define EXC_RESET_POWERON 0x0000
  Power-on reset.
- #define EXC_RESET_MANUAL 0x0020
  Manual reset.
- #define EXC_RESET_UDI 0x0000
  Hitachi UDI reset.
- #define EXC_ITLB_MULTIPLE 0x0140
  Instruction TLB multiple hit.
- #define EXC_DTLB_MULTIPLE 0x0140
  Data TLB multiple hit.

7.116.6.1 Detailed Description

These are exceptions that essentially cause a reset of the system. They cannot actually be caught by normal means. They will all automatically cause a branch to address 0xA0000000. These are pretty much fatal.

7.116.6.2 Macro Definition Documentation

EXC_DTLB_MULTIPLE

```
#define EXC_DTLB_MULTIPLE 0x0140
```

Data TLB multiple hit.

EXC_ITLB_MULTIPLE

```
#define EXC_ITLB_MULTIPLE 0x0140
```

Instruction TLB multiple hit.

EXC_RESET_MANUAL

```
#define EXC_RESET_MANUAL 0x0020
```

Manual reset.

EXC_RESET_POWERON

```
#define EXC_RESET_POWERON 0x0000
```

Power-on reset.

EXC_RESET_UDI

```
#define EXC_RESET_UDI 0x0000
```

Hitachi UDI reset.

7.117 Section header flags

Macros

- #define SHF_WRITE 1
  Writable data.
- #define SHF_ALLOC 2
  Resident.
- #define SHF_EXECINSTR 4
  Executable instructions.
- #define SHF_MASKPROC 0xf0000000
  Processor specific mask.

7.117.1 Detailed Description

These are the flags that can be set on a section header. These are related to whether the section should reside in memory and permissions on it.

7.117.2 Macro Definition Documentation

SHF_ALLOC

```
#define SHF_ALLOC 2
```

Resident.

SHF_EXECINSTR

```
#define SHF_EXECINSTR 4
```

Executable instructions.

SHF_MASKPROC

```
#define SHF_MASKPROC 0xf0000000
```

Processor specific mask.

SHF_WRITE

```
#define SHF_WRITE 1
```

Writable data.

7.118 Section header types

Macros

- #define SHT_NULL 0
  Inactive section.
- #define SHT_PROGBITS 1
  Program code/data.
- #define SHT_SYMTAB 2
  Full symbol table.
- #define SHT_STRTAB 3
  String table.
- #define SHT_RELA 4
  Relocation table, with addends.
- #define SHT_HASH 5
  Symbol hash table.
- #define SHT_DYNAMIC 6
  Dynamic linking info.

- #define SHT_NOTE 7

  Notes section.
- #define SHT_NOBITS 8

  A section that occupies no space in the file.
- #define SHT_REL 9

  Relocation table, no addends.
- #define SHT_SHLIB 10

  Reserved.
- #define SHT_DYNSYM 11

  Dynamic-only sym tab.
- #define SHT_LOPROC 0x70000000

  Start of processor specific types.
- #define SHT_HIPROC 0x7fffffff

  End of processor specific types.
- #define SHT_LOUSER 0x80000000

  Start of program specific types.
- #define SHT_HIUSER 0xffffffff

  End of program specific types.

7.118.1 Detailed Description

These are the various types of section headers that can exist in an ELF file.

7.118.2 Macro Definition Documentation

SHT_DYNAMIC

```
#define SHT_DYNAMIC 6
```

Dynamic linking info.

SHT_DYNSYM

```
#define SHT_DYNSYM 11
```

Dynamic-only sym tab.

SHT_HASH

```
#define SHT_HASH 5
```

Symbol hash table.

SHT_HIPROC

```
#define SHT_HIPROC 0x7fffffff
```

End of processor specific types.

SHT_HIUSER

```
#define SHT_HIUSER 0xffffffff
```

End of program specific types.

SHT_LOPROC

```
#define SHT_LOPROC 0x70000000
```

Start of processor specific types.

SHT_LOUSER

```
#define SHT_LOUSER 0x80000000
```

Start of program specific types.

SHT_NOBITS

```
#define SHT_NOBITS 8
```

A section that occupies no space in the file.

SHT_NOTE

```
#define SHT_NOTE 7
```

Notes section.

SHT_NULL

```
#define SHT_NULL 0
```

Inactive section.

SHT_PROGBITS

```
#define SHT_PROGBITS 1
```

Program code/data.

SHT_REL

```
#define SHT_REL 9
```

Relocation table, no addends.

SHT_RELA

```
#define SHT_RELA 4
```

Relocation table, with addends.

SHT_SHLIB

```
#define SHT_SHLIB 10
```

Reserved.

SHT_STRTAB

```
#define SHT_STRTAB 3
```

String table.

SHT_SYMTAB

```
#define SHT_SYMTAB 2
```

Full symbol table.

7.119 Seek modes

Macros

- #define SEEK_SET 0

  Set position to offset.
- #define SEEK_CUR 1

  Seek from current position.
- #define SEEK_END 2

  Seek from end of file.

7.119.1 Detailed Description

These are the values you can pass for the whence parameter to fs_seek().

7.119.2 Macro Definition Documentation

SEEK_CUR

```
#define SEEK_CUR 1
```

Seek from current position.

SEEK_END

```
#define SEEK_END 2
```

Seek from end of file.

SEEK_SET

```
#define SEEK_SET 0
```

Set position to offset.

7.120 Socket flags

Macros

- #define FS_SOCKET_NONBLOCK 0x00000001 /** \brief Non-blocking operations */
- #define FS_SOCKET_V6ONLY 0x00000002 /** \brief IPv6 Only */
- #define FS_SOCKET_GEN_MAX 0x00008000 /** \brief Maximum generic flag */
- #define FS_SOCKET_FAM_MAX 0x00800000 /** \brief Maximum family flag */

7.120.1 Detailed Description

These are the available flags defined for sockets.

Every flag after FS_SOCKET_FAM_MAX is for internal-use only, and should never be passed into any functions.

7.120.2 Macro Definition Documentation

FS_SOCKET_FAM_MAX

```
#define FS_SOCKET_FAM_MAX 0x00800000 /** \brief Maximum family flag */
```

FS_SOCKET_GEN_MAX

```
#define FS_SOCKET_GEN_MAX 0x00008000 /** \brief Maximum generic flag */
```

FS_SOCKET_NONBLOCK

```
#define FS_SOCKET_NONBLOCK 0x00000001 /** \brief Non-blocking operations */
```

FS_SOCKET_V6ONLY

```
#define FS_SOCKET_V6ONLY 0x00000002 /** \brief IPv6 Only */
```

7.121 Socket message flags

Macros

- #define MSG_CTRUNC 0x01
  Control data truncated (U)
- #define MSG_DONTROUTE 0x02
  Send without routing (U)
- #define MSG_EOR 0x04
  Terminate a record (U)
- #define MSG_OOB 0x08
  Out-of-band data (U)
- #define MSG_PEEK 0x10
  Leave received data in queue.
- #define MSG_TRUNC 0x20
  Normal data truncated (U)
- #define MSG_WAITALL 0x40
  Attempt to fill read buffer.
- #define MSG_DONTWAIT 0x80
  Make this call non-blocking (non-standard)

7.121.1 Detailed Description

The following flags can be used with the recv(), recvfrom(), send(), and sendto() functions as the flags parameter.

Note that not all of these are currently supported, but they are listed for completeness. Those that are unsupported have (U) at the end of their description. Also, for the time being, the supported flags are only supported for TCP.

7.121.2 Macro Definition Documentation

MSG_CTRUNC

```
#define MSG_CTRUNC 0x01
```

Control data truncated (U)

MSG_DONTROUTE

```
#define MSG_DONTROUTE 0x02
```

Send without routing (U)

MSG_DONTWAIT

```
#define MSG_DONTWAIT 0x80
```

Make this call non-blocking (non-standard)

MSG_EOR

```
#define MSG_EOR 0x04
```

Terminate a record (U)

MSG_OOB

```
#define MSG_OOB 0x08
```

Out-of-band data (U)

MSG_PEEK

```
#define MSG_PEEK 0x10
```

Leave received data in queue.

MSG_TRUNC

```
#define MSG_TRUNC 0x20
```

Normal data truncated (U)

MSG_WAITALL

```
#define MSG_WAITALL 0x40
```

Attempt to fill read buffer.

7.122 Socket-level options

Macros

- #define SO_ACCEPTCONN 1

  Socket is accepting connections (get)
- #define SO_BROADCAST 2

  Support broadcasting (get/set)
- #define SO_DEBUG 3

  Record debugging info (get/set)
- #define SO_DONTROUTE 4

  Do not route packets (get/set)
- #define SO_ERROR 5

  Retrieve error status (get)
- #define SO_KEEPALIVE 6

  Send keepalive messages (get/set)
- #define SO_LINGER 7

  Socket lingers on close (get/set)
- #define SO_OOBINLINE 8

  OOB data is inline (get/set)
- #define SO_RCVBUF 9

  Receive buffer size (get/set)
- #define SO_RCVLOWAT 10

  Receive low-water mark (get/set)
- #define SO_RCVTIMEO 11

  Receive timeout value (get/set)
- #define SO_REUSEADDR 12

  Reuse local addresses (get/set)
- #define SO_SNDBUF 13

  Send buffer size (get/set)
- #define SO_SNDLOWAT 14

  Send low-water mark (get/set)
- #define SO_SNDTIMEO 15

  Send timeout value (get/set)
- #define SO_TYPE 16

  Socket type (get)

7.122.1 Detailed Description

These are the various socket-level options that can be accessed with the setsockopt() and getsockopt() functions for the SOL_SOCKET level value.

Not all of these are currently supported, but they are listed for completeness.

See also

IPv6 protocol level options

IPv4 protocol level options

UDP protocol level options

7.122.2 Macro Definition Documentation

SO_ACCEPTCONN

```
#define SO_ACCEPTCONN 1
```

Socket is accepting connections (get)

SO_BROADCAST

```
#define SO_BROADCAST 2
```

Support broadcasting (get/set)

SO_DEBUG

```
#define SO_DEBUG 3
```

Record debugging info (get/set)

SO_DONTROUTE

```
#define SO_DONTROUTE 4
```

Do not route packets (get/set)

SO_ERROR

```
#define SO_ERROR 5
```

Retrieve error status (get)

SO_KEEPALIVE

```
#define SO_KEEPALIVE 6
```

Send keepalive messages (get/set)

SO_LINGER

```
#define SO_LINGER 7
```

Socket lingers on close (get/set)

SO_OOBINLINE

```
#define SO_OOBINLINE 8
```

OOB data is inline (get/set)

SO_RCVBUF

```
#define SO_RCVBUF 9
```

Receive buffer size (get/set)

SO_RCVLOWAT

```
#define SO_RCVLOWAT 10
```

Receive low-water mark (get/set)

SO_RCVTIMEO

```
#define SO_RCVTIMEO 11
```

Receive timeout value (get/set)

SO_REUSEADDR

```
#define SO_REUSEADDR 12
```

Reuse local addresses (get/set)

SO_SNDBUF

```
#define SO_SNDBUF 13
```

Send buffer size (get/set)

SO_SNDLOWAT

```
#define SO_SNDLOWAT 14
```

Send low-water mark (get/set)

SO_SNDTIMEO

```
#define SO_SNDTIMEO 15
```

Send timeout value (get/set)

SO_TYPE

```
#define SO_TYPE 16
```

Socket type (get)

7.123 Special section indeces

Macros

- #define SHN_UNDEF 0

  Undefined, missing, irrelevant.
- #define SHN_ABS 0xfff1

  Absolute values.

7.123.1 Detailed Description

These are the indices to be used in special situations in the section array.

7.123.2 Macro Definition Documentation

SHN_ABS

```
#define SHN_ABS 0xfff1
```

Absolute values.

SHN_UNDEF

```
#define SHN_UNDEF 0
```

Undefined, missing, irrelevant.

7.124 States each key can be in.

Macros

- #define KEY_STATE_NONE 0
- #define KEY_STATE_WAS_PRESSED 1
- #define KEY_STATE_PRESSED 2

7.124.1 Detailed Description

These are the different 'states' each key can be in. They are stored in kbd_state_t->matrix, and manipulated/checked by kbd_check_poll.

none-> pressed or none was pressed-> pressed or none pressed-> was_pressed

7.124.2 Macro Definition Documentation

KEY_STATE_NONE

```
#define KEY_STATE_NONE 0
```

KEY_STATE_PRESSED

```
#define KEY_STATE_PRESSED 2
```

KEY_STATE_WAS_PRESSED

```
#define KEY_STATE_WAS_PRESSED 1
```

7.125 States that frames can be in

Macros

- #define MAPLE_FRAME_VACANT 0

  Ready to be used.
- #define MAPLE_FRAME_UNSENT 1

  Ready to be sent.
- #define MAPLE_FRAME_SENT 2

  Frame has been sent, but no response yet.
- #define MAPLE_FRAME_RESPONDED 3

  Frame has a response.

7.125.1 Detailed Description

7.125.2 Macro Definition Documentation

MAPLE_FRAME_RESPONDED

```
#define MAPLE_FRAME_RESPONDED 3
```

Frame has a response.

MAPLE_FRAME_SENT

```
#define MAPLE_FRAME_SENT 2
```

Frame has been sent, but no response yet.

MAPLE_FRAME_UNSENT

```
#define MAPLE_FRAME_UNSENT 1
```

Ready to be sent.

MAPLE_FRAME_VACANT

```
#define MAPLE_FRAME_VACANT 0
```

Ready to be used.

7.126 Store Queues

SH4 CPU Peripheral for burst memory transactions.

Files

- file sq.h

  Functions to access the SH4 Store Queues.

Macros

- #define QACR0 (∗(volatile uint32_t ∗)(void ∗)0xff000038)

  Store Queue 0 access register.
- #define QACR1 (∗(volatile uint32_t ∗)(void ∗)0xff00003c)

  Store Queue 1 access register <>
- #define SET_QACR_REGS(dest0, dest1)

  Set Store Queue QACR∗ registers.
- #define SQ_MASK_DEST_ADDR(dest) (MEM_AREA_SQ_BASE | ((uintptr_t)(dest) & 0x03ffffe0))

  Mask dest to Store Queue area as address.
- #define SQ_MASK_DEST(dest) ((uint32_t ∗)(void ∗) SQ_MASK_DEST_ADDR(dest))

  Mask dest to Store Queue area as pointer.

Functions

- void sq_lock (void)

  Lock Store Queues.
- void sq_unlock (void)

  Unlock Store Queues.
- void ∗ sq_cpy (void ∗dest, const void ∗src, size_t n)

  Copy a block of memory.
- void ∗ sq_set (void ∗dest, uint32_t c, size_t n)

  Set a block of memory to an 8-bit value.
- void ∗ sq_set16 (void ∗dest, uint32_t c, size_t n)

  Set a block of memory to a 16-bit value.
- void ∗ sq_set32 (void ∗dest, uint32_t c, size_t n)

  Set a block of memory to a 32-bit value.
- void sq_clr (void ∗dest, size_t n)

  Clear a block of memory.
- void ∗ sq_cpy_pvr (void ∗dest, const void ∗src, size_t n)

  Copy a block of memory to VRAM.
- void ∗ sq_set_pvr (void ∗dest, uint32_t c, size_t n)

  Set a block of PVR memory to a 16-bit value.

7.126.1 Detailed Description

SH4 CPU Peripheral for burst memory transactions.

The store queues are a way to do efficient burst transfers from the CPU to external memory. They can be used in a variety of ways, such as to transfer a texture to PVR memory. The transfers are in units of 32-bytes, and the destinations must be 32-byte aligned.

Note

Mastery over knowing when and how to utilize the store queues is important when trying to push the limits of the Dreamcast, specifically when transferring chunks of data between regions of memory. It is often the case that the DMA is faster for transactions which are consistently large; however, the store queues tend to have better performance and have less configuration overhead when bursting smaller chunks of data.

7.126.2 Macro Definition Documentation

QACR0

```
#define QACR0 (*(volatile uint32_t *)(void *)0xff000038)
```

Store Queue 0 access register.

QACR1

```
#define QACR1 (*(volatile uint32_t *)(void *)0xff00003c)
```

Store Queue 1 access register <>

SET_QACR_REGS

```
#define SET_QACR_REGS(
              dest0,
              dest1 )
```

Value:

```
    do { \
        QACR0 = ((uintptr_t)(dest0)) » 24; \
        QACR1 = ((uintptr_t)(dest1)) » 24; \
    } while(0)
```

Set Store Queue QACR* registers.

SQ_MASK_DEST

```
#define SQ_MASK_DEST(
              dest )  ((uint32_t *)(void *) SQ_MASK_DEST_ADDR(dest))
```

Mask dest to Store Queue area as pointer.

SQ_MASK_DEST_ADDR

```
#define SQ_MASK_DEST_ADDR(
              dest )  (MEM_AREA_SQ_BASE | ((uintptr_t)(dest) & 0x03ffffe0))
```

Mask dest to Store Queue area as address.

7.126.3 Function Documentation

sq_clr()

```
void sq_clr (
             void * dest,
             size_t n )
```

Clear a block of memory.

This function is similar to calling memset() with a value to set of 0, but uses the store queues to do its work.

Warning

The dest pointer must be a 32-byte aligned with n being a multiple of 32!

Parameters

dest	The address to begin clearing at (32-byte aligned).
n	The number of bytes to clear (multiple of 32).

sq_cpy()

```
void * sq_cpy (
            void * dest,
            const void * src,
            size_t n )
```

Copy a block of memory.

This function is similar to memcpy4(), but uses the store queues to do its work.

Warning

The dest pointer must be at least 32-byte aligned, the src pointer must be at least 4-byte aligned (8-byte aligned uses fast path), and n must be a multiple of 32!

Parameters

dest	The address to copy to (32-byte aligned).
src	The address to copy from (32-bit (4/8-byte) aligned).
n	The number of bytes to copy (multiple of 32).

Returns

The original value of dest.

See also

sq_cpy_pvr()

sq_cpy_pvr()

```
void * sq_cpy_pvr (
            void * dest,
            const void * src,
            size_t n )
```

Copy a block of memory to VRAM.

Author

TapamN

This function is similar to sq_cpy(), but it has been optimized for writing to a destination residing within VRAM.

Note

TapamN has reported over a 2x speedup versus the regular sq_cpy() when using this function to write to VRAM.

Warning

This function cannot be used at the same time as a PVR DMA transfer.

The dest pointer must be at least 32-byte aligned and reside in video memory, the src pointer must be at least 8-byte aligned, and n must be a multiple of 32.

Parameters

dest	The address to copy to (32-byte aligned).
src	The address to copy from (32-bit (8-byte) aligned).
n	The number of bytes to copy (multiple of 32).

Returns

The original value of dest.

See also

sq_cpy()

sq_lock()

```
void sq_lock (
              void  )
```

Lock Store Queues.

Locks the store queues so that they cannot be used from another thread until unlocked.

Warning

This function is called automatically by the store queue API provided by KOS; however, it must be called manually when driving the SQs directly from outside of this API.

See also

sq_unlock()

sq_set()

```
void * sq_set (
             void * dest,
             uint32_t c,
             size_t n )
```

Set a block of memory to an 8-bit value.

This function is similar to calling memset(), but uses the store queues to do its work.

Warning

The dest pointer must be a 32-byte aligned with n being a multiple of 32, and only the low 8-bits are used from c.

Parameters

dest	The address to begin setting at (32-byte aligned).
c	The value to set (in the low 8-bits).
n	The number of bytes to set (multiple of 32).

Returns

The original value of dest.

See also

sq_set16(), sq_set32(), sq_set_pvr()

sq_set16()

```
void * sq_set16 (
             void * dest,
             uint32_t c,
             size_t n )
```

Set a block of memory to a 16-bit value.

This function is similar to calling memset2(), but uses the store queues to do its work.

Warning

The dest pointer must be a 32-byte aligned with n being a multiple of 32, and only the low 16-bits are used from c.

Parameters

dest	The address to begin setting at (32-byte aligned).
c	The value to set (in the low 16-bits).
n	The number of bytes to set (multiple of 32).

Returns

The original value of dest.

See also

sq_set(), sq_set32(), sq_set_pvr()

sq_set32()

```
void * sq_set32 (
            void * dest,
            uint32_t c,
            size_t n )
```

Set a block of memory to a 32-bit value.

This function is similar to calling memset4(), but uses the store queues to do its work.

Warning

The dest pointer must be a 32-byte aligned with n being a multiple of 32!

Parameters

dest	The address to begin setting at (32-byte aligned).
c	The value to set (all 32-bits).
n	The number of bytes to set (multiple of 32).

Returns

The original value of dest.

See also

sq_set(), sq_set16(), sq_set_pvr()

sq_set_pvr()

```
void * sq_set_pvr (
            void * dest,
            uint32_t c,
            size_t n )
```

Set a block of PVR memory to a 16-bit value.

This function is similar to sq_set16(), but it has been optimized for writing to a destination residing within VRAM.

Warning

This function cannot be used at the same time as a PVR DMA transfer.

The dest pointer must be at least 32-byte aligned and reside in video memory, n must be a multiple of 32 and only the low 16-bits are used from c.

Parameters

dest	The address to begin setting at (32-byte aligned).
c	The value to set (in the low 16-bits).
n	The number of bytes to set (multiple of 32).

Returns

The original value of dest.

See also

sq_set(), sq_set16(), sq_set32()

sq_unlock()

```
void sq_unlock (
            void  )
```

Unlock Store Queues.

Unlocks the store queues so that they can be used from any thread.

Note

sq_lock() should've already been called previously.

Warning

sq_lock() and sq_unlock() are called automatically by the store queue API provided by KOS; however, they must be called manually when driving the SQs directly from outside this API.

See also

sq_lock()

7.127 Structure of the Bios Font

Modules

- Builtin VMU Icons

Macros

- #define BFONT_NARROW_START 0

  Start of Narrow Characters in Font Block.
- #define BFONT_OVERBAR BFONT_NARROW_START
- #define BFONT_ISO_8859_1_33_126 BFONT_NARROW_START+(1∗BFONT_THIN_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_YEN BFONT_NARROW_START+(95∗BFONT_THIN_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_ISO_8859_1_160_255 BFONT_NARROW_START+(96∗BFONT_THIN_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_WIDE_START (288∗BFONT_THIN_WIDTH∗BFONT_HEIGHT/8)

  Start of Wide Characters in Font Block.
- #define BFONT_JISX_0208_ROW1 BFONT_WIDE_START

  Start of JISX-0208 Rows 1-7 in Font Block.
- #define BFONT_JISX_0208_ROW16 BFONT_WIDE_START+(658∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)

  Start of JISX-0208 Row 16-47 (Start of Level 1) in Font Block.
- #define BFONT_JISX_0208_ROW48 BFONT_JISX_0208_ROW16+((32∗JISX_0208_ROW_SIZE)∗BFONT_WIDE_WIDTH∗BFON

  JISX-0208 Row 48-84 (Start of Level 2) in Font Block.
- #define BFONT_DREAMCAST_SPECIFIC BFONT_WIDE_START+(7056∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)

  Start of DC Specific Characters in Font Block.
- #define BFONT_CIRCLECOPYRIGHT BFONT_DREAMCAST_SPECIFIC+(0∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_CIRCLER BFONT_DREAMCAST_SPECIFIC+(1∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_TRADEMARK BFONT_DREAMCAST_SPECIFIC+(2∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_UPARROW BFONT_DREAMCAST_SPECIFIC+(3∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_DOWNARROW BFONT_DREAMCAST_SPECIFIC+(4∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_LEFTARROW BFONT_DREAMCAST_SPECIFIC+(5∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_RIGHTARROW BFONT_DREAMCAST_SPECIFIC+(6∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_UPRIGHTARROW BFONT_DREAMCAST_SPECIFIC+(7∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_DOWNRIGHTARROW BFONT_DREAMCAST_SPECIFIC+(8∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_DOWNLEFTARROW BFONT_DREAMCAST_SPECIFIC+(9∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_UPLEFTARROW BFONT_DREAMCAST_SPECIFIC+(10∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_ABUTTON BFONT_DREAMCAST_SPECIFIC+(11∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_BBUTTON BFONT_DREAMCAST_SPECIFIC+(12∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_CBUTTON BFONT_DREAMCAST_SPECIFIC+(13∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_DBUTTON BFONT_DREAMCAST_SPECIFIC+(14∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_XBUTTON BFONT_DREAMCAST_SPECIFIC+(15∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_YBUTTON BFONT_DREAMCAST_SPECIFIC+(16∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_ZBUTTON BFONT_DREAMCAST_SPECIFIC+(17∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_LTRIGGER BFONT_DREAMCAST_SPECIFIC+(18∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_RTRIGGER BFONT_DREAMCAST_SPECIFIC+(19∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_STARTBUTTON BFONT_DREAMCAST_SPECIFIC+(20∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_VMUICON BFONT_DREAMCAST_SPECIFIC+(21∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_ICON_DIMEN 32

  Dimension of vmu icons.
- #define BFONT_VMU_DREAMCAST_SPECIFIC BFONT_DREAMCAST_SPECIFIC+(22∗BFONT_WIDE_WIDTH∗BFONT_HEIGH

7.127.1 Detailed Description

7.127.2 Macro Definition Documentation

BFONT_ABUTTON

```
#define BFONT_ABUTTON BFONT_DREAMCAST_SPECIFIC+(11*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_BBUTTON

```
#define BFONT_BBUTTON BFONT_DREAMCAST_SPECIFIC+(12*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_CBUTTON

```
#define BFONT_CBUTTON BFONT_DREAMCAST_SPECIFIC+(13*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_CIRCLECOPYRIGHT

```
#define BFONT_CIRCLECOPYRIGHT BFONT_DREAMCAST_SPECIFIC+(0*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_CIRCLER

```
#define BFONT_CIRCLER BFONT_DREAMCAST_SPECIFIC+(1*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_DBUTTON

```
#define BFONT_DBUTTON BFONT_DREAMCAST_SPECIFIC+(14*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_DOWNARROW

```
#define BFONT_DOWNARROW BFONT_DREAMCAST_SPECIFIC+(4*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_DOWNLEFTARROW

```
#define BFONT_DOWNLEFTARROW BFONT_DREAMCAST_SPECIFIC+(9*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_DOWNRIGHTARROW

```
#define BFONT_DOWNRIGHTARROW BFONT_DREAMCAST_SPECIFIC+(8*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_DREAMCAST_SPECIFIC

```
#define BFONT_DREAMCAST_SPECIFIC BFONT_WIDE_START+(7056*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

Start of DC Specific Characters in Font Block.

BFONT_ICON_DIMEN

```
#define BFONT_ICON_DIMEN 32
```

Dimension of vmu icons.

BFONT_ISO_8859_1_160_255

```
#define BFONT_ISO_8859_1_160_255 BFONT_NARROW_START+(96*BFONT_THIN_WIDTH*BFONT_HEIGHT/8)
```

BFONT_ISO_8859_1_33_126

```
#define BFONT_ISO_8859_1_33_126 BFONT_NARROW_START+( 1*BFONT_THIN_WIDTH*BFONT_HEIGHT/8)
```

BFONT_JISX_0208_ROW1

```
#define BFONT_JISX_0208_ROW1 BFONT_WIDE_START
```

Start of JISX-0208 Rows 1-7 in Font Block.

BFONT_JISX_0208_ROW16

```
#define BFONT_JISX_0208_ROW16 BFONT_WIDE_START+(658*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

Start of JISX-0208 Row 16-47 (Start of Level 1) in Font Block.

BFONT_JISX_0208_ROW48

```
#define BFONT_JISX_0208_ROW48 BFONT_JISX_0208_ROW16+((32*JISX_0208_ROW_SIZE)*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

JISX-0208 Row 48-84 (Start of Level 2) in Font Block.

BFONT_LEFTARROW

```
#define BFONT_LEFTARROW BFONT_DREAMCAST_SPECIFIC+(5*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_LTRIGGER

```
#define BFONT_LTRIGGER BFONT_DREAMCAST_SPECIFIC+(18*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_NARROW_START

```
#define BFONT_NARROW_START 0
```

Start of Narrow Characters in Font Block.

BFONT_OVERBAR

```
#define BFONT_OVERBAR BFONT_NARROW_START
```

BFONT_RIGHTARROW

```
#define BFONT_RIGHTARROW BFONT_DREAMCAST_SPECIFIC+(6*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_RTRIGGER

```
#define BFONT_RTRIGGER BFONT_DREAMCAST_SPECIFIC+(19*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_STARTBUTTON

```
#define BFONT_STARTBUTTON BFONT_DREAMCAST_SPECIFIC+(20*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_TRADEMARK

```
#define BFONT_TRADEMARK BFONT_DREAMCAST_SPECIFIC+(2*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_UPARROW

```
#define BFONT_UPARROW BFONT_DREAMCAST_SPECIFIC+(3*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_UPLEFTARROW

```
#define BFONT_UPLEFTARROW BFONT_DREAMCAST_SPECIFIC+(10*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_UPRIGHTARROW

```
#define BFONT_UPRIGHTARROW BFONT_DREAMCAST_SPECIFIC+(7*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_VMU_DREAMCAST_SPECIFIC

```
#define BFONT_VMU_DREAMCAST_SPECIFIC BFONT_DREAMCAST_SPECIFIC+(22*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_VMUICON

```
#define BFONT_VMUICON BFONT_DREAMCAST_SPECIFIC+(21*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_WIDE_START

```
#define BFONT_WIDE_START (288*BFONT_THIN_WIDTH*BFONT_HEIGHT/8)
```

Start of Wide Characters in Font Block.

BFONT_XBUTTON

```
#define BFONT_XBUTTON BFONT_DREAMCAST_SPECIFIC+(15*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_YBUTTON

```
#define BFONT_YBUTTON BFONT_DREAMCAST_SPECIFIC+(16*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

BFONT_YEN

```
#define BFONT_YEN BFONT_NARROW_START+(95*BFONT_THIN_WIDTH*BFONT_HEIGHT/8)
```

BFONT_ZBUTTON

```
#define BFONT_ZBUTTON BFONT_DREAMCAST_SPECIFIC+(17*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
```

7.127.3 Builtin VMU Icons

Macros

- #define BFONT_ICON_INVALID_VMU 0x00
- #define BFONT_ICON_HOURGLASS_ONE 0x01
- #define BFONT_ICON_HOURGLASS_TWO 0x02
- #define BFONT_ICON_HOURGLASS_THREE 0x03
- #define BFONT_ICON_HOURGLASS_FOUR 0x04
- #define BFONT_ICON_VMUICON 0x05
- #define BFONT_ICON_EARTH 0x06
- #define BFONT_ICON_SATURN 0x07
- #define BFONT_ICON_QUARTER_MOON 0x08
- #define BFONT_ICON_LAUGHING_FACE 0x09
- #define BFONT_ICON_SMILING_FACE 0x0A
- #define BFONT_ICON_CASUAL_FACE 0x0B
- #define BFONT_ICON_ANGRY_FACE 0x0C
- #define BFONT_ICON_COW 0x0D
- #define BFONT_ICON_HORSE 0x0E
- #define BFONT_ICON_RABBIT 0x0F
- #define BFONT_ICON_CAT 0x10
- #define BFONT_ICON_CHICK 0x11
- #define BFONT_ICON_LION 0x12
- #define BFONT_ICON_MONKEY 0x13
- #define BFONT_ICON_PANDA 0x14
- #define BFONT_ICON_BEAR 0x15
- #define BFONT_ICON_PIG 0x16
- #define BFONT_ICON_DOG 0x17
- #define BFONT_ICON_FISH 0x18
- #define BFONT_ICON_OCTOPUS 0x19
- #define BFONT_ICON_SQUID 0x1A
- #define BFONT_ICON_WHALE 0x1B
- #define BFONT_ICON_CRAB 0x1C
- #define BFONT_ICON_BUTTERFLY 0x1D
- #define BFONT_ICON_LADYBUG 0x1E
- #define BFONT_ICON_ANGLER_FISH 0x1F
- #define BFONT_ICON_PENGUIN 0x20
- #define BFONT_ICON_CHERRIES 0x21
- #define BFONT_ICON_TULIP 0x22
- #define BFONT_ICON_LEAF 0x23
- #define BFONT_ICON_SAKURA 0x24
- #define BFONT_ICON_APPLE 0x25
- #define BFONT_ICON_ICECREAM 0x26
- #define BFONT_ICON_CACTUS 0x27
- #define BFONT_ICON_PIANO 0x28
- #define BFONT_ICON_GUITAR 0x29
- #define BFONT_ICON_EIGHTH_NOTE 0x2A
- #define BFONT_ICON_TREBLE_CLEF 0x2B
- #define BFONT_ICON_BOAT 0x2C
- #define BFONT_ICON_CAR 0x2D

- #define BFONT_ICON_HELMET 0x2E
- #define BFONT_ICON_MOTORCYCLE 0x2F
- #define BFONT_ICON_VAN 0x30
- #define BFONT_ICON_TRUCK 0x31
- #define BFONT_ICON_CLOCK 0x32
- #define BFONT_ICON_TELEPHONE 0x33
- #define BFONT_ICON_PENCIL 0x34
- #define BFONT_ICON_CUP 0x35
- #define BFONT_ICON_SILVERWARE 0x36
- #define BFONT_ICON_HOUSE 0x37
- #define BFONT_ICON_BELL 0x38
- #define BFONT_ICON_CROWN 0x39
- #define BFONT_ICON_SOCK 0x3A
- #define BFONT_ICON_CAKE 0x3B
- #define BFONT_ICON_KEY 0x3C
- #define BFONT_ICON_BOOK 0x3D
- #define BFONT_ICON_BASEBALL 0x3E
- #define BFONT_ICON_SOCCER 0x3F
- #define BFONT_ICON_BULB 0x40
- #define BFONT_ICON_TEDDY_BEAR 0x41
- #define BFONT_ICON_BOW_TIE 0x42
- #define BFONT_ICON_BOW_ARROW 0x43
- #define BFONT_ICON_SNOWMAN 0x44
- #define BFONT_ICON_LIGHTNING 0x45
- #define BFONT_ICON_SUN 0x46
- #define BFONT_ICON_CLOUD 0x47
- #define BFONT_ICON_UMBRELLA 0x48
- #define BFONT_ICON_ONE_STAR 0x49
- #define BFONT_ICON_TWO_STARS 0x4A
- #define BFONT_ICON_THREE_STARS 0x4B
- #define BFONT_ICON_FOUR_STARS 0x4C
- #define BFONT_ICON_HEART 0x4D
- #define BFONT_ICON_DIAMOND 0x4E
- #define BFONT_ICON_SPADE 0x4F
- #define BFONT_ICON_CLUB 0x50
- #define BFONT_ICON_JACK 0x51
- #define BFONT_ICON_QUEEN 0x52
- #define BFONT_ICON_KING 0x53
- #define BFONT_ICON_JOKER 0x54
- #define BFONT_ICON_ISLAND 0x55
- #define BFONT_ICON_0 0x56
- #define BFONT_ICON_1 0x57
- #define BFONT_ICON_2 0x58
- #define BFONT_ICON_3 0x59
- #define BFONT_ICON_4 0x5A
- #define BFONT_ICON_5 0x5B
- #define BFONT_ICON_6 0x5C
- #define BFONT_ICON_7 0x5D
- #define BFONT_ICON_8 0x5E
- #define BFONT_ICON_9 0x5F
- #define BFONT_ICON_A 0x60

- #define BFONT_ICON_B 0x61
- #define BFONT_ICON_C 0x62
- #define BFONT_ICON_D 0x63
- #define BFONT_ICON_E 0x64
- #define BFONT_ICON_F 0x65
- #define BFONT_ICON_G 0x66
- #define BFONT_ICON_H 0x67
- #define BFONT_ICON_I 0x68
- #define BFONT_ICON_J 0x69
- #define BFONT_ICON_K 0x6A
- #define BFONT_ICON_L 0x6B
- #define BFONT_ICON_M 0x6C
- #define BFONT_ICON_N 0x6D
- #define BFONT_ICON_O 0x6E
- #define BFONT_ICON_P 0x6F
- #define BFONT_ICON_Q 0x70
- #define BFONT_ICON_R 0x71
- #define BFONT_ICON_S 0x72
- #define BFONT_ICON_T 0x73
- #define BFONT_ICON_U 0x74
- #define BFONT_ICON_V 0x75
- #define BFONT_ICON_W 0x76
- #define BFONT_ICON_X 0x77
- #define BFONT_ICON_Y 0x78
- #define BFONT_ICON_Z 0x79
- #define BFONT_ICON_CHECKER_BOARD 0x7A
- #define BFONT_ICON_GRID 0x7B
- #define BFONT_ICON_LIGHT_GRAY 0x7C
- #define BFONT_ICON_DIAG_GRID 0x7D
- #define BFONT_ICON_PACMAN_GRID 0x7E
- #define BFONT_ICON_DARK_GRAY 0x7F
- #define BFONT_ICON_EMBROIDERY 0x80

7.127.3.1 Detailed Description

Builtin VMU volume user icons. The Dreamcast's BIOS allows the user to set these when formatting the VMU.

7.127.3.2 Macro Definition Documentation

BFONT_ICON_0

```
#define BFONT_ICON_0 0x56
```

BFONT_ICON_1

```
#define BFONT_ICON_1 0x57
```

BFONT_ICON_2

```
#define BFONT_ICON_2 0x58
```

BFONT_ICON_3

```
#define BFONT_ICON_3 0x59
```

BFONT_ICON_4

```
#define BFONT_ICON_4 0x5A
```

BFONT_ICON_5

```
#define BFONT_ICON_5 0x5B
```

BFONT_ICON_6

```
#define BFONT_ICON_6 0x5C
```

BFONT_ICON_7

```
#define BFONT_ICON_7 0x5D
```

BFONT_ICON_8

```
#define BFONT_ICON_8 0x5E
```

BFONT_ICON_9

```
#define BFONT_ICON_9 0x5F
```

BFONT_ICON_A

```
#define BFONT_ICON_A 0x60
```

BFONT_ICON_ANGLER_FISH

```
#define BFONT_ICON_ANGLER_FISH 0x1F
```

BFONT_ICON_ANGRY_FACE

```
#define BFONT_ICON_ANGRY_FACE 0x0C
```

BFONT_ICON_APPLE

```
#define BFONT_ICON_APPLE 0x25
```

BFONT_ICON_B

```
#define BFONT_ICON_B 0x61
```

BFONT_ICON_BASEBALL

```
#define BFONT_ICON_BASEBALL 0x3E
```

BFONT_ICON_BEAR

```
#define BFONT_ICON_BEAR 0x15
```

BFONT_ICON_BELL

```
#define BFONT_ICON_BELL 0x38
```

BFONT_ICON_BOAT

```
#define BFONT_ICON_BOAT 0x2C
```

BFONT_ICON_BOOK

```
#define BFONT_ICON_BOOK 0x3D
```

BFONT_ICON_BOW_ARROW

```
#define BFONT_ICON_BOW_ARROW 0x43
```

BFONT_ICON_BOW_TIE

```
#define BFONT_ICON_BOW_TIE 0x42
```

BFONT_ICON_BULB

```
#define BFONT_ICON_BULB 0x40
```

BFONT_ICON_BUTTERFLY

```
#define BFONT_ICON_BUTTERFLY 0x1D
```

BFONT_ICON_C

```
#define BFONT_ICON_C 0x62
```

BFONT_ICON_CACTUS

```
#define BFONT_ICON_CACTUS 0x27
```

BFONT_ICON_CAKE

```
#define BFONT_ICON_CAKE 0x3B
```

BFONT_ICON_CAR

```
#define BFONT_ICON_CAR 0x2D
```

BFONT_ICON_CASUAL_FACE

```
#define BFONT_ICON_CASUAL_FACE 0x0B
```

BFONT_ICON_CAT

```
#define BFONT_ICON_CAT 0x10
```

BFONT_ICON_CHECKER_BOARD

```
#define BFONT_ICON_CHECKER_BOARD 0x7A
```

BFONT_ICON_CHERRIES

```
#define BFONT_ICON_CHERRIES 0x21
```

BFONT_ICON_CHICK

```
#define BFONT_ICON_CHICK 0x11
```

BFONT_ICON_CLOCK

```
#define BFONT_ICON_CLOCK 0x32
```

BFONT_ICON_CLOUD

```
#define BFONT_ICON_CLOUD 0x47
```

BFONT_ICON_CLUB

```
#define BFONT_ICON_CLUB 0x50
```

BFONT_ICON_COW

```
#define BFONT_ICON_COW 0x0D
```

BFONT_ICON_CRAB

```
#define BFONT_ICON_CRAB 0x1C
```

BFONT_ICON_CROWN

```
#define BFONT_ICON_CROWN 0x39
```

BFONT_ICON_CUP

```
#define BFONT_ICON_CUP 0x35
```

BFONT_ICON_D

```
#define BFONT_ICON_D 0x63
```

BFONT_ICON_DARK_GRAY

```
#define BFONT_ICON_DARK_GRAY 0x7F
```

BFONT_ICON_DIAG_GRID

```
#define BFONT_ICON_DIAG_GRID 0x7D
```

BFONT_ICON_DIAMOND

```
#define BFONT_ICON_DIAMOND 0x4E
```

BFONT_ICON_DOG

```
#define BFONT_ICON_DOG 0x17
```

BFONT_ICON_E

```
#define BFONT_ICON_E 0x64
```

BFONT_ICON_EARTH

```
#define BFONT_ICON_EARTH 0x06
```

BFONT_ICON_EIGHTH_NOTE

```
#define BFONT_ICON_EIGHTH_NOTE 0x2A
```

BFONT_ICON_EMBROIDERY

```
#define BFONT_ICON_EMBROIDERY 0x80
```

BFONT_ICON_F

```
#define BFONT_ICON_F 0x65
```

BFONT_ICON_FISH

```
#define BFONT_ICON_FISH 0x18
```

BFONT_ICON_FOUR_STARS

```
#define BFONT_ICON_FOUR_STARS 0x4C
```

BFONT_ICON_G

```
#define BFONT_ICON_G 0x66
```

BFONT_ICON_GRID

```
#define BFONT_ICON_GRID 0x7B
```

BFONT_ICON_GUITAR

```
#define BFONT_ICON_GUITAR 0x29
```

BFONT_ICON_H

```
#define BFONT_ICON_H 0x67
```

BFONT_ICON_HEART

```
#define BFONT_ICON_HEART 0x4D
```

BFONT_ICON_HELMET

```
#define BFONT_ICON_HELMET 0x2E
```

BFONT_ICON_HORSE

```
#define BFONT_ICON_HORSE 0x0E
```

BFONT_ICON_HOURGLASS_FOUR

```
#define BFONT_ICON_HOURGLASS_FOUR 0x04
```

BFONT_ICON_HOURGLASS_ONE

```
#define BFONT_ICON_HOURGLASS_ONE 0x01
```

BFONT_ICON_HOURGLASS_THREE

```
#define BFONT_ICON_HOURGLASS_THREE 0x03
```

BFONT_ICON_HOURGLASS_TWO

```
#define BFONT_ICON_HOURGLASS_TWO 0x02
```

BFONT_ICON_HOUSE

```
#define BFONT_ICON_HOUSE 0x37
```

BFONT_ICON_I

```
#define BFONT_ICON_I 0x68
```

BFONT_ICON_ICECREAM

```
#define BFONT_ICON_ICECREAM 0x26
```

BFONT_ICON_INVALID_VMU

```
#define BFONT_ICON_INVALID_VMU 0x00
```

BFONT_ICON_ISLAND

```
#define BFONT_ICON_ISLAND 0x55
```

BFONT_ICON_J

```
#define BFONT_ICON_J 0x69
```

BFONT_ICON_JACK

```
#define BFONT_ICON_JACK 0x51
```

BFONT_ICON_JOKER

```
#define BFONT_ICON_JOKER 0x54
```

BFONT_ICON_K

```
#define BFONT_ICON_K 0x6A
```

BFONT_ICON_KEY

```
#define BFONT_ICON_KEY 0x3C
```

BFONT_ICON_KING

```
#define BFONT_ICON_KING 0x53
```

BFONT_ICON_L

```
#define BFONT_ICON_L 0x6B
```

BFONT_ICON_LADYBUG

```
#define BFONT_ICON_LADYBUG 0x1E
```

BFONT_ICON_LAUGHING_FACE

```
#define BFONT_ICON_LAUGHING_FACE 0x09
```

BFONT_ICON_LEAF

```
#define BFONT_ICON_LEAF 0x23
```

BFONT_ICON_LIGHT_GRAY

```
#define BFONT_ICON_LIGHT_GRAY 0x7C
```

BFONT_ICON_LIGHTNING

```
#define BFONT_ICON_LIGHTNING 0x45
```

BFONT_ICON_LION

```
#define BFONT_ICON_LION 0x12
```

BFONT_ICON_M

```
#define BFONT_ICON_M 0x6C
```

BFONT_ICON_MONKEY

```
#define BFONT_ICON_MONKEY 0x13
```

BFONT_ICON_MOTORCYCLE

```
#define BFONT_ICON_MOTORCYCLE 0x2F
```

BFONT_ICON_N

```
#define BFONT_ICON_N 0x6D
```

BFONT_ICON_O

```
#define BFONT_ICON_O 0x6E
```

BFONT_ICON_OCTOPUS

```
#define BFONT_ICON_OCTOPUS 0x19
```

BFONT_ICON_ONE_STAR

```
#define BFONT_ICON_ONE_STAR 0x49
```

BFONT_ICON_P

```
#define BFONT_ICON_P 0x6F
```

BFONT_ICON_PACMAN_GRID

```
#define BFONT_ICON_PACMAN_GRID 0x7E
```

BFONT_ICON_PANDA

```
#define BFONT_ICON_PANDA 0x14
```

BFONT_ICON_PENCIL

```
#define BFONT_ICON_PENCIL 0x34
```

BFONT_ICON_PENGUIN

```
#define BFONT_ICON_PENGUIN 0x20
```

BFONT_ICON_PIANO

```
#define BFONT_ICON_PIANO 0x28
```

BFONT_ICON_PIG

```
#define BFONT_ICON_PIG 0x16
```

BFONT_ICON_Q

```
#define BFONT_ICON_Q 0x70
```

BFONT_ICON_QUARTER_MOON

```
#define BFONT_ICON_QUARTER_MOON 0x08
```

BFONT_ICON_QUEEN

```
#define BFONT_ICON_QUEEN 0x52
```

BFONT_ICON_R

```
#define BFONT_ICON_R 0x71
```

BFONT_ICON_RABBIT

```
#define BFONT_ICON_RABBIT 0x0F
```

BFONT_ICON_S

```
#define BFONT_ICON_S 0x72
```

BFONT_ICON_SAKURA

```
#define BFONT_ICON_SAKURA 0x24
```

BFONT_ICON_SATURN

```
#define BFONT_ICON_SATURN 0x07
```

BFONT_ICON_SILVERWARE

```
#define BFONT_ICON_SILVERWARE 0x36
```

BFONT_ICON_SMILING_FACE

```
#define BFONT_ICON_SMILING_FACE 0x0A
```

BFONT_ICON_SNOWMAN

```
#define BFONT_ICON_SNOWMAN 0x44
```

BFONT_ICON_SOCCER

```
#define BFONT_ICON_SOCCER 0x3F
```

BFONT_ICON_SOCK

```
#define BFONT_ICON_SOCK 0x3A
```

BFONT_ICON_SPADE

```
#define BFONT_ICON_SPADE 0x4F
```

BFONT_ICON_SQUID

```
#define BFONT_ICON_SQUID 0x1A
```

BFONT_ICON_SUN

```
#define BFONT_ICON_SUN 0x46
```

BFONT_ICON_T

```
#define BFONT_ICON_T 0x73
```

BFONT_ICON_TEDDY_BEAR

```
#define BFONT_ICON_TEDDY_BEAR 0x41
```

BFONT_ICON_TELEPHONE

```
#define BFONT_ICON_TELEPHONE 0x33
```

BFONT_ICON_THREE_STARS

```
#define BFONT_ICON_THREE_STARS 0x4B
```

BFONT_ICON_TREBLE_CLEF

```
#define BFONT_ICON_TREBLE_CLEF 0x2B
```

BFONT_ICON_TRUCK

```
#define BFONT_ICON_TRUCK 0x31
```

BFONT_ICON_TULIP

```
#define BFONT_ICON_TULIP 0x22
```

BFONT_ICON_TWO_STARS

```
#define BFONT_ICON_TWO_STARS 0x4A
```

BFONT_ICON_U

```
#define BFONT_ICON_U 0x74
```

BFONT_ICON_UMBRELLA

```
#define BFONT_ICON_UMBRELLA 0x48
```

BFONT_ICON_V

```
#define BFONT_ICON_V 0x75
```

BFONT_ICON_VAN

```
#define BFONT_ICON_VAN 0x30
```

BFONT_ICON_VMUICON

```
#define BFONT_ICON_VMUICON 0x05
```

BFONT_ICON_W

```
#define BFONT_ICON_W 0x76
```

BFONT_ICON_WHALE

```
#define BFONT_ICON_WHALE 0x1B
```

BFONT_ICON_X

```
#define BFONT_ICON_X 0x77
```

BFONT_ICON_Y

```
#define BFONT_ICON_Y 0x78
```

BFONT_ICON_Z

```
#define BFONT_ICON_Z 0x79
```

7.128 Symbol binding types.

Macros

- #define STB_LOCAL 0

  Local (non-exported) symbol.
- #define STB_GLOBAL 1

  Global (exported) symbol.
- #define STB_WEAK 2

  Weak-linked symbol.

7.128.1 Detailed Description

These are the values that can be set to say how a symbol is bound in an ELF binary. This is stored in the upper 4 bits of the info field in elf_sym_t.

7.128.2 Macro Definition Documentation

STB_GLOBAL

```
#define STB_GLOBAL 1
```

Global (exported) symbol.

STB_LOCAL

```
#define STB_LOCAL 0
```

Local (non-exported) symbol.

STB_WEAK

```
#define STB_WEAK 2
```

Weak-linked symbol.

7.129 Symbol types.

Macros

- #define STT_NOTYPE 0
  Symbol has no type.
- #define STT_OBJECT 1
  Symbol is an object.
- #define STT_FUNC 2
  Symbol is a function.
- #define STT_SECTION 3
  Symbol is a section.
- #define STT_FILE 4
  Symbol is a file name.

7.129.1 Detailed Description

These are the values that can be set to say what kind of symbol a given symbol in an ELF file is. This is stored in the lower 4 bits of the info field in elf_sym_t.

7.129.2 Macro Definition Documentation

STT_FILE

```
#define STT_FILE 4
```

Symbol is a file name.

STT_FUNC

```
#define STT_FUNC 2
```

Symbol is a function.

STT_NOTYPE

```
#define STT_NOTYPE 0
```

Symbol has no type.

STT_OBJECT

```
#define STT_OBJECT 1
```

Symbol is an object.

STT_SECTION

```
#define STT_SECTION 3
```

Symbol is a section.

7.130 TA command values

Macros

- #define PVR_CMD_POLYHDR 0x80840000
  PVR polygon header. Striplength set to 2.
- #define PVR_CMD_VERTEX 0xe0000000
  PVR vertex data.
- #define PVR_CMD_VERTEX_EOL 0xf0000000
  PVR vertex, end of strip.
- #define PVR_CMD_USERCLIP 0x20000000
  PVR user clipping area.
- #define PVR_CMD_MODIFIER 0x80000000
  PVR modifier volume.
- #define PVR_CMD_SPRITE 0xA0000000
  PVR sprite header.

7.130.1 Detailed Description

These are are appropriate values for TA commands. Use whatever goes with the primitive type you're using.

7.130.2 Macro Definition Documentation

PVR_CMD_MODIFIER

```
#define PVR_CMD_MODIFIER 0x80000000
```

PVR modifier volume.

PVR_CMD_POLYHDR

```
#define PVR_CMD_POLYHDR 0x80840000
```

PVR polygon header. Striplength set to 2.

PVR_CMD_SPRITE

```
#define PVR_CMD_SPRITE 0xA0000000
```

PVR sprite header.

PVR_CMD_USERCLIP

```
#define PVR_CMD_USERCLIP 0x20000000
```

PVR user clipping area.

PVR_CMD_VERTEX

```
#define PVR_CMD_VERTEX 0xe0000000
```

PVR vertex data.

PVR_CMD_VERTEX_EOL

```
#define PVR_CMD_VERTEX_EOL 0xf0000000
```

PVR vertex, end of strip.

7.131 TCP protocol level options

Macros

- #define TCP_NODELAY 1

  Don't delay to coalesce.

7.131.1 Detailed Description

These are the various socket-level optoins that can be accessed with the setsockopt() and getsockopt() fnctions for the IPPROTO_TCP level value.

All options listed here are at least guaranteed to be accepted by setsockopt() and getsockopt() for IPPROTO_TCP, however they are not guaranteed to be implemented in any meaningful way.

See also

Socket-level options

IPv4 protocol level options

IPv6 protocol level options

UDP protocol level options

UDP-Lite protocol level options

7.131.2 Macro Definition Documentation

TCP_NODELAY

```
#define TCP_NODELAY 1
```

Don't delay to coalesce.

7.132 Texture color calculation modes

Macros

- #define PVR_TXRENV_REPLACE 0

  C = Ct, A = At.
- #define PVR_TXRENV_MODULATE 1

  C = Cs ∗ Ct, A = At.
- #define PVR_TXRENV_DECAL 2

  C = (Cs ∗ At) + (Cs ∗ (1-At)), A = As.
- #define PVR_TXRENV_MODULATEALPHA 3

  C = Cs ∗ Ct, A = As ∗ At.

7.132.1 Detailed Description

7.132.2 Macro Definition Documentation

PVR_TXRENV_DECAL

```
#define PVR_TXRENV_DECAL 2
```

C = (Cs ∗ At) + (Cs ∗ (1-At)), A = As.

PVR_TXRENV_MODULATE

```
#define PVR_TXRENV_MODULATE 1
```

C = Cs ∗ Ct, A = At.

PVR_TXRENV_MODULATEALPHA

```
#define PVR_TXRENV_MODULATEALPHA 3
```

C = Cs ∗ Ct, A = As ∗ At.

PVR_TXRENV_REPLACE

```
#define PVR_TXRENV_REPLACE 0
```

C = Ct, A = At.

7.133 Texture loading constants

Macros

- #define PVR_TXRLOAD_4BPP 0x01
  4BPP format
- #define PVR_TXRLOAD_8BPP 0x02
  8BPP format
- #define PVR_TXRLOAD_16BPP 0x03
  16BPP format
- #define PVR_TXRLOAD_FMT_MASK 0x0f
  Bits used for basic formats.
- #define PVR_TXRLOAD_VQ_LOAD 0x10
  Do VQ encoding (not supported yet, if ever)
- #define PVR_TXRLOAD_INVERT_Y 0x20
  Invert the Y axis while loading.

- #define PVR_TXRLOAD_FMT_VQ 0x40

  Texture is already VQ encoded.
- #define PVR_TXRLOAD_FMT_TWIDDLED 0x80

  Texture is already twiddled.
- #define PVR_TXRLOAD_FMT_NOTWIDDLE 0x80

  Don't twiddle the texture while loading.
- #define PVR_TXRLOAD_DMA 0x8000

  Use DMA to load the texture.
- #define PVR_TXRLOAD_NONBLOCK 0x4000

  Use non-blocking loads (only for DMA)
- #define PVR_TXRLOAD_SQ 0x2000

  Use store queues to load.

7.133.1 Detailed Description

These are constants for the flags parameter to pvr_txr_load_ex() or pvr_txr_load_kimg().

7.133.2 Macro Definition Documentation

PVR_TXRLOAD_16BPP

```
#define PVR_TXRLOAD_16BPP 0x03
```

16BPP format

PVR_TXRLOAD_4BPP

```
#define PVR_TXRLOAD_4BPP 0x01
```

4BPP format

PVR_TXRLOAD_8BPP

```
#define PVR_TXRLOAD_8BPP 0x02
```

8BPP format

PVR_TXRLOAD_DMA

```
#define PVR_TXRLOAD_DMA 0x8000
```

Use DMA to load the texture.

PVR_TXRLOAD_FMT_MASK

```
#define PVR_TXRLOAD_FMT_MASK 0x0f
```

Bits used for basic formats.

PVR_TXRLOAD_FMT_NOTWIDDLE

```
#define PVR_TXRLOAD_FMT_NOTWIDDLE 0x80
```

Don't twiddle the texture while loading.

PVR_TXRLOAD_FMT_TWIDDLED

```
#define PVR_TXRLOAD_FMT_TWIDDLED 0x80
```

Texture is already twiddled.

PVR_TXRLOAD_FMT_VQ

```
#define PVR_TXRLOAD_FMT_VQ 0x40
```

Texture is already VQ encoded.

PVR_TXRLOAD_INVERT_Y

```
#define PVR_TXRLOAD_INVERT_Y 0x20
```

Invert the Y axis while loading.

PVR_TXRLOAD_NONBLOCK

```
#define PVR_TXRLOAD_NONBLOCK 0x4000
```

Use non-blocking loads (only for DMA)

PVR_TXRLOAD_SQ

```
#define PVR_TXRLOAD_SQ 0x2000
```

Use store queues to load.

PVR_TXRLOAD_VQ_LOAD

```
#define PVR_TXRLOAD_VQ_LOAD 0x10
```

Do VQ encoding (not supported yet, if ever)

7.134 Threading

Threading and Concurrency.

Modules

- KThreads

  KOS Native Threading API.

Files

- file pthread.h

  POSIX-compatibile (sorta) threading support.
- file threads.h

  C11 Threading API.

7.134.1 Detailed Description

Threading and Concurrency.

KOS offers a variety of different threading APIs, ranging from low-level and platform-specific to high-level and cross-platform. Which you should choose to interface with depends on the needs of your particular application.

API	Description
kthreads	Dreamcast-specific direct threading API
C11 threads	Cross-platform builtin C threading API
C++11 threads	Cross-platform builtin C++ threading API
pthreads	Cross-platform POSIX threading API

KOS's kthreads are the lowest-level threading back-end, with C11, C++11, and POSIX threading APIs being thin layers of abstraction built upon them. If you're writing a Dreamcast-specific application, kthreads will give you the least indirection; however, if you're writing a cross-platform application or simply want to use an API that you're familiar with, one of the other cross-platform APIs may be preferable.

7.134.2 KThreads

KOS Native Threading API.

Files

- file cond.h
  Condition variables.
- file genwait.h
  Generic wait system.
- file mutex.h
  Mutual exclusion locks.
- file once.h
  Dynamic package initialization.
- file recursive_lock.h
  Definitions for a recursive mutex.
- file rwsem.h
  Definition for a reader/writer semaphore.
- file sem.h
  Semaphores.
- file thread.h
  Threading support.
- file tls.h
  Thread-local storage support.

7.134.2.1 Detailed Description

KOS Native Threading API.

The thread scheduler itself is a relatively simplistic priority scheduler. There is no provision for priorities to erode over time, so keep that in mind. That practically means that if you have 2 high priority threads that are always runnable and one low priority thread that is always runnable, the low priority thread will never actually run (since it will never get to the front of the run queue because of the high priority threads).

The scheduler supports two distinct types of threads: joinable and detached threads. A joinable thread is one that can return a value to the creating thread (or for that matter, any other thread that wishes to join it). A detached thread is one that is completely detached from the rest of the system and cannot return values by "normal" means. Detached threads automatically clean up all of the internal resources associated with the thread when it exits. Joinable threads, on the other hand, must keep some state available for the ability to return values. To make sure that all memory allocated by the thread's internal structures gets freed, you must either join with the thread (with thd_join()) or detach it (with thd_detach()). The old KOS threading system only had what would be considered detached threads.

See also

semaphore_t, mutex_t, kthread_once_t, kthread_key_t, rw_semaphore_t

7.135 Transfer modes with PVR DMA

Macros

- #define PVR_DMA_VRAM64 0
  Transfer to VRAM in interleaved mode.
- #define PVR_DMA_VRAM32 1
  Transfer to VRAM in linear mode.
- #define PVR_DMA_TA 2
  Transfer to the tile accelerator.
- #define PVR_DMA_YUV 3
  Transfer to the YUV converter.

7.135.1 Detailed Description

7.135.2 Macro Definition Documentation

PVR_DMA_TA

```
#define PVR_DMA_TA 2
```

Transfer to the tile accelerator.

PVR_DMA_VRAM32

```
#define PVR_DMA_VRAM32 1
```

Transfer to VRAM in linear mode.

PVR_DMA_VRAM64

```
#define PVR_DMA_VRAM64 0
```

Transfer to VRAM in interleaved mode.

PVR_DMA_YUV

```
#define PVR_DMA_YUV 3
```

Transfer to the YUV converter.

7.136 UBC Registers

Macros

- #define BARA (∗((vuint32∗)0xFF200000))
  BARA register.
- #define BASRA (∗((vuint8∗)0xFF000014))
  BASRA register.
- #define BAMRA (∗((vuint8∗)0xFF200004))
  BAMRA register.
- #define BBRA (∗((vuint16∗)0xFF200008))
  BBRA register.
- #define BARB (∗((vuint32∗)0xFF20000C))
  BARB register.
- #define BASRB (∗((vuint8∗)0xFF000018))
  BASRB register.
- #define BAMRB (∗((vuint8∗)0xFF200010))
  BAMRB register.
- #define BBRB (∗((vuint16∗)0xFF200014))
  BBRB register.
- #define BRCR (∗((vuint16∗)0xFF200020))
  BRCR register.

7.136.1 Detailed Description

These registers are as documented in the SH4 manual. Consult it for more information.

7.136.2 Macro Definition Documentation

BAMRA

```
#define BAMRA (*((vuint8*)0xFF200004))
```

BAMRA register.

BAMRB

```
#define BAMRB (*((vuint8*)0xFF200010))
```

BAMRB register.

BARA

```
#define BARA (*((vuint32*)0xFF200000))
```

BARA register.

BARB

```
#define BARB (*((vuint32*)0xFF20000C))
```

BARB register.

BASRA

```
#define BASRA (*((vuint8*)0xFF000014))
```

BASRA register.

BASRB

```
#define BASRB (*((vuint8*)0xFF000018))
```

BASRB register.

BBRA

```
#define BBRA (*((vuint16*)0xFF200008))
```

BBRA register.

BBRB

```
#define BBRB (*((vuint16*)0xFF200014))
```

BBRB register.

BRCR

```
#define BRCR (*((vuint16*)0xFF200020))
```

BRCR register.

7.137 UDP protocol level options

Macros

- #define UDP_NOCHECKSUM 25

  Don't calculate UDP checksums.

7.137.1 Detailed Description

These are the various socket-level options that can be accessed with the setsockopt() and getsockopt() functions for the IPPROTO_UDP level value.

See also

Socket-level options
IPv4 protocol level options
IPv6 protocol level options
UDP-Lite protocol level options
TCP protocol level options

7.137.2 Macro Definition Documentation

UDP_NOCHECKSUM

```
#define UDP_NOCHECKSUM 25
```

Don't calculate UDP checksums.

7.138 UDP-Lite protocol level options

Macros

- #define UDPLITE_SEND_CSCOV 26

  Sending checksum coverage.
- #define UDPLITE_RECV_CSCOV 27

  Receiving checksum coverage.

7.138.1 Detailed Description

These are the various socket-level options that can be accessed with the setsockopt() and getsockopt() functions for the IPPROTO_UDPLITE level value.

See also

Socket-level options

IPv4 protocol level options

IPv6 protocol level options

UDP protocol level options

TCP protocol level options

7.138.2 Macro Definition Documentation

UDPLITE_RECV_CSCOV

```
#define UDPLITE_RECV_CSCOV 27
```

Receiving checksum coverage.

UDPLITE_SEND_CSCOV

```
#define UDPLITE_SEND_CSCOV 26
```

Sending checksum coverage.

7.139 Valid field constants for the flashrom_ispcfg_t struct

Macros

- #define FLASHROM_ISP_IP (1 << 0)
  Static IP address.
- #define FLASHROM_ISP_NETMASK (1 << 1)
  Netmask.
- #define FLASHROM_ISP_BROADCAST (1 << 2)
  Broadcast address.
- #define FLASHROM_ISP_GATEWAY (1 << 3)
  Gateway address.
- #define FLASHROM_ISP_DNS (1 << 4)
  DNS servers.
- #define FLASHROM_ISP_HOSTNAME (1 << 5)
  Hostname.
- #define FLASHROM_ISP_EMAIL (1 << 6)
  Email address.
- #define FLASHROM_ISP_SMTP (1 << 7)
  SMTP server.
- #define FLASHROM_ISP_POP3 (1 << 8)
  POP3 server.
- #define FLASHROM_ISP_POP3_USER (1 << 9)
  POP3 username.
- #define FLASHROM_ISP_POP3_PASS (1 << 10)
  POP3 password.
- #define FLASHROM_ISP_PROXY_HOST (1 << 11)
  Proxy hostname.
- #define FLASHROM_ISP_PROXY_PORT (1 << 12)
  Proxy port.
- #define FLASHROM_ISP_PPP_USER (1 << 13)
  PPP username.
- #define FLASHROM_ISP_PPP_PASS (1 << 14)
  PPP password.
- #define FLASHROM_ISP_OUT_PREFIX (1 << 15)
  Outside dial prefix.
- #define FLASHROM_ISP_CW_PREFIX (1 << 16)
  Call waiting prefix.
- #define FLASHROM_ISP_REAL_NAME (1 << 17)
  Real name.
- #define FLASHROM_ISP_MODEM_INIT (1 << 18)
  Modem init string.
- #define FLASHROM_ISP_AREA_CODE (1 << 19)
  Area code.
- #define FLASHROM_ISP_LD_PREFIX (1 << 20)
  Long distance prefix.
- #define FLASHROM_ISP_PHONE1 (1 << 21)
  Phone number 1.
- #define FLASHROM_ISP_PHONE2 (1 << 22)
  Phone number 2.

7.139.1 Detailed Description

The valid_fields field of the flashrom_ispcfg_t will have some combination of these ORed together to represent what data is filled in and believed valid.

7.139.2 Macro Definition Documentation

FLASHROM_ISP_AREA_CODE

```
#define FLASHROM_ISP_AREA_CODE (1 << 19)
```

Area code.

FLASHROM_ISP_BROADCAST

```
#define FLASHROM_ISP_BROADCAST (1 << 2)
```

Broadcast address.

FLASHROM_ISP_CW_PREFIX

```
#define FLASHROM_ISP_CW_PREFIX (1 << 16)
```

Call waiting prefix.

FLASHROM_ISP_DNS

```
#define FLASHROM_ISP_DNS (1 << 4)
```

DNS servers.

FLASHROM_ISP_EMAIL

```
#define FLASHROM_ISP_EMAIL (1 << 6)
```

Email address.

FLASHROM_ISP_GATEWAY

```
#define FLASHROM_ISP_GATEWAY (1 << 3)
```

Gateway address.

FLASHROM_ISP_HOSTNAME

```
#define FLASHROM_ISP_HOSTNAME (1 << 5)
```

Hostname.

FLASHROM_ISP_IP

```
#define FLASHROM_ISP_IP (1 << 0)
```

Static IP address.

FLASHROM_ISP_LD_PREFIX

```
#define FLASHROM_ISP_LD_PREFIX (1 << 20)
```

Long distance prefix.

FLASHROM_ISP_MODEM_INIT

```
#define FLASHROM_ISP_MODEM_INIT (1 << 18)
```

Modem init string.

FLASHROM_ISP_NETMASK

```
#define FLASHROM_ISP_NETMASK (1 << 1)
```

Netmask.

FLASHROM_ISP_OUT_PREFIX

```
#define FLASHROM_ISP_OUT_PREFIX (1 << 15)
```

Outside dial prefix.

FLASHROM_ISP_PHONE1

```
#define FLASHROM_ISP_PHONE1 (1 << 21)
```

Phone number 1.

FLASHROM_ISP_PHONE2

```
#define FLASHROM_ISP_PHONE2 (1 << 22)
```

Phone number 2.

FLASHROM_ISP_POP3

```
#define FLASHROM_ISP_POP3 (1 << 8)
```

POP3 server.

FLASHROM_ISP_POP3_PASS

```
#define FLASHROM_ISP_POP3_PASS (1 << 10)
```

POP3 password.

FLASHROM_ISP_POP3_USER

```
#define FLASHROM_ISP_POP3_USER (1 << 9)
```

POP3 username.

FLASHROM_ISP_PPP_PASS

```
#define FLASHROM_ISP_PPP_PASS (1 << 14)
```

PPP password.

FLASHROM_ISP_PPP_USER

```
#define FLASHROM_ISP_PPP_USER (1 << 13)
```

PPP username.

FLASHROM_ISP_PROXY_HOST

```
#define FLASHROM_ISP_PROXY_HOST (1 << 11)
```

Proxy hostname.

FLASHROM_ISP_PROXY_PORT

```
#define FLASHROM_ISP_PROXY_PORT (1 << 12)
```

Proxy port.

FLASHROM_ISP_REAL_NAME

```
#define FLASHROM_ISP_REAL_NAME (1 << 17)
```

Real name.

FLASHROM_ISP_SMTP

```
#define FLASHROM_ISP_SMTP (1 << 7)
```

SMTP server.

7.140 Values to write to Maple Bus registers

Macros

- #define MAPLE_RESET2_MAGIC 0
  2nd reset value
- #define MAPLE_ENABLE_ENABLED 1
  Enable Maple.
- #define MAPLE_ENABLE_DISABLED 0
  Disable Maple.
- #define MAPLE_STATE_IDLE 0
  Idle state.
- #define MAPLE_STATE_DMA 1
  DMA in-progress.
- #define MAPLE_SPEED_2MBPS 0
  2Mbps bus speed
- #define MAPLE_SPEED_TIMEOUT(n) ((n) << 16)
  Bus timeout macro.
- #define MAPLE_RESET1_MAGIC 0x6155404f
  First reset value.

7.140.1 Detailed Description

These are the values that are written to registers to get them to do their thing.

7.140.2 Macro Definition Documentation

MAPLE_ENABLE_DISABLED

```
#define MAPLE_ENABLE_DISABLED 0
```

Disable Maple.

MAPLE_ENABLE_ENABLED

```
#define MAPLE_ENABLE_ENABLED 1
```

Enable Maple.

MAPLE_RESET1_MAGIC

```
#define MAPLE_RESET1_MAGIC 0x6155404f
```

First reset value.

MAPLE_RESET2_MAGIC

```
#define MAPLE_RESET2_MAGIC 0
```

2nd reset value

MAPLE_SPEED_2MBPS

```
#define MAPLE_SPEED_2MBPS 0
```

2Mbps bus speed

MAPLE_SPEED_TIMEOUT

```
#define MAPLE_SPEED_TIMEOUT(
              n ) ((n) << 16)
```

Bus timeout macro.

MAPLE_STATE_DMA

```
#define MAPLE_STATE_DMA 1
```

DMA in-progress.

MAPLE_STATE_IDLE

```
#define MAPLE_STATE_IDLE 0
```

Idle state.

7.141 Values used to reset parts of the PVR

Macros

- #define PVR_RESET_ALL 0xffffffff
  Reset the whole PVR.
- #define PVR_RESET_NONE 0x00000000
  Cancel reset state.
- #define PVR_RESET_TA 0x00000001
  Reset only the TA.
- #define PVR_RESET_ISPTSP 0x00000002
  Reset only the ISP/TSP.

7.141.1 Detailed Description

These values are written to the PVR_RESET register in order to reset the system or to take it out of reset.

7.141.2 Macro Definition Documentation

PVR_RESET_ALL

```
#define PVR_RESET_ALL 0xffffffff
```

Reset the whole PVR.

PVR_RESET_ISPTSP

```
#define PVR_RESET_ISPTSP 0x00000002
```

Reset only the ISP/TSP.

PVR_RESET_NONE

```
#define PVR_RESET_NONE 0x00000000
```

Cancel reset state.

PVR_RESET_TA

```
#define PVR_RESET_TA 0x00000001
```

Reset only the TA.

7.142 Video Cable types

Macros

- #define CT_ANY -1

  Any cable type. Used only internally.
- #define CT_VGA 0

  VGA Box.
- #define CT_NONE 1

  Nothing connected.
- #define CT_RGB 2

  RGB/SCART cable.
- #define CT_COMPOSITE 3

  Composite cable or RF switch.

7.142.1 Detailed Description

The vid_check_cable() function will return one of this set of values to let you know what type of cable is connected to the Dreamcast. These are also used in the video mode settings to limit modes to certain cable types.

7.142.2 Macro Definition Documentation

CT_ANY

```
#define CT_ANY -1
```

Any cable type. Used only internally.

CT_COMPOSITE

```
#define CT_COMPOSITE 3
```

Composite cable or RF switch.

CT_NONE

```
#define CT_NONE 1
```

Nothing connected.

CT_RGB

```
#define CT_RGB 2
```

RGB/SCART cable.

CT_VGA

```
#define CT_VGA 0
```

VGA Box.

7.143 Video pixel modes

Macros

- #define PM_RGB555 0
  RGB555 pixel mode (15-bit)
- #define PM_RGB565 1
  RGB565 pixel mode (16-bit)
- #define PM_RGB888P 2
  RBG888 packed pixel mode (24-bit)
- #define PM_RGB0888 3
  RGB0888 pixel mode (32-bit)
- #define PM_RGB888 PM_RGB0888
  Backwards compatibility support.

7.143.1 Detailed Description

This set of constants control the pixel mode that the framebuffer is set to.

7.143.2 Macro Definition Documentation

PM_RGB0888

```
#define PM_RGB0888 3
```

RGB0888 pixel mode (32-bit)

PM_RGB555

```
#define PM_RGB555 0
```

RGB555 pixel mode (15-bit)

PM_RGB565

```
#define PM_RGB565 1
```

RGB565 pixel mode (16-bit)

PM_RGB888

```
#define PM_RGB888 PM_RGB0888
```

Backwards compatibility support.

PM_RGB888P

```
#define PM_RGB888P 2
```

RBG888 packed pixel mode (24-bit)

7.144 Visual Memory Unit

VMU/VMS Maple Peripheral API.

Modules

- Clock Function

  API for features of the Clock Maple Function.
- LCD Function

  API for features of the LCD Maple Function.
- Memory Card Function

  API for features of the Memory Card Maple Function.
- Settings

  Customizable configuration data.

Files

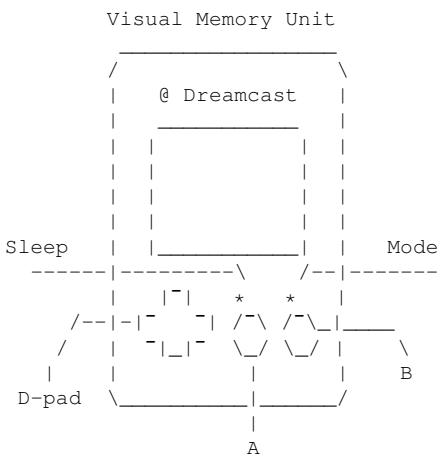
- file vmu.h

  Definitions for using the VMU device.

7.144.1 Detailed Description

VMU/VMS Maple Peripheral API.

The Sega Dreamcast's Visual Memory Unit (VMU) is an 8-Bit gaming device which, when plugged into the controller, communicates with the Dreamcast as a Maple peripheral.



As a Maple peripheral, the VMU implements the following functions:

- **MEMCARD**: Storage device used for saving and loading game files.
- **LCD**: Secondary LCD display on which additonal information may be presented to the player.
- **CLOCK**: A device which maintains the current date and time, provides at least one buzzer for playing tones, and also has buttons used for input.

Each Maple function has a corresponding set of C functions providing a high-level API around its functionality.

7.144.2 Clock Function

API for features of the Clock Maple Function.

Modules

- VMU Buttons

  VMU button masks.

Buzzer

Methods for tone generation.

- int vmu_beep_raw (maple_device_t *dev, uint32_t beep)

  Make a VMU beep (low-level).
- int vmu_beep_waveform (maple_device_t *dev, uint8_t period1, uint8_t duty_cycle1, uint8_t period2, uint8_↩ t duty_cycle2)

  Play VMU Buzzer tone.

Date/Time

Methods for managing date and time.

- int vmu_set_datetime (maple_device_t *dev, time_t unix)
  Set the date and time on the VMU.
- int vmu_get_datetime (maple_device_t *dev, time_t *unix)
  Get the date and time on the VMU.

Input

Methods for polling button states.

- void vmu_set_buttons_enabled (int enable)
  Enable/Disable polling for VMU input.
- int vmu_get_buttons_enabled (void)
  Check whether polling for VMU input has been enabled.

7.144.2.1 Detailed Description

API for features of the Clock Maple Function.

The Clock Maple function provides a high-level API for the following functionality:

- buzzer tone generation
- date/time management
- input/button status

7.144.2.2 Function Documentation

vmu_beep_raw()

```
int vmu_beep_raw (
            maple_device_t * dev,
            uint32_t beep )
```

Make a VMU beep (low-level).

This function sends a raw beep to a VMU, causing the speaker to emit a tone noise.

Note

See http://dcemulation.org/phpBB/viewtopic.php?f=29&t=97048 for the original information about beeping.

Warning

This function is submitting raw, encoded values to the VMU. For a more user-friendly API built around generating simple tones, see vmu_beep_waveform().

Parameters

dev	The device to attempt to beep.
beep	The tone to generate. Byte values are as follows: 1. period of square wave 1 2. duty cycle of square wave 1 3. period of square wave 2 (ignored by standard mono VMUs) 4. duty cycle of square wave 2 (ignored by standard mono VMUs)

Return values

MAPLE_EOK	On success.
MAPLE_EAGAIN	If the command couldn't be sent. Try again later.
MAPLE_ETIMEOUT	If the command timed out while blocking.

See also

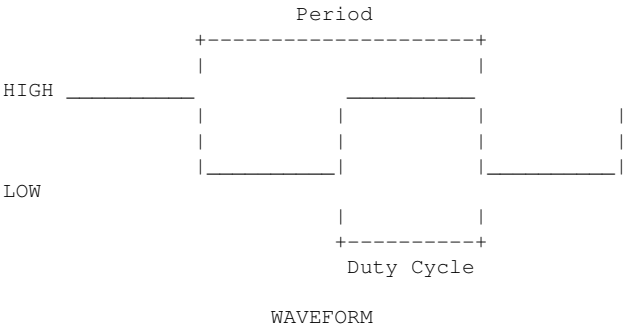
vmu_beep_waveform

vmu_beep_waveform()

```
int vmu_beep_waveform (
             maple_device_t * dev,
             uint8_t period1,
             uint8_t duty_cycle1,
             uint8_t period2,
             uint8_t duty_cycle2 )
```

Play VMU Buzzer tone.

Sends two different square waves to generate tone(s) on the VMU. Each waveform is configured as shown by the following diagram. On a standard VMU, there is only one piezoelectric buzzer, so waveform 2 is ignored; however, the parameters do support dual-channel stereo in case such a VMU ever does come along.



To stop an active tone, one can simply generate a flat wave, such as by submitting both values as 0s.

Warning

Any submitted waveform which has a duty cycle of greater than or equal to its period will result in an invalid waveform being generated and is going to mute or end the tone.

Note

Note that there are no units given for the waveform, so any 3rd party VMU is free to use any base clock rate, potentially resulting in different frequencies (or tones) being generated for the same parameters on different devices.

On the VMU-side, this tone is generated using the VMU's Timer1 peripheral as a pulse generator, which is then fed into its piezoelectric buzzer. The calculated range of the standard VMU, given its 6MHz CF clock running with a divisor of 6 is driving the Timer1 counter, is approximately 3.9KHz-500Khz; however, due to physical characteristics of the buzzer, not every frequency can be produced at a decent volume, so it's recommended that you test your values, using the KOS example found at `/examples/dreamcast/vmu/beep`.

Parameters

dev	The VMU device to play the tone on
period1	The period or total interval of the first waveform
duty_cycle1	The duty cycle or active interval of the first waveform
period2	The period or total interval of the second waveform (ignored by standard first-party VMUs).
duty_cycle2	The duty cycle or active interval of the second waveform (ignored by standard first-party VMUs).

Return values

MAPLE_EOK	On success.
MAPLE_EAGAIN	If the command couldn't be sent. Try again later.
MAPLE_ETIMEOUT	If the command timed out while blocking.

vmu_get_buttons_enabled()

```
int vmu_get_buttons_enabled (
             void  )
```

Check whether polling for VMU input has been enabled.

This function is used to check whether per-frame polling of the VMU's button states has been enabled in the driver.

Note

Polling for VMU input is disabled by default to reduce unecessary Maple BUS traffic.

See also

vmu_set_buttons_enabled

vmu_get_datetime()

```
int vmu_get_datetime (
              maple_device_t * dev,
              time_t * unix )
```

Get the date and time on the VMU.

This function gets the VMU's date and time values as a single standard C Unix timestamp.

Note

This is the VMU equivalent of calling `time(unix)`.

Parameters

dev	The device to write to.
unix	Seconds since Unix epoch (set to -1 upon failure)

Return values

MAPLE_EOK	On success.
MAPLE_ETIMEOUT	If the command timed out while blocking.
MAPLE_EFAIL	On errors other than timeout.

See also

vmu_set_datetime

vmu_set_buttons_enabled()

```
void vmu_set_buttons_enabled (
              int enable )
```

Enable/Disable polling for VMU input.

This function is used to either enable or disable polling the VMU buttons' states for input each frame.

Note

These buttons are not usually accessible to the player; however, several devices, such as the ASCII pad, the arcade pad, and the Retro Fighters controller leave the VMU partially exposed, so that these buttons remain accessible, allowing them to be used as extended controller inputs.

Polling for VMU input is disabled by default to reduce unecessary Maple BUS traffic.

See also

vmu_get_buttons_enabled

vmu_set_datetime()

```
int vmu_set_datetime (
            maple_device_t * dev,
            time_t unix )
```

Set the date and time on the VMU.

This function sets the VMU's date and time values to the given standard C Unix timestamp.

Parameters

dev	The device to write to.
unix	Seconds since Unix epoch

Return values

MAPLE_EOK	On success.
MAPLE_ETIMEOUT	If the command timed out while blocking.
MAPLE_EFAIL	On errors other than timeout.

See also

vmu_get_datetime

7.144.2.3 VMU Buttons

VMU button masks.

Data Structures

- union vmu_state_t

  VMU's "civilized" state data: 0 = RELEASED, 1 = PRESSED.

Macros

- #define VMU_DPAD_UP (0<<1)

  Up Dpad button on the VMU.
- #define VMU_DPAD_DOWN (1<<1)

  Down Dpad button on the VMU.
- #define VMU_DPAD_LEFT (2<<1)

  Left Dpad button on the VMU.
- #define VMU_DPAD_RIGHT (3<<1)

  Right Dpad button on the VMU.
- #define VMU_A (4<<1)

'A' button on the VMU

- #define VMU_B (5<<1)

  'B' button on the VMU

- #define VMU_MODE (6<<1)

  Mode button on the VMU.

- #define VMU_SLEEP (7<<1)

  Sleep button on the VMU.

Typedefs

- typedef uint8_t vmu_cond_t

  VMU's raw condition data: 0 = PRESSED, 1 = RELEASED.

7.144.2.3.1 Detailed Description

VMU button masks.

VMU's button state/cond masks, same as capability masks

Note

The MODE and SLEEP button states are not pollable on a standard VMU.

7.144.2.3.2 Macro Definition Documentation

VMU_A

```
#define VMU_A (4<<1)
```

'A' button on the VMU

VMU_B

```
#define VMU_B (5<<1)
```

'B' button on the VMU

VMU_DPAD_DOWN

```
#define VMU_DPAD_DOWN (1<<1)
```

Down Dpad button on the VMU.

VMU_DPAD_LEFT

```
#define VMU_DPAD_LEFT (2<<1)
```

Left Dpad button on the VMU.

VMU_DPAD_RIGHT

```
#define VMU_DPAD_RIGHT (3<<1)
```

Right Dpad button on the VMU.

VMU_DPAD_UP

```
#define VMU_DPAD_UP (0<<1)
```

Up Dpad button on the VMU.

VMU_MODE

```
#define VMU_MODE (6<<1)
```

Mode button on the VMU.

VMU_SLEEP

```
#define VMU_SLEEP (7<<1)
```

Sleep button on the VMU.

7.144.2.3.3 Typedef Documentation

vmu_cond_t

```
typedef uint8_t vmu_cond_t
```

VMU's raw condition data: 0 = PRESSED, 1 = RELEASED.

7.144.3 LCD Function

API for features of the LCD Maple Function.

Macros

- #define VMU_SCREEN_WIDTH 48

  Pixel width of a standard VMU screen.
- #define VMU_SCREEN_HEIGHT 32

  Pixel height of a standard VMU screen.

Functions

- int vmu_draw_lcd (maple_device_t ∗dev, const void ∗bitmap)

  Display a 1bpp bitmap on a VMU screen.
- int vmu_draw_lcd_rotated (maple_device_t ∗dev, const void ∗bitmap)

  Display a 1bpp bitmap on a VMU screen.
- int vmu_draw_lcd_xbm (maple_device_t ∗dev, const char ∗vmu_icon)

  Display a Xwindows XBM image on a VMU screen.
- void vmu_set_icon (const char ∗vmu_icon)

  Display a Xwindows XBM on all VMUs.

7.144.3.1 Detailed Description

API for features of the LCD Maple Function.

The LCD Maple function is for exposing a secondary LCD screen that gets attached to a controller, which can be used to display additional game information, or information you only want visible to a single player.

7.144.3.2 Macro Definition Documentation

VMU_SCREEN_HEIGHT

```
#define VMU_SCREEN_HEIGHT 32
```

Pixel height of a standard VMU screen.

VMU_SCREEN_WIDTH

```
#define VMU_SCREEN_WIDTH 48
```

Pixel width of a standard VMU screen.

7.144.3.3 Function Documentation

vmu_draw_lcd()

```
int vmu_draw_lcd (
            maple_device_t * dev,
            const void * bitmap )
```

Display a 1bpp bitmap on a VMU screen.

This function sends a raw bitmap to a VMU to display on its screen. This bitmap is 1bpp, and is 48x32 in size.

Parameters

dev	The device to draw to.
bitmap	The bitmap to show.

Return values

MAPLE_EOK	On success.
MAPLE_EAGAIN	If the command couldn't be sent. Try again later.
MAPLE_ETIMEOUT	If the command timed out while blocking.

See also

vmu_draw_lcd_rotated, vmu_draw_lcd_xbm, vmu_set_icon

vmu_draw_lcd_rotated()

```
int vmu_draw_lcd_rotated (
            maple_device_t * dev,
            const void * bitmap )
```

Display a 1bpp bitmap on a VMU screen.

This function sends a raw bitmap to a VMU to display on its screen. This bitmap is 1bpp, and is 48x32 in size. This function is equivalent to vmu_draw_lcd(), but the image is rotated 180° so that the first byte of the bitmap corresponds to the top-left corner, instead of the bottom-right one.

Warning

This function is optimized by an assembly routine which operates on 32 bits at a time. As such, the given bitmap must be 4-byte aligned.

Parameters

dev	The device to draw to.
bitmap	The bitmap to show.

Return values

MAPLE_EOK	On success.
MAPLE_EAGAIN	If the command couldn't be sent. Try again later.
MAPLE_ETIMEOUT	If the command timed out while blocking.

See also

vmu_draw_lcd, vmu_draw_lcd_xbm, vmu_set_icon

vmu_draw_lcd_xbm()

```
int vmu_draw_lcd_xbm (
            maple_device_t * dev,
            const char * vmu_icon )
```

Display a Xwindows XBM image on a VMU screen.

This function takes in a Xwindows XBM, converts it to a raw bitmap, and sends it to a VMU to display on its screen. This XBM image is 48x32 in size.

Parameters

dev	The device to draw to.
vmu_icon	The icon to set.

Return values

MAPLE_EOK	On success.
MAPLE_EAGAIN	If the command couldn't be sent. Try again later.
MAPLE_ETIMEOUT	If the command timed out while blocking.

See also

vmu_draw_lcd, vmu_set_icon

vmu_set_icon()

```
void vmu_set_icon (
            const char * vmu_icon )
```

Display a Xwindows XBM on all VMUs.

This function takes in a Xwindows XBM and displays the image on all VMUs.

Note

This is a convenience function for vmu_draw_lcd() to broadcast across all VMUs.

Todo Prevent this routine from broadcasting to rear VMUs.

Parameters

vmu_icon	The icon to set.

See also

vmu_draw_lcd_xbm

7.144.4 Memory Card Function

API for features of the Memory Card Maple Function.

Functions

- int vmu_block_read (maple_device_t ∗dev, uint16_t blocknum, uint8_t ∗buffer)
  Read a block from a memory card.
- int vmu_block_write (maple_device_t ∗dev, uint16_t blocknum, const uint8_t ∗buffer)
  Write a block to a memory card.

7.144.4.1 Detailed Description

API for features of the Memory Card Maple Function.

The Memory Card Maple function is for exposing a low-level, block-based API that allows you to read from and write to random blocks within the memory card's filesytem.

Note

A standard memory card has a block size of 512 bytes; however, the block size is a configurable parameter in the "root" block, which can be queried to cover supporting homebrew memory cards with larger block sizes.

Warning

You should never use these functions directly, unless you *really* know what you're doing, as you can easily corrupt the filesystem by writing incorrect data. Instead, you should favor the high-level filesystem API found in vmufs.h, or just use the native C standard filesystem API within the virtual `/vmu/` root directory to operate on VMU data.

7.144.4.2 Function Documentation

vmu_block_read()

```
int vmu_block_read (
            maple_device_t * dev,
            uint16_t blocknum,
            uint8_t * buffer )
```

Read a block from a memory card.

This function performs a low-level raw block read from a memory card.

Parameters

dev	The device to read from.
blocknum	The block number to read.
buffer	The buffer to read into (512 bytes).

Return values

MAPLE_EOK	On success.
MAPLE_ETIMEOUT	If the command timed out while blocking.
MAPLE_EFAIL	On errors other than timeout.

See also

vmu_block_write

vmu_block_write()

```
int vmu_block_write (
            maple_device_t * dev,
            uint16_t blocknum,
            const uint8_t * buffer )
```

Write a block to a memory card.

This function performs a low-level raw block write to a memory card.

Parameters

dev	The device to write to.
blocknum	The block number to write.
buffer	The buffer to write from (512 bytes).

Return values

MAPLE_EOK	On success.
MAPLE_ETIMEOUT	If the command timed out while blocking.
MAPLE_EFAIL	On errors other than timeout.

See also

vmu_block_read

7.144.5 Settings

Customizable configuration data.

Functions

- int vmu_has_241_blocks (maple_device_t *dev)

  Get the status of a VMUs extra 41 blocks.
- int vmu_toggle_241_blocks (maple_device_t *dev, int enable)

  Enable the extra 41 blocks of a VMU.
- int vmu_use_custom_color (maple_device_t *dev, int enable)

  Enable custom color of a VMU.
- int vmu_set_custom_color (maple_device_t *dev, uint8_t red, uint8_t green, uint8_t blue, uint8_t alpha)

  Set custom color of a VMU.
- int vmu_get_custom_color (maple_device_t *dev, uint8_t *red, uint8_t *green, uint8_t *blue, uint8_t *alpha)

  Get custom color of a VMU.
- int vmu_set_icon_shape (maple_device_t *dev, uint8_t icon_shape)

  Set icon shape of a VMU.
- int vmu_get_icon_shape (maple_device_t *dev, uint8_t *icon_shape)

  Get icon shape of a VMU.

7.144.5.1 Detailed Description

Customizable configuration data.

This module provides a high-level abstraction around various features and settings which can be modified on the VMU. Many of these operations are provided by the Dreamcast's BIOS when a VMU has been formatted.

7.144.5.2 Function Documentation

vmu_get_custom_color()

```
int vmu_get_custom_color (
            maple_device_t * dev,
            uint8_t * red,
            uint8_t * green,
            uint8_t * blue,
            uint8_t * alpha )
```

Get custom color of a VMU.

This function gets the custom color of a specific VMU. This color is only displayed in the Dreamcast's file manager. This function also returns whether the custom color is currently enabled.

Parameters

dev	The device to change the color of.
red	The red component. 0-255
green	The green component. 0-255
blue	The blue component. 0-255
alpha	The alpha component. 0-255; 100-255 Recommended

Return values

1	On success: custom color is enabled
0	On success: custom color is disabled
-1	On failure

See also

vmu_set_custom_color, vmu_use_custom_color

vmu_get_icon_shape()

```
int vmu_get_icon_shape (
            maple_device_t * dev,
            uint8_t * icon_shape )
```

Get icon shape of a VMU.

This function gets the icon shape of a specific VMU. The icon shape is a VMU icon that is displayed on the LCD screen while navigating the Dreamcast BIOS menu and is the GUI representation of the VMU in the menu's file manager. The Dreamcast BIOS provides a set of 124 icons to choose from.

Note

When a custom file named "ICONDATA_VMS" is present on a VMU, it overrides this icon by providing custom icons for both the DC BIOS menu and the VMU's LCD screen.

Parameters

dev	The device to change the icon shape of.
icon_shape	One of the values found in Builtin VMU Icons.

Return values

0	On success
-1	On failure

See also

Builtin VMU Icons, vmu_set_icon_shape

vmu_has_241_blocks()

```
int vmu_has_241_blocks (
            maple_device_t * dev )
```

Get the status of a VMUs extra 41 blocks.

This function checks if the extra 41 blocks of a VMU have been enabled.

Parameters

dev	The device to check the status of.

Return values

1	On success: extra blocks are enabled
0	On success: extra blocks are disabled
-1	On failure

vmu_set_custom_color()

```
int vmu_set_custom_color (
            maple_device_t * dev,
            uint8_t red,
            uint8_t green,
            uint8_t blue,
            uint8_t alpha )
```

Set custom color of a VMU.

This function sets the custom color of a specific VMU. This color is only displayed in the Dreamcast's file manager. This function also enables the use of the custom color. Otherwise it wouldn't show up.

Parameters

dev	The device to change the color of.
red	The red component. 0-255
green	The green component. 0-255
blue	The blue component. 0-255
alpha	The alpha component. 0-255; 100-255 Recommended

Return values

0	On success
-1	On failure

See also

vmu_get_custom_color, vmu_use_custom_color

vmu_set_icon_shape()

```
int vmu_set_icon_shape (
            maple_device_t * dev,
            uint8_t icon_shape )
```

Set icon shape of a VMU.

This function sets the icon shape of a specific VMU. The icon shape is a VMU icon that is displayed on the LCD screen while navigating the Dreamcast BIOS menu and is the GUI representation of the VMU in the menu's file manager. The Dreamcast BIOS provides a set of 124 icons to choose from.

Note

When a custom file named "ICONDATA_VMS" is present on a VMU, it overrides this icon by providing custom icons for both the DC BIOS menu and the VMU's LCD screen.

Parameters

dev	The device to change the icon shape of.
icon_shape	One of the values found in Builtin VMU Icons.

Return values

0	On success
-1	On failure

See also

Builtin VMU Icons, vmu_get_icon_shape

vmu_toggle_241_blocks()

```
int vmu_toggle_241_blocks (
            maple_device_t * dev,
            int enable )
```

Enable the extra 41 blocks of a VMU.

This function enables/disables the extra 41 blocks of a specific VMU.

Warning

Enabling the extra blocks of a VMU may render it unusable for a very few commercial games.

Parameters

dev	The device to enable/disable 41 blocks.
enable	Values other than 0 enables. Equal to 0 disables.

Return values

0	On success
-1	On failure

vmu_use_custom_color()

```
int vmu_use_custom_color (
            maple_device_t * dev,
            int enable )
```

Enable custom color of a VMU.

This function enables/disables the custom color of a specific VMU. This color is only displayed in the Dreamcast's file manager.

Parameters

dev	The device to enable/disable custom color.
enable	Values other than 0 enables. Equal to 0 disables.

Return values

0	On success
-1	On failure

See also

vmu_set_custom_color

8 Data Structure Documentation

8.1 __newlib_recursive_lock_t Struct Reference

```
#include <lock.h>
```

Data Fields

- void * owner
- int nest
- volatile int lock

8.1.1 Field Documentation

lock

```
volatile int __newlib_recursive_lock_t::lock
```

nest

```
int __newlib_recursive_lock_t::nest
```

owner

```
void* __newlib_recursive_lock_t::owner
```

The documentation for this struct was generated from the following file:

- include/sys/lock.h

8.2 _mbstate_t Struct Reference

Conversion state information.

```
#include <sys/_types.h>
```

Data Fields

- int __count
- union {
  wint_t __wch
  unsigned char __wchb [4]
  } __value

8.2.1 Detailed Description

Conversion state information.

8.2.2 Field Documentation

__count

```
int _mbstate_t::__count
```

[union]

```
union { ... } _mbstate_t::__value
```

__wch

```
wint_t _mbstate_t::__wch
```

__wchb

```
unsigned char _mbstate_t::__wchb[4]
```

The documentation for this struct was generated from the following file:

- include/sys/_types.h

8.3 addrinfo Struct Reference

Network address information structure.

```
#include <netdb.h>
```

Data Fields

- int ai_flags
  Input flags.
- int ai_family
  Socket address family.
- int ai_socktype
  Socket type.
- int ai_protocol
  Socket protocol.
- socklen_t ai_addrlen
  Address length.
- struct sockaddr * ai_addr
  Address structure.
- char * ai_canonname
  Canonical name.
- struct addrinfo * ai_next
  Next address entry (if any).

8.3.1 Detailed Description

Network address information structure.

This structure describes information on an address in the database. This structure is used by functions such as getaddrinfo() to return information about the specified host.

8.3.2 Field Documentation

ai_addr

```
struct sockaddr* addrinfo::ai_addr
```

Address structure.

ai_addrlen

```
socklen_t addrinfo::ai_addrlen
```

Address length.

ai_canonname

```
char* addrinfo::ai_canonname
```

Canonical name.

ai_family

```
int addrinfo::ai_family
```

Socket address family.

ai_flags

```
int addrinfo::ai_flags
```

Input flags.

See also

Flags for ai_flags in struct addrinfo

ai_next

```
struct addrinfo* addrinfo::ai_next
```

Next address entry (if any).

ai_protocol

```
int addrinfo::ai_protocol
```

Socket protocol.

ai_socktype

```
int addrinfo::ai_socktype
```

Socket type.

The documentation for this struct was generated from the following file:

- include/netdb.h

8.4 aica_channel_t Struct Reference

```
#include <aica_comm.h>
```

Data Fields

- uint32 cmd
- uint32 base
- uint32 type
- uint32 length
- uint32 loop
- uint32 loopstart
- uint32 loopend
- uint32 freq
- uint32 vol
- uint32 pan
- uint32 pos
- uint32 pad [5]

8.4.1 Field Documentation

base

```
uint32 aica_channel_t::base
```

cmd

```
uint32 aica_channel_t::cmd
```

freq

```
uint32 aica_channel_t::freq
```

length

```
uint32 aica_channel_t::length
```

loop

```
uint32 aica_channel_t::loop
```

loopend

```
uint32 aica_channel_t::loopend
```

loopstart

```
uint32 aica_channel_t::loopstart
```

pad

```
uint32 aica_channel_t::pad[5]
```

pan

```
uint32 aica_channel_t::pan
```

pos

```
uint32 aica_channel_t::pos
```

type

```
uint32 aica_channel_t::type
```

vol

```
uint32 aica_channel_t::vol
```

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/sound/aica_comm.h

8.5 aica_cmd_t Struct Reference

```
#include <aica_comm.h>
```

Data Fields

- uint32 size
- uint32 cmd
- uint32 timestamp
- uint32 cmd_id
- uint32 misc [4]
- uint8 cmd_data [ ]

8.5.1 Field Documentation

cmd

```
uint32 aica_cmd_t::cmd
```

cmd_data

```
uint8 aica_cmd_t::cmd_data[]
```

cmd_id

```
uint32 aica_cmd_t::cmd_id
```

misc

```
uint32 aica_cmd_t::misc[4]
```

size

```
uint32 aica_cmd_t::size
```

timestamp

```
uint32 aica_cmd_t::timestamp
```

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/sound/aica_comm.h

8.6 aica_queue_t Struct Reference

```
#include <aica_comm.h>
```

Data Fields

- uint32 head
- uint32 tail
- uint32 size
- uint32 valid
- uint32 process_ok
- uint32 data

8.6.1 Field Documentation

data

```
uint32 aica_queue_t::data
```

head

```
uint32 aica_queue_t::head
```

process_ok

```
uint32 aica_queue_t::process_ok
```

size

```
uint32 aica_queue_t::size
```

tail

```
uint32 aica_queue_t::tail
```

valid

```
uint32 aica_queue_t::valid
```

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/sound/aica_comm.h

8.7 CDROM_TOC Struct Reference

TOC structure returned by the BIOS.

```
#include <dc/cdrom.h>
```

Data Fields

- uint32 entry [99]
  TOC space for 99 tracks.
- uint32 first
  Point A0 information (1st track)
- uint32 last
  Point A1 information (last track)
- uint32 leadout_sector
  Point A2 information (leadout)

8.7.1 Detailed Description

TOC structure returned by the BIOS.

This is the structure that the CMD_GETTOC2 syscall command will return for the TOC. Note the data is in FAD, not LBA/LSN.

8.7.2 Field Documentation

entry

```
uint32 CDROM_TOC::entry[99]
```

TOC space for 99 tracks.

first

```
uint32 CDROM_TOC::first
```

Point A0 information (1st track)

last

```
uint32 CDROM_TOC::last
```

Point A1 information (last track)

leadout_sector

```
uint32 CDROM_TOC::leadout_sector
```

Point A2 information (leadout)

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/cdrom.h

8.8 condvar_t Struct Reference

Condition variable.

```
#include <kos/cond.h>
```

Data Fields

- int dummy
- int dynamic

8.8.1 Detailed Description

Condition variable.

There are no public members of this structure for you to actually do anything with in your code, so don't try.

8.8.2 Field Documentation

dummy

```
int condvar_t::dummy
```

dynamic

```
int condvar_t::dynamic
```

The documentation for this struct was generated from the following file:

- include/kos/cond.h

8.9 cont_state_t Struct Reference

Controller state structure.

```
#include <controller.h>
```

Data Fields

- union {
  uint32_t buttons
  bit-packed controller button states
  struct {
  uint32_t c: 1
  C button value.
  uint32_t b: 1
  B button value.
  uint32_t a: 1
  A button value.
  uint32_t start: 1
  Start button value.
  uint32_t dpad_up: 1
  Main Dpad Up button value.
  uint32_t dpad_down: 1
  Main Dpad Down button value.
  uint32_t dpad_left: 1
  Main Dpad Left button value.

```
        uint32_t dpad_right: 1
            Main Dpad Right button value.
        uint32_t z: 1
            Z button value.
        uint32_t y: 1
            Y button value.
        uint32_t x: 1
            X button value.
        uint32_t d: 1
            D button value.
        uint32_t dpad2_up: 1
            Secondary Dpad Up button value.
        uint32_t dpad2_down: 1
            Secondary Dpad Down button value.
        uint32_t dpad2_left: 1
            Secondary Dpad Left button value.
        uint32_t dpad2_right: 1
            Secondary Dpad Right button value.
        uint32_t : 16
    }
};
```

- int ltrig

  Left trigger value (0-255).
- int rtrig

  Right trigger value (0-255).
- int joyx

  Main joystick x-axis value (-128 - 127).
- int joyy

  Main joystick y-axis value.
- int joy2x

  Secondary joystick x-axis value.
- int joy2y

  Secondary joystick y-axis value.

8.9.1 Detailed Description

Controller state structure.

This structure contains information about the status of the controller device and can be fetched by casting the result of maple_dev_status() to this structure.

A 1 bit in the buttons' bitfield indicates that a button is pressed, and the joyx, joyy, joy2x, joy2 values are all 0 based (0 is centered).

Note

Whether a particular field or button is actually used by the controller depends upon its capabilities. See Querying Capabilities.

See also

maple_dev_status

8.9.2 Field Documentation

[union]

```
union { ... } cont_state_t
```

__pad0__

```
uint32_t cont_state_t::__pad0__
```

a

```
uint32_t cont_state_t::a
```

A button value.

b

```
uint32_t cont_state_t::b
```

B button value.

buttons

```
uint32_t cont_state_t::buttons
```

bit-packed controller button states

See also

controller_buttons

c

```
uint32_t cont_state_t::c
```

C button value.

d

```
uint32_t cont_state_t::d
```

D button value.

dpad2_down

```
uint32_t cont_state_t::dpad2_down
```

Secondary Dpad Down button value.

dpad2_left

```
uint32_t cont_state_t::dpad2_left
```

Secondary Dpad Left button value.

dpad2_right

```
uint32_t cont_state_t::dpad2_right
```

Secondary Dpad Right button value.

dpad2_up

```
uint32_t cont_state_t::dpad2_up
```

Secondary Dpad Up button value.

dpad_down

```
uint32_t cont_state_t::dpad_down
```

Main Dpad Down button value.

dpad_left

```
uint32_t cont_state_t::dpad_left
```

Main Dpad Left button value.

dpad_right

```
uint32_t cont_state_t::dpad_right
```

Main Dpad Right button value.

dpad_up

```
uint32_t cont_state_t::dpad_up
```

Main Dpad Up button value.

joy2x

```
int cont_state_t::joy2x
```

Secondary joystick x-axis value.

joy2y

```
int cont_state_t::joy2y
```

Secondary joystick y-axis value.

joyx

```
int cont_state_t::joyx
```

Main joystick x-axis value (-128 - 127).

joyy

```
int cont_state_t::joyy
```

Main joystick y-axis value.

ltrig

```
int cont_state_t::ltrig
```

Left trigger value (0-255).

rtrig

```
int cont_state_t::rtrig
```

Right trigger value (0-255).

start

```
uint32_t cont_state_t::start
```

Start button value.

x

```
uint32_t cont_state_t::x
```

X button value.

y

```
uint32_t cont_state_t::y
```

Y button value.

z

```
uint32_t cont_state_t::z
```

Z button value.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple/controller.h

8.10 dbgio_handler_t Struct Reference

Debug I/O Interface.

```
#include <kos/dbgio.h>
```

Data Fields

- const char * name

  Name of the dbgio handler.
- int(* detected)(void)

  Detect this debug interface.
- int(* init)(void)

  Initialize this debug interface with default parameters.
- int(* shutdown)(void)

  Shutdown this debug interface.
- int(* set_irq_usage)(int mode)

  Set either polled or IRQ usage for this interface.
- int(* read)(void)

  Read one character from the console.
- int(* write)(int c)

  Write one character to the console.
- int(* flush)(void)

  Flush any queued output.
- int(* write_buffer)(const uint8 *data, int len, int xlat)

  Write an entire buffer of data to the console.
- int(* read_buffer)(uint8 *data, int len)

  Read an entire buffer of data from the console.

8.10.1 Detailed Description

Debug I/O Interface.

This struct represents a single dbgio interface. This should represent a generic pollable console interface. We will store an ordered list of these statically linked into the program and fall back from one to the next until one returns true for detected(). Note that the last device in this chain is the null console, which will always return true.

8.10.2 Field Documentation

detected

```
int(* dbgio_handler_t::detected) (void)
```

Detect this debug interface.

Return values

1	If the device is available and useable
0	If the device is unavailable

flush

```
int(* dbgio_handler_t::flush) (void)
```

Flush any queued output.

Return values

0	On success
-1	On error (set errno as appropriate)

init

```
int(* dbgio_handler_t::init) (void)
```

Initialize this debug interface with default parameters.

Return values

0	On success
-1	On failure

name

```
const char* dbgio_handler_t::name
```

Name of the dbgio handler.

read

```
int(* dbgio_handler_t::read) (void)
```

Read one character from the console.

Return values

0	On success
-1	On failure (set errno as appropriate)

read_buffer

```
int(* dbgio_handler_t::read_buffer) (uint8 *data, int len)
```

Read an entire buffer of data from the console.

Parameters

data	The buffer to read into
len	The length of the buffer

Returns

Number of characters read on success, or -1 on failure (set errno as appropriate)

set_irq_usage

```
int(* dbgio_handler_t::set_irq_usage) (int mode)
```

Set either polled or IRQ usage for this interface.

Parameters

mode	1 for IRQ-based usage, 0 for polled I/O

Return values

0	On success
-1	On failure

shutdown

```
int(* dbgio_handler_t::shutdown) (void)
```

Shutdown this debug interface.

Return values

0	On success
-1	On failure

write

```
int(* dbgio_handler_t::write) (int c)
```

Write one character to the console.

Parameters

c	The character to write

Return values

1	On success
-1	On error (set errno as appropriate)

Note

Interfaces may require a call to flush() before the output is actually flushed to the console.

write_buffer

```
int(* dbgio_handler_t::write_buffer) (const uint8 *data, int len, int xlat)
```

Write an entire buffer of data to the console.

Parameters

data	The buffer to write
len	The length of the buffer
xlat	If non-zero, newline transformations may occur

Returns

Number of characters written on success, or -1 on failure (set errno as appropriate)

The documentation for this struct was generated from the following file:

- include/kos/dbgio.h

8.11 DIR Struct Reference

Type representing a directory stream.

```
#include <sys/dirent.h>
```

Data Fields

- file_t fd

  File descriptor for the directory.
- struct dirent d_ent

  Current directory entry.

8.11.1 Detailed Description

Type representing a directory stream.

This type represents a directory stream and is used by the directory reading functions to trace their position in the directory.

The values in this function are all private and subject to change. Do not attempt to use any of them directly.

8.11.2 Field Documentation

d_ent

```
struct dirent DIR::d_ent
```

Current directory entry.

fd

```
file_t DIR::fd
```

File descriptor for the directory.

The documentation for this struct was generated from the following file:

- include/sys/dirent.h

8.12 dirent Struct Reference

POSIX directory entry structure.

```
#include <sys/dirent.h>
```

Data Fields

- int d_ino
  File unique identifier.
- off_t d_off
  File offset.
- uint16 d_reclen
  Record length.
- uint8 d_type
  File type.
- char d_name [256]
  Filename.

8.12.1 Detailed Description

POSIX directory entry structure.

This structure contains information about a single entry in a directory in the VFS.

8.12.2 Field Documentation

d_ino

```
int dirent::d_ino
```

File unique identifier.

d_name

```
char dirent::d_name[256]
```

Filename.

d_off

```
off_t dirent::d_off
```

File offset.

d_reclen

```
uint16 dirent::d_reclen
```

Record length.

d_type

```
uint8 dirent::d_type
```

File type.

The documentation for this struct was generated from the following file:

- include/sys/dirent.h

8.13 dirent_t Struct Reference

Directory entry.

`#include <kos/fs.h>`

Data Fields

- int size
  Size of the file in bytes.
- char name [NAME_MAX]
  Name of the file.
- time_t time
  Last access/mod/change time (depends on VFS)
- uint32 attr
  Attributes of the file.

8.13.1 Detailed Description

Directory entry.

All VFS handlers must conform to this interface in their directory entries.

8.13.2 Field Documentation

attr

```
uint32 dirent_t::attr
```

Attributes of the file.

name

```
char dirent_t::name[NAME_MAX]
```

Name of the file.

size

```
int dirent_t::size
```

Size of the file in bytes.

time

`time_t dirent_t::time`

Last access/mod/change time (depends on VFS)

The documentation for this struct was generated from the following file:

- include/kos/fs.h

8.14 dreameye_state_t Struct Reference

Dreameye status structure.

`#include <dc/maple/dreameye.h>`

Data Fields

- int image_count
  The number of images on the device.
- int image_count_valid
  Is the image_count field valid?
- int transfer_count
  The number of transfer operations required for the selected image.
- int img_transferring
  Is an image transferring now?
- uint8 * img_buf
  Storage for image data.
- int img_size
  The size of the image in bytes.
- uint8 img_number
  The image number currently being transferred.

8.14.1 Detailed Description

Dreameye status structure.

This structure contains information about the status of the Camera device and can be fetched with maple_dev_status(). You should not change any of this information, it should all be considered read-only. Most of the fields in here are related to image transfers, and messing with them during a transfer could screw things up.

8.14.2 Field Documentation

image_count

```
int dreameye_state_t::image_count
```

The number of images on the device.

image_count_valid

```
int dreameye_state_t::image_count_valid
```

Is the image_count field valid?

img_buf

```
uint8* dreameye_state_t::img_buf
```

Storage for image data.

img_number

```
uint8 dreameye_state_t::img_number
```

The image number currently being transferred.

img_size

```
int dreameye_state_t::img_size
```

The size of the image in bytes.

img_transferring

```
int dreameye_state_t::img_transferring
```

Is an image transferring now?

transfer_count

```
int dreameye_state_t::transfer_count
```

The number of transfer operations required for the selected image.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple/dreameye.h

8.15 elf_hdr_t Struct Reference

ELF file header.

```
#include <kos/elf.h>
```

Data Fields

- uint8 ident [16]
  ELF identifier.
- uint16 type
  ELF file type.
- uint16 machine
  ELF file architecture.
- uint32 version
  Object file version.
- uint32 entry
  Entry point.
- uint32 phoff
  Program header offset.
- uint32 shoff
  Section header offset.
- uint32 flags
  Processor flags.
- uint16 ehsize
  ELF header size in bytes.
- uint16 phentsize
  Program header entry size.
- uint16 phnum
  Program header entry count.
- uint16 shentsize
  Section header entry size.
- uint16 shnum
  Section header entry count.
- uint16 shstrndx
  String table section index.

8.15.1 Detailed Description

ELF file header.

This header is at the beginning of any valid ELF binary and serves to identify the architecture of the binary and various data about it.

8.15.2 Field Documentation

ehsize

```
uint16 elf_hdr_t::ehsize
```

ELF header size in bytes.

entry

```
uint32 elf_hdr_t::entry
```

Entry point.

flags

```
uint32 elf_hdr_t::flags
```

Processor flags.

ident

```
uint8 elf_hdr_t::ident[16]
```

ELF identifier.

machine

```
uint16 elf_hdr_t::machine
```

ELF file architecture.

phentsize

```
uint16 elf_hdr_t::phentsize
```

Program header entry size.

phnum

```
uint16 elf_hdr_t::phnum
```

Program header entry count.

phoff

```
uint32 elf_hdr_t::phoff
```

Program header offset.

shentsize

```
uint16 elf_hdr_t::shentsize
```

Section header entry size.

shnum

```
uint16 elf_hdr_t::shnum
```

Section header entry count.

shoff

```
uint32 elf_hdr_t::shoff
```

Section header offset.

shstrndx

```
uint16 elf_hdr_t::shstrndx
```

String table section index.

type

```
uint16 elf_hdr_t::type
```

ELF file type.

version

```
uint32 elf_hdr_t::version
```

Object file version.

The documentation for this struct was generated from the following file:

- include/kos/elf.h

8.16 elf_prog_t Struct Reference

Kernel-specific definition of a loaded ELF binary.

```
#include <kos/elf.h>
```

Data Fields

- void * data
  Pointer to program in memory.
- uint32 size
  Memory image size (rounded up to page size)
- ptr_t lib_get_name
  Pointer to get_name() function.
- ptr_t lib_get_version
  Pointer to get_version() function.
- ptr_t lib_open
  Pointer to library's open function.
- ptr_t lib_close
  Pointer to library's close function.
- char fn [256]
  Filename of library.

8.16.1 Detailed Description

Kernel-specific definition of a loaded ELF binary.

This structure represents the internal representation of a loaded ELF binary in KallistiOS (specifically as a dynamically loaded library).

8.16.2 Field Documentation

data

```
void* elf_prog_t::data
```

Pointer to program in memory.

fn

```
char elf_prog_t::fn[256]
```

Filename of library.

lib_close

```
ptr_t elf_prog_t::lib_close
```

Pointer to library's close function.

lib_get_name

```
ptr_t elf_prog_t::lib_get_name
```

Pointer to get_name() function.

lib_get_version

```
ptr_t elf_prog_t::lib_get_version
```

Pointer to get_version() function.

lib_open

```
ptr_t elf_prog_t::lib_open
```

Pointer to library's open function.

size

```
uint32 elf_prog_t::size
```

Memory image size (rounded up to page size)

The documentation for this struct was generated from the following file:

- include/kos/elf.h

8.17 elf_rel_t Struct Reference

ELF Relocation entry (without explicit addend).

`#include <kos/elf.h>`

Data Fields

- uint32 offset

  Offset within section.
- uint32 info

  Symbol and type.

8.17.1 Detailed Description

ELF Relocation entry (without explicit addend).

This structure represents an ELF relocation entry without an explicit addend. This structure is used on some architectures, whereas others use the elf_rela_t structure instead.

8.17.2 Field Documentation

info

`uint32 elf_rel_t::info`

Symbol and type.

offset

`uint32 elf_rel_t::offset`

Offset within section.

The documentation for this struct was generated from the following file:

- include/kos/elf.h

8.18 elf_rela_t Struct Reference

ELF Relocation entry (with explicit addend).

`#include <kos/elf.h>`

Data Fields

- uint32 offset

  Offset within section.
- uint32 info

  Symbol and type.
- int32 addend

  Constant addend for the symbol.

8.18.1 Detailed Description

ELF Relocation entry (with explicit addend).

This structure represents an ELF relocation entry with an explicit addend. This structure is used on some architectures, whereas others use the elf_rel_t structure instead.

8.18.2 Field Documentation

addend

```
int32 elf_rela_t::addend
```

Constant addend for the symbol.

info

```
uint32 elf_rela_t::info
```

Symbol and type.

offset

```
uint32 elf_rela_t::offset
```

Offset within section.

The documentation for this struct was generated from the following file:

- include/kos/elf.h

8.19 elf_shdr_t Struct Reference

ELF Section header.

```
#include <kos/elf.h>
```

Data Fields

- uint32 name
  Index into string table.
- uint32 type
  Section type.
- uint32 flags
  Section flags.
- uint32 addr
  In-memory offset.
- uint32 offset
  On-disk offset.
- uint32 size
  Size (if SHT_NOBITS, amount of 0s needed)
- uint32 link
  Section header table index link.
- uint32 info
  Section header extra info.
- uint32 addralign
  Alignment constraints.
- uint32 entsize
  Fixed-size table entry sizes.

8.19.1 Detailed Description

ELF Section header.

This structure represents the header on each ELF section.

8.19.2 Field Documentation

addr

```
uint32 elf_shdr_t::addr
```

In-memory offset.

addralign

```
uint32 elf_shdr_t::addralign
```

Alignment constraints.

entsize

`uint32 elf_shdr_t::entsize`

Fixed-size table entry sizes.

flags

`uint32 elf_shdr_t::flags`

Section flags.

See also

Section header flags

info

`uint32 elf_shdr_t::info`

Section header extra info.

link

`uint32 elf_shdr_t::link`

Section header table index link.

name

`uint32 elf_shdr_t::name`

Index into string table.

offset

`uint32 elf_shdr_t::offset`

On-disk offset.

size

`uint32 elf_shdr_t::size`

Size (if SHT_NOBITS, amount of 0s needed)

type

`uint32 elf_shdr_t::type`

Section type.

See also

Section header types

The documentation for this struct was generated from the following file:

- include/kos/elf.h

8.20 elf_sym_t Struct Reference

Symbol table entry.

`#include <kos/elf.h>`

Data Fields

- uint32 name
  Index into file's string table.
- uint32 value
  Value of the symbol.
- uint32 size
  Size of the symbol.
- uint8 info
  Symbol type and binding.
- uint8 other
  0. Holds no meaning.
- uint16 shndx
  Section index.

8.20.1 Detailed Description

Symbol table entry.

This structure represents a single entry in a symbol table in an ELF file.

8.20.2 Field Documentation

info

```
uint8 elf_sym_t::info
```

Symbol type and binding.

name

```
uint32 elf_sym_t::name
```

Index into file's string table.

other

```
uint8 elf_sym_t::other
```

0. Holds no meaning.

shndx

```
uint16 elf_sym_t::shndx
```

Section index.

size

```
uint32 elf_sym_t::size
```

Size of the symbol.

value

```
uint32 elf_sym_t::value
```

Value of the symbol.

The documentation for this struct was generated from the following file:

- include/kos/elf.h

8.21 export_sym_t Struct Reference

A single export symbol.

`#include <kos/exports.h>`

Data Fields

- const char * name
  The name of the symbol.
- ptr_t ptr
  A pointer to the symbol.

8.21.1 Detailed Description

A single export symbol.

This structure holds a single symbol that has been exported from the kernel. These will be patched into loaded ELF binaries at load time.

8.21.2 Field Documentation

name

```
const char* export_sym_t::name
```

The name of the symbol.

ptr

```
ptr_t export_sym_t::ptr
```

A pointer to the symbol.

The documentation for this struct was generated from the following file:

- include/kos/exports.h

8.22 fd_set Struct Reference

`#include <select.h>`

Data Fields

- unsigned long fds_bits [FD_SETSIZE/NFDBITS]

8.22.1 Field Documentation

fds_bits

```
unsigned long fd_set::fds_bits[FD_SETSIZE/NFDBITS]
```

The documentation for this struct was generated from the following file:

- include/sys/select.h

8.23 flashrom_ispcfg_t Struct Reference

ISP configuration structure.

```
#include <dc/flashrom.h>
```

Data Fields

- int method
  DHCP, Static, dialup(?), PPPoE.
- uint32 valid_fields
  Which fields are valid?
- uint32 flags
  Various flags that can be set in options.
- uint8 ip [4]
  Host IP address.
- uint8 nm [4]
  Netmask.
- uint8 bc [4]
  Broadcast address.
- uint8 gw [4]
  Gateway address.
- uint8 dns [2][4]
  DNS servers (2)
- int proxy_port
  Proxy server port.
- char hostname [24]
  DHCP/Host name.
- char email [64]
  Email address.
- char smtp [31]

SMTP server.

- char pop3 [31]

  POP3 server.
- char pop3_login [20]

  POP3 login.
- char pop3_passwd [32]

  POP3 passwd.
- char proxy_host [31]

  Proxy server hostname.
- char ppp_login [29]

  PPP login.
- char ppp_passwd [20]

  PPP password.
- char out_prefix [9]

  Outside dial prefix.
- char cw_prefix [9]

  Call waiting prefix.
- char real_name [31]

  The "Real Name" field of PlanetWeb.
- char modem_init [33]

  The modem init string to use.
- char area_code [4]

  The area code the user is in.
- char ld_prefix [21]

  The long-distance dial prefix.
- char p1_areacode [4]

  Phone number 1's area code.
- char phone1 [26]

  Phone number 1.
- char p2_areacode [4]

  Phone number 2's area code.
- char phone2 [26]

  Phone number 2.

8.23.1 Detailed Description

ISP configuration structure.

This structure will be filled in by flashrom_get_ispcfg() (DreamPassport) or flashrom_get_pw_ispcfg() (PlanetWeb). Thanks to Sam Steele for the information about DreamPassport's ISP settings.

8.23.2 Field Documentation

area_code

```
char flashrom_ispcfg_t::area_code[4]
```

The area code the user is in.

bc

```
uint8 flashrom_ispcfg_t::bc[4]
```

Broadcast address.

cw_prefix

```
char flashrom_ispcfg_t::cw_prefix[9]
```

Call waiting prefix.

dns

```
uint8 flashrom_ispcfg_t::dns[2][4]
```

DNS servers (2)

email

```
char flashrom_ispcfg_t::email[64]
```

Email address.

flags

```
uint32 flashrom_ispcfg_t::flags
```

Various flags that can be set in options.

See also

Flags for the flashrom_ispcfg_t struct

gw

```
uint8 flashrom_ispcfg_t::gw[4]
```

Gateway address.

hostname

```
char flashrom_ispcfg_t::hostname[24]
```

DHCP/Host name.

ip

```
uint8 flashrom_ispcfg_t::ip[4]
```

Host IP address.

ld_prefix

```
char flashrom_ispcfg_t::ld_prefix[21]
```

The long-distance dial prefix.

method

```
int flashrom_ispcfg_t::method
```

DHCP, Static, dialup(?), PPPoE.

See also

Connection method types

modem_init

```
char flashrom_ispcfg_t::modem_init[33]
```

The modem init string to use.

nm

```
uint8 flashrom_ispcfg_t::nm[4]
```

Netmask.

out_prefix

```
char flashrom_ispcfg_t::out_prefix[9]
```

Outside dial prefix.

p1_areacode

```
char flashrom_ispcfg_t::p1_areacode[4]
```

Phone number 1's area code.

p2_areacode

```
char flashrom_ispcfg_t::p2_areacode[4]
```

Phone number 2's area code.

phone1

```
char flashrom_ispcfg_t::phone1[26]
```

Phone number 1.

phone2

```
char flashrom_ispcfg_t::phone2[26]
```

Phone number 2.

pop3

```
char flashrom_ispcfg_t::pop3[31]
```

POP3 server.

pop3_login

```
char flashrom_ispcfg_t::pop3_login[20]
```

POP3 login.

pop3_passwd

```
char flashrom_ispcfg_t::pop3_passwd[32]
```

POP3 passwd.

ppp_login

```
char flashrom_ispcfg_t::ppp_login[29]
```

PPP login.

ppp_passwd

```
char flashrom_ispcfg_t::ppp_passwd[20]
```

PPP password.

proxy_host

```
char flashrom_ispcfg_t::proxy_host[31]
```

Proxy server hostname.

proxy_port

```
int flashrom_ispcfg_t::proxy_port
```

Proxy server port.

real_name

```
char flashrom_ispcfg_t::real_name[31]
```

The "Real Name" field of PlanetWeb.

smtp

```
char flashrom_ispcfg_t::smtp[31]
```

SMTP server.

valid_fields

```
uint32 flashrom_ispcfg_t::valid_fields
```

Which fields are valid?

See also

Valid field constants for the flashrom_ispcfg_t struct

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/flashrom.h

8.24 flashrom_syscfg_t Struct Reference

System configuration structure.

```
#include <dc/flashrom.h>
```

Data Fields

- int language
  Language setting.
- int audio
  Stereo/mono setting. 0 == mono, 1 == stereo.
- int autostart
  Autostart discs? 0 == off, 1 == on.

8.24.1 Detailed Description

System configuration structure.

This structure is filled in with the settings set in the BIOS from the flashrom_get_syscfg() function.

8.24.2 Field Documentation

audio

```
int flashrom_syscfg_t::audio
```

Stereo/mono setting. 0 == mono, 1 == stereo.

autostart

`int flashrom_syscfg_t::autostart`

Autostart discs? 0 == off, 1 == on.

language

`int flashrom_syscfg_t::language`

Language setting.

See also

Language settings possible in the BIOS menu

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/flashrom.h

8.25 fs_socket_proto_t Struct Reference

Internal sockets protocol handler.

`#include <kos/fs_socket.h>`

Public Member Functions

- TAILQ_ENTRY (fs_socket_proto) entry
  Entry into the global list of protocols.

Data Fields

- int domain
  Domain of support for this protocol handler.
- int type
  Type of support for this protocol handler.
- int protocol
  Protocol of support for this protocol handler.
- int(∗ socket)(net_socket_t ∗s, int domain, int type, int protocol)
  Create a new socket for the protocol.
- void(∗ close)(net_socket_t ∗hnd)
  Close a socket that was created with the protocol.
- int(∗ accept)(net_socket_t ∗s, struct sockaddr ∗addr, socklen_t ∗alen)

Accept a connection on a socket created with the protocol.

- int(∗ bind)(net_socket_t ∗s, const struct sockaddr ∗addr, socklen_t alen)

  Bind a socket created with the protocol to an address.
- int(∗ connect)(net_socket_t ∗s, const struct sockaddr ∗addr, socklen_t alen)

  Connect a socket created with the protocol to a remote system.
- int(∗ listen)(net_socket_t ∗s, int backlog)

  Listen for incoming connections on a socket created with the protocol.
- ssize_t(∗ recvfrom)(net_socket_t ∗s, void ∗buffer, size_t len, int flags, struct sockaddr ∗addr, socklen_t ∗alen)

  Recieve data on a socket created with the protocol.
- ssize_t(∗ sendto)(net_socket_t ∗s, const void ∗msg, size_t len, int flags, const struct sockaddr ∗addr, socklen_t alen)

  Send data on a socket created with the protocol.
- int(∗ shutdownsock)(net_socket_t ∗s, int how)

  Shut down a socket created with the protocol.
- int(∗ input)(netif_t ∗src, int domain, const void ∗hdr, const uint8 ∗data, size_t size)

  Input a packet into a protocol.
- int(∗ getsockopt)(net_socket_t ∗s, int level, int option_name, void ∗option_value, socklen_t ∗option_len)

  Get socket options.
- int(∗ setsockopt)(net_socket_t ∗s, int level, int option_name, const void ∗option_value, socklen_t option_len)

  Set socket options.
- int(∗ getsockname)(net_socket_t ∗s, struct sockaddr ∗name, socklen_t ∗name_len)

  Get socket name.
- int(∗ fcntl)(net_socket_t ∗s, int cmd, va_list ap)

  Manipulate file options.
- short(∗ poll)(net_socket_t ∗s, short events)

  Poll for events.

8.25.1 Detailed Description

Internal sockets protocol handler.

This structure is a protocol handler used within fs_socket. Each protocol that is supported has one of these registered for it within the kernel. Generally, users will not come in contact with this structure (unless you're planning on writing a protocol handler), and it can generally be ignored.

For a complete list of appropriate errno values to return from any functions that are in here, take a look at the Single Unix Specification (aka, the POSIX spec), specifically the page about sys/socket.h (and all the functions that it defines, which is available at http://www.opengroup.org/onlinepubs/9699919799/basedefs/sys_socket.h.html .

8.25.2 Member Function Documentation

TAILQ_ENTRY()

```
fs_socket_proto_t::TAILQ_ENTRY (
            fs_socket_proto  )
```

Entry into the global list of protocols.

Contrary to what Doxygen might think, this is **NOT** a function. This should be initialized with the FS_SOCKET_PROTO_ENTRY macro before adding the protocol to the kernel with fs_socket_proto_add().

8.25.3 Field Documentation

accept

```
int(* fs_socket_proto_t::accept) (net_socket_t *s, struct sockaddr *addr, socklen_t *alen)
```

Accept a connection on a socket created with the protocol.

This function should implement the accept() system call for the protocol. The semantics are exactly as expected for that function.

Parameters

s	The socket to accept a connection on
addr	The address of the incoming connection
alen	The length of the address

Returns

A newly created socket for the incoming connection or -1 on error (with errno set appropriately)

bind

```
int(* fs_socket_proto_t::bind) (net_socket_t *s, const struct sockaddr *addr, socklen_t alen)
```

Bind a socket created with the protocol to an address.

This function should implement the bind() system call for the protocol. The semantics are exactly as expected for that function.

Parameters

s	The socket to bind to the address
addr	The address to bind to
alen	The length of the address

Return values

-1	On error (set errno appropriately)
0	On success

close

```
void(* fs_socket_proto_t::close) (net_socket_t *hnd)
```

Close a socket that was created with the protocol.

This function must do any work required to close a socket and destroy it. This function will be called when a socket requests to be closed with the close system call. There are no errors defined for this function.

Parameters

s	The socket to close

connect

```
int(* fs_socket_proto_t::connect) (net_socket_t *s, const struct sockaddr *addr, socklen_t alen)
```

Connect a socket created with the protocol to a remote system.

This function should implement the connect() system call for the protocol. The semantics are exactly as expected for that function.

Parameters

s	The socket to connect with
addr	The address to connect to
alen	The length of the address

Return values

-1	On error (with errno set appropriately)
0	On success

domain

```
int fs_socket_proto_t::domain
```

Domain of support for this protocol handler.

This field determines which sockets domain this protocol handler actually supports. This corresponds with the domain argument of the socket() function.

fcntl

```
int(* fs_socket_proto_t::fcntl) (net_socket_t *s, int cmd, va_list ap)
```

Manipulate file options.

This function should implement the fcntl() system call for the given protocol. The semantics are exactly as defined for that function.

Parameters

s	The socket to manipulate.
cmd	The fcntl command to run.
ap	Arguments to the command.

Return values

-1	On error (generally, set errno appropriately).

getsockname

```
int(* fs_socket_proto_t::getsockname) (net_socket_t *s, struct sockaddr *name, socklen_t *name_↩
len)
```

Get socket name.

This function should implement the getsockname() system call for the given protocol. The semantics are exactly as defined for that function.

Currently all options (regardless of level) are passed onto the protocol handler.

Parameters

s	The socket to get the name of.
name	Pointer to a sockaddr structure which will hold the resulting address information.
name_len	The amount of space pointed to by name, in bytes. On return, this is set to the actual size of the returned address information.

Return values

-1	On error (set errno appropriately).
0	On success.

getsockopt

```
int(* fs_socket_proto_t::getsockopt) (net_socket_t *s, int level, int option_name, void *option_↩
value, socklen_t *option_len)
```

Get socket options.

This function should implement the getsockopt() system call for the given protocol. The semantics are exactly as defined for that function.

Currently all options (regardless of level) are passed onto the protocol handler.

Parameters

s	The socket to get options for.
level	The protocol level to get options at.
option_name	The option to look up.
option_value	Storage for the value of the option.
option_len	The length of option_value on call, and the real option length (if less than the original value) on return.

Return values

-1	On error (set errno appropriately).
0	On success.

input

```
int(* fs_socket_proto_t::input) (netif_t *src, int domain, const void *hdr, const uint8 *data,
size_t size)
```

Input a packet into a protocol.

This function should read in the packet specified by the arguments and sort out what exactly to do with it. This usually involves checking if there is an open socket with the source address and adding it to a packet queue if there is.

Parameters

src	The interface the packet was input on
domain	The low-level protocol used (AF_INET or AF_INET6)
hdr	The low-level protocol header
data	The packet itself, including any protcol headers, but not any from lower-level protocols
size	The size of the packet, not including any lower- level protocol headers

Return values

-1	On error (the packet is discarded)
0	On success

listen

```
int(* fs_socket_proto_t::listen) (net_socket_t *s, int backlog)
```

Listen for incoming connections on a socket created with the protocol.

This function should implement the listen() system call for the protocol. The semantics are exactly as expected for that function.

Parameters

s	The socket to listen on
backlog	The number of connections to queue

Return values

-1	On error (with errno set appropriately)
0	On success

poll

```
short(* fs_socket_proto_t::poll) (net_socket_t *s, short events)
```

Poll for events.

This function should check the given socket for any events that may have already occured that are specified. This is used to back the poll() system call. This function should not block to wait for any events. This function may be called in an interrupt.

Parameters

s	The socket to poll.
events	The events to check for.

Return values

A	mask of any of the events specified that are currently true in the socket. 0 if none are true.

protocol

```
int fs_socket_proto_t::protocol
```

Protocol of support for this protocol handler.

This field determines the protocol that this protocol handler actually pays attention to. This corresponds with the protocol argument of the socket() function.

recvfrom

```
ssize_t(* fs_socket_proto_t::recvfrom) (net_socket_t *s, void *buffer, size_t len, int flags,
struct sockaddr *addr, socklen_t *alen)
```

Recieve data on a socket created with the protocol.

This function should implement the recvfrom() system call for the protocol. The semantics are exactly as expected for that function. Also, this function should implement the recv() system call, which will call this function with NULL for addr and alen.

Parameters

s	The socket to receive data on
buffer	The buffer to save data in
len	The length of the buffer
flags	Flags to the function
addr	Space to store the address that data came from (NULL if this was called by recv())
alen	Space to store the length of the address (NULL if this was called by recv())

Return values

-1	On error (set errno appropriately)
0	No outstanding data and the peer has disconnected cleanly
n	The number of bytes received (may be less than len)

sendto

```
ssize_t(* fs_socket_proto_t::sendto) (net_socket_t *s, const void *msg, size_t len, int flags,
const struct sockaddr *addr, socklen_t alen)
```

Send data on a socket created with the protocol.

This function should implement the sendto() system call for the protocol. The semantics are exactly as expected for that function. Also, this function should implement the send() system call, which will call this function with NULL for addr and 0 for alen.

Parameters

s	The socket to send data on
msg	The data to send
len	The length of data to send
flags	Flags to the function
addr	The address to send data to (NULL if this was called by send())
alen	The length of the address (0 if this was called by send())

Return values

-1	On error (set errno appropriately)
n	The number of bytes actually sent (may be less than len)

setsockopt

```
int(* fs_socket_proto_t::setsockopt) (net_socket_t *s, int level, int option_name, const void
*option_value, socklen_t option_len)
```

Set socket options.

This function should implement the setsockopt() system call for the given protocol. The semantics are exactly as defined for that function.

Currently all options (regardless of level) are passed onto the protocol handler.

Parameters

s	The socket to set options for.
level	The protocol level to set options at.
option_name	The option to set.
option_value	The value to set for the option.
option_len	The length of the option_value value.

Return values

-1	On error (set errno appropriately).
0	On success.

shutdownsock

```
int(* fs_socket_proto_t::shutdownsock) (net_socket_t *s, int how)
```

Shut down a socket created with the protocol.

This function should implement the shutdown() system call for the protocol. The semantics are exactly as expected for that function.

Parameters

s	The socket to shut down
how	What should be shut down on the socket

Return values

-1	On error (set errno appropriately)
0	On success

socket

```
int(* fs_socket_proto_t::socket) (net_socket_t *s, int domain, int type, int protocol)
```

Create a new socket for the protocol.

This function must create a new socket, initializing any data that the protocol might need for the socket, based on the parameters passed in. The socket passed in is already initialized prior to the handler being called, and will be cleaned up by fs_socket if an error is returned from the handler (a return value of -1).

Parameters

s	The socket structure to initialize
domain	Domain of the socket
type	Type of the socket
protocol	Protocol of the socket

Return values

-1	On error (errno should be set appropriately)
0	On success

type

```
int fs_socket_proto_t::type
```

Type of support for this protocol handler.

This field determines which types of sockets that this protocol handler pays attention to. This corresponds with the type argument of the socket() function.

The documentation for this struct was generated from the following file:

- include/kos/fs_socket.h

8.26 g2_ctx_t Struct Reference

G2 context.

```
#include <g2bus.h>
```

Data Fields

- uint32_t irq_state

8.26.1 Detailed Description

G2 context.

A G2 context containing the states of IRQs and G2 DMA. This struct is used in with g2_lock() and g2_unlock().

8.26.2 Field Documentation

irq_state

```
uint32_t g2_ctx_t::irq_state
```

Referenced by g2_lock(), and g2_unlock().

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/g2bus.h

8.27 hostent Struct Reference

Network host entry.

```
#include <netdb.h>
```

Data Fields

- char * h_name
  Official name of the host.
- char ** h_aliases
  Alternative host names.
- int h_addrtype
  Address type.
- int h_length
  Length of address, in bytes.
- char ** h_addr_list
  Network addresses of host.

8.27.1 Detailed Description

Network host entry.

This structure describes a network host entry in the address database. When looking up an address with the gethostbyname() function, one of these will be returned with information about the host.

8.27.2 Field Documentation

h_addr_list

```
char** hostent::h_addr_list
```

Network addresses of host.

h_addrtype

```
int hostent::h_addrtype
```

Address type.

h_aliases

```
char** hostent::h_aliases
```

Alternative host names.

h_length

```
int hostent::h_length
```

Length of address, in bytes.

h_name

```
char* hostent::h_name
```

Official name of the host.

The documentation for this struct was generated from the following file:

- include/netdb.h

8.28 in6_addr Struct Reference

Structure used to store an IPv6 address.

```
#include <netinet/in.h>
```

Data Fields

- union {
  uint8_t __s6_addr8 [16]
  uint16_t __s6_addr16 [8]
  uint32_t __s6_addr32 [4]
  uint64_t __s6_addr64 [2]
  } __s6_addr

8.28.1 Detailed Description

Structure used to store an IPv6 address.

8.28.2 Field Documentation

[union]

```
union { ...  } in6_addr::__s6_addr
```

__s6_addr16

```
uint16_t in6_addr::__s6_addr16[8]
```

__s6_addr32

```
uint32_t in6_addr::__s6_addr32[4]
```

__s6_addr64

```
uint64_t in6_addr::__s6_addr64[2]
```

__s6_addr8

```
uint8_t in6_addr::__s6_addr8[16]
```

The documentation for this struct was generated from the following file:

- include/netinet/in.h

8.29 in_addr Struct Reference

Structure used to store an IPv4 address.

```
#include <netinet/in.h>
```

Data Fields

- in_addr_t s_addr

8.29.1 Detailed Description

Structure used to store an IPv4 address.

8.29.2 Field Documentation

s_addr

```
in_addr_t in_addr::s_addr
```

The documentation for this struct was generated from the following file:

- include/netinet/in.h

8.30 iovec_t Struct Reference

I/O vector structure.

```
#include <sys/uio.h>
```

Data Fields

- void * iov_base
  Base address of memory for I/O.
- size_t iov_len
  Size of memory pointed to by iov_base.

8.30.1 Detailed Description

I/O vector structure.

Compatibility typedef for old code.

8.30.2 Field Documentation

iov_base

```
void* iovec_t::iov_base
```

Base address of memory for I/O.

iov_len

```
size_t iovec_t::iov_len
```

Size of memory pointed to by iov_base.

The documentation for this struct was generated from the following file:

- include/sys/uio.h

8.31 ip_hdr_t Struct Reference

IPv4 Packet header.

```
#include <kos/net.h>
```

Data Fields

- uint8 version_ihl
  IP version and header length.
- uint8 tos
  Type of Service.
- uint16 length
  Length.
- uint16 packet_id
  Packet ID.
- uint16 flags_frag_offs
  Flags and fragment offset.
- uint8 ttl
  Time to live.
- uint8 protocol
  IP protocol.
- uint16 checksum
  IP checksum.
- uint32 src
  Source IP address.
- uint32 dest
  Destination IP address.

8.31.1 Detailed Description

IPv4 Packet header.

8.31.2 Field Documentation

checksum

```
uint16 ip_hdr_t::checksum
```

IP checksum.

dest

```
uint32 ip_hdr_t::dest
```

Destination IP address.

flags_frag_offs

```
uint16 ip_hdr_t::flags_frag_offs
```

Flags and fragment offset.

length

```
uint16 ip_hdr_t::length
```

Length.

packet_id

```
uint16 ip_hdr_t::packet_id
```

Packet ID.

protocol

```
uint8 ip_hdr_t::protocol
```

IP protocol.

src

```
uint32 ip_hdr_t::src
```

Source IP address.

tos

```
uint8 ip_hdr_t::tos
```

Type of Service.

ttl

```
uint8 ip_hdr_t::ttl
```

Time to live.

version_ihl

```
uint8 ip_hdr_t::version_ihl
```

IP version and header length.

The documentation for this struct was generated from the following file:

- include/kos/net.h

8.32 ipv6_hdr_t Struct Reference

IPv6 Packet header.

```
#include <kos/net.h>
```

Data Fields

- uint8 version_lclass
  Version and low-order class byte.
- uint8 hclass_lflow
  High-order class byte, low-order flow byte.
- uint16 lclass
  Low-order class byte.
- uint16 length
  Length.
- uint8 next_header
  Next header type.
- uint8 hop_limit
  Hop limit.
- struct in6_addr src_addr
  Source IP address.
- struct in6_addr dst_addr
  Destination IP address.

8.32.1 Detailed Description

IPv6 Packet header.

8.32.2 Field Documentation

dst_addr

```
struct in6_addr ipv6_hdr_t::dst_addr
```

Destination IP address.

hclass_lflow

```
uint8 ipv6_hdr_t::hclass_lflow
```

High-order class byte, low-order flow byte.

hop_limit

```
uint8 ipv6_hdr_t::hop_limit
```

Hop limit.

lclass

```
uint16 ipv6_hdr_t::lclass
```

Low-order class byte.

length

```
uint16 ipv6_hdr_t::length
```

Length.

next_header

```
uint8 ipv6_hdr_t::next_header
```

Next header type.

src_addr

```
struct in6_addr ipv6_hdr_t::src_addr
```

Source IP address.

version_lclass

```
uint8 ipv6_hdr_t::version_lclass
```

Version and low-order class byte.

The documentation for this struct was generated from the following file:

- include/kos/net.h

8.33 irq_context_t Struct Reference

Architecture-specific structure for holding the processor state.

```
#include <arch/irq.h>
```

Data Fields

- uint32 r [16]
  16 general purpose (integer) registers
- uint32 pc
  Program counter.
- uint32 pr
  Procedure register (aka return address)
- uint32 gbr
  Global base register.
- uint32 vbr
  Vector base register.
- uint32 mach
  Multiply-and-accumulate register (high)
- uint32 macl
  Multiply-and-accumulate register (low)
- uint32 sr
  Status register.
- uint32 frbank [16]
  Secondary floating poing registers.
- uint32 fr [16]
  Primary floating point registers.
- uint32 fpscr
  Floating-point status/control register.
- uint32 fpul
  Floatint-point communication register.

8.33.1 Detailed Description

Architecture-specific structure for holding the processor state.

This structure should hold register values and other important parts of the processor state. The size of this structure should be less than or equal to the REG_BYTE_CNT value.

8.33.2 Field Documentation

fpscr

```
uint32 irq_context_t::fpscr
```

Floating-point status/control register.

fpul

```
uint32 irq_context_t::fpul
```

Floatint-point communication register.

fr

```
uint32 irq_context_t::fr[16]
```

Primary floating point registers.

frbank

```
uint32 irq_context_t::frbank[16]
```

Secondary floating poing registers.

gbr

```
uint32 irq_context_t::gbr
```

Global base register.

mach

```
uint32 irq_context_t::mach
```

Multiply-and-accumulate register (high)

macl

```
uint32 irq_context_t::macl
```

Multiply-and-accumulate register (low)

pc

```
uint32 irq_context_t::pc
```

Program counter.

pr

```
uint32 irq_context_t::pr
```

Procedure register (aka return address)

r

```
uint32 irq_context_t::r[16]
```

16 general purpose (integer) registers

sr

```
uint32 irq_context_t::sr
```

Status register.

vbr

```
uint32 irq_context_t::vbr
```

Vector base register.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/arch/irq.h

8.34 kbd_cond_t Struct Reference

Keyboard raw condition structure.

`#include <dc/maple/keyboard.h>`

Data Fields

- uint8 modifiers
  Bitmask of set modifiers.
- uint8 leds
  Bitmask of set LEDs.
- uint8 keys [MAX_PRESSED_KEYS]
  Key codes for currently pressed keys.

8.34.1 Detailed Description

Keyboard raw condition structure.

This structure is what the keyboard responds with as its current status.

8.34.2 Field Documentation

keys

```
uint8 kbd_cond_t::keys[MAX_PRESSED_KEYS]
```

Key codes for currently pressed keys.

leds

```
uint8 kbd_cond_t::leds
```

Bitmask of set LEDs.

modifiers

```
uint8 kbd_cond_t::modifiers
```

Bitmask of set modifiers.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple/keyboard.h

8.35 kbd_keymap_t Struct Reference

Keyboard keymap.

`#include <dc/maple/keyboard.h>`

Data Fields

- uint8 base [MAX_KBD_KEYS]
- uint8 shifted [MAX_KBD_KEYS]
- uint8 alt [MAX_KBD_KEYS]

8.35.1 Detailed Description

Keyboard keymap.

This structure represents a mapping from raw key values to ASCII values, if appropriate. This handles base values as well as shifted ("shift" and "Alt" keys) values.

8.35.2 Field Documentation

alt

```
uint8 kbd_keymap_t::alt[MAX_KBD_KEYS]
```

base

```
uint8 kbd_keymap_t::base[MAX_KBD_KEYS]
```

shifted

```
uint8 kbd_keymap_t::shifted[MAX_KBD_KEYS]
```

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple/keyboard.h

8.36 kbd_state_t Struct Reference

Keyboard status structure.

`#include <dc/maple/keyboard.h>`

Data Fields

- kbd_cond_t cond

  The lastest raw condition of the keyboard.
- uint8 matrix [MAX_KBD_KEYS]

  Key array.
- int shift_keys

  Modifier key status.
- int region

  Keyboard type/region.
- uint32 key_queue [KBD_QUEUE_SIZE]

  Individual keyboard queue. You should not access this variable directly. Please use the appropriate function to access it.
- int queue_tail

  Key queue tail.
- int queue_head

  Key queue head.
- int queue_len

  Current length of queue.
- uint8 kbd_repeat_key

  Key that is repeating.
- uint64 kbd_repeat_timer

  Time that the next repeat will trigger.

8.36.1 Detailed Description

Keyboard status structure.

This structure holds information about the current status of the keyboard device. This is what maple_dev_status() will return.

8.36.2 Field Documentation

cond

```
kbd_cond_t kbd_state_t::cond
```

The lastest raw condition of the keyboard.

kbd_repeat_key

```
uint8 kbd_state_t::kbd_repeat_key
```

Key that is repeating.

kbd_repeat_timer

```
uint64 kbd_state_t::kbd_repeat_timer
```

Time that the next repeat will trigger.

key_queue

```
uint32 kbd_state_t::key_queue[KBD_QUEUE_SIZE]
```

Individual keyboard queue. You should not access this variable directly. Please use the appropriate function to access it.

matrix

```
uint8 kbd_state_t::matrix[MAX_KBD_KEYS]
```

Key array.

This array lists the state of all possible keys on the keyboard. It can be used for key repeat and debouncing. This will be non-zero if the key is currently being pressed.

See also

Keyboard keys

queue_head

```
int kbd_state_t::queue_head
```

Key queue head.

queue_len

```
int kbd_state_t::queue_len
```

Current length of queue.

queue_tail

```
int kbd_state_t::queue_tail
```

Key queue tail.

region

```
int kbd_state_t::region
```

Keyboard type/region.

shift_keys

```
int kbd_state_t::shift_keys
```

Modifier key status.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple/keyboard.h

8.37 klibrary_t Struct Reference

Loaded library structure.

```
#include <kos/library.h>
```

Public Member Functions

- LIST_ENTRY (klibrary) list
  Library list handle.

Data Fields

- libid_t libid
  Library ID (assigned at runtime).
- uint32 flags
  Library flags.
- elf_prog_t image
  ELF image for this library.
- int refcnt
  Library reference count.
- const char *(* lib_get_name)(void)
  Retrieve the library's symbolic name.
- uint32(* lib_get_version)(void)
  Retrieve the library's version.
- int(* lib_open)(struct klibrary *lib)
  Open a library.
- int(* lib_close)(struct klibrary *lib)
  Close an opened library.

8.37.1 Detailed Description

Loaded library structure.

This structure represents a single loaded library. Each library is essentially a loaded binary of code and a set of exported entry points that are standardized.

Each loaded library should export at least the functions described in this structure:

- const char *lib_get_name()
- uint32 lib_get_version()
- int lib_open(struct klibrary *lib)
- int lib_close(struct klibrary *lib)

You should not modify any members of this structure yourself (except if you are implementing a library).

8.37.2 Member Function Documentation

LIST_ENTRY()

```
klibrary_t::LIST_ENTRY (
            klibrary  )
```

Library list handle.

Contrary to what doxygen might think, this is not a function.

8.37.3 Field Documentation

flags

```
uint32 klibrary_t::flags
```

Library flags.

image

```
elf_prog_t klibrary_t::image
```

ELF image for this library.

This can be used to look up additional entry points in the library.

lib_close

```
int(* klibrary_t::lib_close) (struct klibrary *lib)
```

Close an opened library.

This function must be implemented by all loadable libraries to close and deinitialize a library. If the library's reference count is $>$ 1 when this function is called, this may involve simply decrementing the reference count.

Parameters

lib	The library structure

Returns

Values >= 0 indicate success, < 0 indicates failure

lib_get_name

```
const char *(* klibrary_t::lib_get_name) (void)
```

Retrieve the library's symbolic name.

This function must be implemented by all loadable libraries to fetch the library's symbolic name. This function must work before calling lib_open() on the library.

Returns

The library's symbolic name

lib_get_version

```
uint32(* klibrary_t::lib_get_version) (void)
```

Retrieve the library's version.

This function must be implemented by all loadble libraries to fetch the library's version number. This function must work before calling lib_open() on the library.

Returns

The library's version number

lib_open

```
int(* klibrary_t::lib_open) (struct klibrary *lib)
```

Open a library.

This function must be implemented by all loadable libraries to initialize the library on load. If the library is already opened, this may only involve increasing the reference count.

Parameters

lib	The library structure

Returns

Values >= 0 indicate success, < 0 indicates failure. A failure on the first lib_open is indicative that the library should be removed from memory.

libid

```
libid_t klibrary_t::libid
```

Library ID (assigned at runtime).

refcnt

```
int klibrary_t::refcnt
```

Library reference count.

This value is incremented every time the library is opened, and decremented each time it is closed. Once the library's reference count hits 0, a close will actually destroy the library.

The documentation for this struct was generated from the following file:

- include/kos/library.h

8.38 kos_blockdev_t Struct Reference

A simple block device.

```
#include <kos/blockdev.h>
```

Data Fields

- void * dev_data
  Internal device data.
- uint32_t l_block_size
  Log base 2 of the bytes per block.
- int(* init)(struct kos_blockdev *d)
  Initialize the block device.
- int(* shutdown)(struct kos_blockdev *d)
  Shut down the block device.
- int(* read_blocks)(struct kos_blockdev *d, uint64_t block, size_t count, void *buf)
  Read a number of blocks from the device.
- int(* write_blocks)(struct kos_blockdev *d, uint64_t block, size_t count, const void *buf)
  Write a number of blocks to the device.
- uint64_t(* count_blocks)(struct kos_blockdev *d)
  Count the number of blocks on the device.
- int(* flush)(struct kos_blockdev *d)
  Flush the write cache (if any) of the device.

8.38.1 Detailed Description

A simple block device.

This structure represents a single block device. Each block device should be associated with exactly one filesystem and is used to actually read the data from the disk (or other device) where it is stored.

By using a block device with any new filesystems, we can abstract away a few things so that filesystems can be used with a variety of different "devices", such as the SD card reader for the Dreamcast or a disk image file of some sort.

8.38.2 Field Documentation

count_blocks

```
uint64_t(* kos_blockdev_t::count_blocks) (struct kos_blockdev *d)
```

Count the number of blocks on the device.

This function should return the total number of blocks on the device. There is no expectation of the device to keep track of which blocks are in use or anything else of the sort.

Parameters

d	The device to read the block count from.

Returns

The number of blocks that the device has.

dev_data

```
void* kos_blockdev_t::dev_data
```

Internal device data.

flush

```
int(* kos_blockdev_t::flush) (struct kos_blockdev *d)
```

Flush the write cache (if any) of the device.

This function shall signal to the device that any write caches that are present on the device shall be flushed so that all data written to this point shall persist to the underlying storage.

Parameters

d	The device to flush caches on.

Return values

0	On success.
-1	On failure. Set errno as appropriate.

init

```
int(* kos_blockdev_t::init) (struct kos_blockdev *d)
```

Initialize the block device.

This function should do any necessary initialization to use the block device passed in.

Parameters

d	The device to initialize.

Return values

0	On success.
-1	On failure. Set errno as appropriate.

l_block_size

```
uint32_t kos_blockdev_t::l_block_size
```

Log base 2 of the bytes per block.

read_blocks

```
int(* kos_blockdev_t::read_blocks) (struct kos_blockdev *d, uint64_t block, size_t count, void
*buf)
```

Read a number of blocks from the device.

This function should read the specified number of device blocks into the given buffer. The buffer will already be allocated by the caller.

Parameters

d	The device to read from.
block	The first block to read.
count	The number of blocks to read.
buf	The buffer to read into.

Return values

0	On success.
-1	On failure. Set errno as appropriate.

shutdown

```
int(* kos_blockdev_t::shutdown) (struct kos_blockdev *d)
```

Shut down the block device.

This function should do any teardown work that is needed to clean up the block device.

Parameters

d	The device to shut down.

Return values

0	On success.
-1	On failure. Set errno as appropriate.

write_blocks

```
int(* kos_blockdev_t::write_blocks) (struct kos_blockdev *d, uint64_t block, size_t count, const
void *buf)
```

Write a number of blocks to the device.

This function should write the specified number of device blocks onto the device from the given buffer.

Parameters

d	The device to write to.
block	The first block to write.
count	The number of blocks to write.
buf	The buffer to write from.

Return values

0	On success.
-1	On failure. Set errno as appropriate.

The documentation for this struct was generated from the following file:

- include/kos/blockdev.h

8.39 kos_img_t Struct Reference

Platform-indpendent image type.

`#include <kos/img.h>`

Data Fields

- void * data
  Image data in the specified format.
- uint32 w
  Width of the image.
- uint32 h
  Height of the image.
- uint32 fmt
  Format of the image data.
- uint32 byte_count
  Length of the image data, in bytes.

8.39.1 Detailed Description

Platform-indpendent image type.

You can use this type for textures or whatever you feel it's appropriate for. "width" and "height" are as you would expect. "format" has a lower-half which is platform-independent and used to basically describe the contained data; the upper-half is platform-dependent and can hold anything (so AND it off if you only want the bottom part).

Note that in some of the more obscure formats (like the paletted formats) the data interpretation may be platform dependent. Thus we also provide a data length field.

8.39.2 Field Documentation

byte_count

`uint32 kos_img_t::byte_count`

Length of the image data, in bytes.

data

```
void* kos_img_t::data
```

Image data in the specified format.

fmt

```
uint32 kos_img_t::fmt
```

Format of the image data.

See also

Image format types

Macros for accessing the format of an image

h

```
uint32 kos_img_t::h
```

Height of the image.

w

```
uint32 kos_img_t::w
```

Width of the image.

The documentation for this struct was generated from the following file:

- addons/include/kos/img.h

8.40 kos_md5_cxt_t Struct Reference

MD5 context.

```
#include <kos/md5.h>
```

Data Fields

- uint64 size
  Size of the data in buf.
- uint32 hash [4]
  Intermediate hash value.
- uint8 buf [64]
  Temporary storage of values to be hashed.

8.40.1 Detailed Description

MD5 context.

This structure contains the variables needed to maintain the internal state of the MD5 code. You should not manipulate these variables manually, but rather use the kos_md5_* functions to do everything you need.

8.40.2 Field Documentation

buf

```
uint8 kos_md5_cxt_t::buf[64]
```

Temporary storage of values to be hashed.

hash

```
uint32 kos_md5_cxt_t::hash[4]
```

Intermediate hash value.

size

```
uint64 kos_md5_cxt_t::size
```

Size of the data in buf.

The documentation for this struct was generated from the following file:

- addons/include/kos/md5.h

8.41 kthread_attr_t Struct Reference

Thread creation attributes.

```
#include <kos/thread.h>
```

Data Fields

- int create_detached
  1 for a detached thread.
- uint32_t stack_size
  Set the size of the stack to be created.
- void * stack_ptr
  Pre-allocate a stack for the thread.
- prio_t prio
  Set the thread's priority.
- const char * label
  Thread label.

8.41.1 Detailed Description

Thread creation attributes.

This structure allows you to specify the various attributes for a thread to have when it is created. These can only be modified (in general) at thread creation time (with the exception of detaching a thread, which can be done later with thd_detach()).

Leaving any of the attributes in this structure 0 will set them to their default value.

8.41.2 Field Documentation

create_detached

```
int kthread_attr_t::create_detached
```

1 for a detached thread.

label

```
const char* kthread_attr_t::label
```

Thread label.

prio

```
prio_t kthread_attr_t::prio
```

Set the thread's priority.

stack_ptr

```
void* kthread_attr_t::stack_ptr
```

Pre-allocate a stack for the thread.

Note

If you use this attribute, you must also set stack_size.

stack_size

```
uint32_t kthread_attr_t::stack_size
```

Set the size of the stack to be created.

The documentation for this struct was generated from the following file:

- include/kos/thread.h

8.42 kthread_t Struct Reference

Structure describing one running thread.

```
#include <kos/thread.h>
```

Public Member Functions

- LIST_ENTRY (kthread) t_list

  Thread list handle. Not a function.
- TAILQ_ENTRY (kthread) thdq

  Run/Wait queue handle. Once again, not a function.
- TAILQ_ENTRY (kthread) timerq

  Timer queue handle (if applicable). Also not a function.

Data Fields

- tid_t tid

  Kernel thread id.
- prio_t prio

  Static priority: 0..PRIO_MAX (higher means lower priority).
- uint32_t flags

  Thread flags.
- int state

  Process state.
- void * wait_obj

  Generic wait target, if waiting.
- const char * wait_msg

  Generic wait message, if waiting.
- void(* wait_callback)(void *obj)

  Wait timeout callback.
- uint64_t wait_timeout

  Next scheduled time. This value is used for sleep and timed block operations. This value is in milliseconds since the start of timer_ms_gettime(). This should be enough for something like 2 million years of wait time. ;)
- char label [KTHREAD_LABEL_SIZE]

  Thread label. This value is used when printing out a user-readable process listing.
- char pwd [KTHREAD_PWD_SIZE]

  Current file system path.
- irq_context_t context

  Register store – used to save thread context.
- uint32_t * stack

  Thread private stack. This should be a pointer to the base of a stack page.
- uint32_t stack_size

  Size of the thread's stack, in bytes.
- int thd_errno

  Thread errno variable.
- struct _reent thd_reent

  Our reent struct for newlib.
- struct kthread_tls_kv_list tls_list

  OS-level thread-local storage.
- tcbhead_t * tcbhead

  Compiler-level thread-local storage.
- void * rv

  Return value of the thread function. This is only used in joinable threads.

Related Symbols

(Note that these are not member symbols.)

- kthread_t * thd_by_tid (tid_t tid)

  Given a thread ID, locates the thread structure.
- void thd_add_to_runnable (kthread_t *t, int front_of_line)

  Enqueue a process in the runnable queue.
- int thd_remove_from_runnable (kthread_t *thd)

  Removes a thread from the runnable queue, if it's there.
- kthread_t * thd_create (int detach, void *(*routine)(void *param), void *param)

  Create a new thread.
- kthread_t * thd_create_ex (const kthread_attr_t *__RESTRICT attr, void *(*routine)(void *param), void *param)

  Create a new thread with the specified set of attributes.
- int thd_destroy (kthread_t *thd)

  Brutally kill the given thread.
- void thd_schedule_next (kthread_t *thd)

  Force a given thread to the front of the queue.
- int thd_set_prio (kthread_t *thd, prio_t prio)

  Set a thread's priority value.
- kthread_t * thd_get_current (void)

  Retrieve the current thread's kthread struct.
- const char * thd_get_label (kthread_t *thd)

  Retrieve the thread's label.
- void thd_set_label (kthread_t *thd, const char *__RESTRICT label)

  Set the thread's label.
- const char * thd_get_pwd (kthread_t *thd)

  Retrieve the thread's current working directory.
- void thd_set_pwd (kthread_t *thd, const char *__RESTRICT pwd)

  Set the thread's current working directory.
- int * thd_get_errno (kthread_t *thd)

  Retrieve a pointer to the thread errno.
- struct _reent * thd_get_reent (kthread_t *thd)

  Retrieve a pointer to the thread reent struct.
- int thd_join (kthread_t *thd, void **value_ptr)

  Wait for a thread to exit.
- int thd_detach (kthread_t *thd)

  Detach a joinable thread.
- int thd_each (int(*cb)(kthread_t *thd, void *user_data), void *data)

  Iterate all threads and call the passed callback for each.

8.42.1 Detailed Description

Structure describing one running thread.

Each thread has one of these structures assigned to it, which holds all the data associated with the thread. There are various functions to manipulate the data in here, so you shouldn't generally do so manually.

8.42.2 Member Function Documentation

LIST_ENTRY()

```
kthread_t::LIST_ENTRY (
            kthread  )
```

Thread list handle. Not a function.

TAILQ_ENTRY() [1/2]

```
kthread_t::TAILQ_ENTRY (
            kthread  )
```

Run/Wait queue handle. Once again, not a function.

TAILQ_ENTRY() [2/2]

```
kthread_t::TAILQ_ENTRY (
            kthread  )
```

Timer queue handle (if applicable). Also not a function.

8.42.3 Friends And Related Symbol Documentation

thd_add_to_runnable()

```
void thd_add_to_runnable (
            kthread_t * t,
            int front_of_line )  [related]
```

Enqueue a process in the runnable queue.

This function adds a thread to the runnable queue after the process group of the same priority if front_of_line is zero, otherwise queues it at the front of its priority group. Generally, you will not have to do this manually.

Parameters

t	The thread to queue.
front_of_line	Set to 1 to put this thread in front of other threads of the same priority, 0 to put it behind the other threads (normal behavior).

See also

thd_remove_from_runnable

thd_by_tid()

```
kthread_t * thd_by_tid (
            tid_t tid )  [related]
```

Given a thread ID, locates the thread structure.

Parameters

tid	The thread ID to retrieve.

Returns

The thread on success, NULL on failure.

thd_create()

```
kthread_t * thd_create (
            int detach,
            void *(*)(void *param) routine,
            void * param )  [related]
```

Create a new thread.

This function creates a new kernel thread with default parameters to run the given routine. The thread will terminate and clean up resources when the routine completes if the thread is created detached, otherwise you must join the thread with thd_join() to clean up after it.

Parameters

detach	Set to 1 to create a detached thread. Set to 0 to create a joinable thread.
routine	The function to call in the new thread.
param	A parameter to pass to the function called.

Returns

The new thread on success, NULL on failure.

See also

thd_create_ex, thd_destroy

thd_create_ex()

```
kthread_t * thd_create_ex (
            const kthread_attr_t *__RESTRICT attr,
            void *(*)(void *param) routine,
            void * param )  [related]
```

Create a new thread with the specified set of attributes.

This function creates a new kernel thread with the specified set of parameters to run the given routine.

Parameters

attr	A set of thread attributes for the created thread. Passing NULL will initialize all attributes to their default values.
routine	The function to call in the new thread.
param	A parameter to pass to the function called.

Returns

The new thread on success, NULL on failure.

See also

thd_create, thd_destroy

thd_destroy()

```
int thd_destroy (
            kthread_t * thd )  [related]
```

Brutally kill the given thread.

This function kills the given thread, removing it from the execution chain, cleaning up thread-local data and other internal structures. In general, you shouldn't call this function at all.

Warning

You should never call this function on the current thread.

Parameters

thd	The thread to destroy.

Return values

0	On success.

See also

thd_create

thd_detach()

```
int thd_detach (
             kthread_t * thd )  [related]
```

Detach a joinable thread.

This function switches the specified thread's mode from THD_MODE_JOINABLE to THD_MODE_DETACHED. This will ensure that the thread cleans up all of its internal resources when it exits.

Parameters

thd	The joinable thread to detach.

Returns

0 on success or less than 0 if the thread is non-existant or already detached.

See also

thd_join()

thd_each()

```
int thd_each (
             int(*)(kthread_t *thd, void *user_data) cb,
             void * data )  [related]
```

Iterate all threads and call the passed callback for each.

Parameters

cb	The callback to call for each thread. If a nonzero value is returned, iteration ceases immediately.
data	User data to be passed to the callback

Return values

0	or the first nonzero value returned by `cb`.

See also

thd_pslist

thd_get_current()

```
kthread_t * thd_get_current (
            void  )  [related]
```

Retrieve the current thread's kthread struct.

Returns

The current thread's structure.

thd_get_errno()

```
int * thd_get_errno (
            kthread_t * thd )  [related]
```

Retrieve a pointer to the thread errno.

This function retrieves a pointer to the errno value for the thread. You should generally just use the errno variable to access this.

Parameters

thd	The thread to retrieve from.

Returns

A pointer to the thread's errno.

thd_get_label()

```
const char * thd_get_label (
            kthread_t * thd )  [related]
```

Retrieve the thread's label.

Parameters

thd	The thread to retrieve from.

Returns

The human-readable label of the thread.

See also

thd_set_label

thd_get_pwd()

```
const char * thd_get_pwd (
             kthread_t * thd )  [related]
```

Retrieve the thread's current working directory.

This function retrieves the working directory of a thread. Generally, you will want to use either fs_getwd() or one of the standard C functions for doing this, but this is here in case you need it when the thread isn't active for some reason.

Parameters

thd	The thread to retrieve from.

Returns

The thread's working directory.

See also

thd_set_pd

thd_get_reent()

```
struct _reent * thd_get_reent (
             kthread_t * thd )  [related]
```

Retrieve a pointer to the thread reent struct.

This function is used to retrieve some internal state that is used by newlib to provide a reentrant libc.

Parameters

thd	The thread to retrieve from.

Returns

The thread's reent struct.

thd_join()

```
int thd_join (
              kthread_t * thd,
              void ** value_ptr )  [related]
```

Wait for a thread to exit.

This function "joins" a joinable thread. This means effectively that the calling thread blocks until the speified thread completes execution. It is invalid to join a detached thread, only joinable threads may be joined.

Parameters

thd	The joinable thread to join.
value_ptr	A pointer to storage for the thread's return value, or NULL if you don't care about it.

Returns

0 on success, or less than 0 if the thread is non-existant or not joinable.

See also

thd_detach

thd_remove_from_runnable()

```
int thd_remove_from_runnable (
              kthread_t * thd )  [related]
```

Removes a thread from the runnable queue, if it's there.

This function removes a thread from the runnable queue, if it is currently in that queue. Generally, you shouldn't have to do this manually, as waiting on synchronization primitives and the like will do this for you if needed.

Parameters

thd	The thread to remove from the runnable queue.

Return values

0	On success, or if the thread isn't runnable.

See also

thd_add_to_runnable

thd_schedule_next()

```
void thd_schedule_next (
            kthread_t * thd )  [related]
```

Force a given thread to the front of the queue.

This function promotes the given thread to be the next one that will be swapped in by the scheduler. This function is only callable inside an interrupt context (it simply returns otherwise).

thd_set_label()

```
void thd_set_label (
            kthread_t * thd,
            const char *__RESTRICT label )  [related]
```

Set the thread's label.

This function sets the label of a thread, which is simply a human-readable string that is used to identify the thread. These labels aren't used for anything internally, and you can give them any label you want. These are mainly seen in the printouts from thd_pslist() or thd_pslist_queue().

Parameters

thd	The thread to set the label of.
label	The string to set as the label.

See also

thd_get_label

thd_set_prio()

```
int thd_set_prio (
            kthread_t * thd,
            prio_t prio )  [related]
```

Set a thread's priority value.

This function is used to change the priority value of a thread. If the thread is scheduled already, it will be rescheduled with the new priority value.

Parameters

thd	The thread to change the priority of.
prio	The priority value to assign to the thread.

Return values

0	On success.
-1	thd is NULL.
-2	prio requested was out of range.

thd_set_pwd()

```
void thd_set_pwd (
            kthread_t * thd,
            const char *__RESTRICT pwd )  [related]
```

Set the thread's current working directory.

This function will set the working directory of a thread. Generally, you will want to use either fs_chdir() or the standard C chdir() function to do this, but this is here in case you need to do it while the thread isn't active for some reason.

Parameters

thd	The thread to set the working directory of.
pwd	The directory to set as active.

See also

thd_get_pwd

8.42.4 Field Documentation

context

```
irq_context_t kthread_t::context
```

Register store – used to save thread context.

flags

```
uint32_t kthread_t::flags
```

Thread flags.

See also

thd_flags

label

```
char kthread_t::label[KTHREAD_LABEL_SIZE]
```

Thread label. This value is used when printing out a user-readable process listing.

prio

```
prio_t kthread_t::prio
```

Static priority: 0..PRIO_MAX (higher means lower priority).

pwd

```
char kthread_t::pwd[KTHREAD_PWD_SIZE]
```

Current file system path.

rv

```
void* kthread_t::rv
```

Return value of the thread function. This is only used in joinable threads.

stack

```
uint32_t* kthread_t::stack
```

Thread private stack. This should be a pointer to the base of a stack page.

stack_size

```
uint32_t kthread_t::stack_size
```

Size of the thread's stack, in bytes.

state

```
int kthread_t::state
```

Process state.

See also

thd_states

tcbhead

```
tcbhead_t* kthread_t::tcbhead
```

Compiler-level thread-local storage.

thd_errno

```
int kthread_t::thd_errno
```

Thread errno variable.

thd_reent

```
struct _reent kthread_t::thd_reent
```

Our reent struct for newlib.

tid

```
tid_t kthread_t::tid
```

Kernel thread id.

tls_list

```
struct kthread_tls_kv_list kthread_t::tls_list
```

OS-level thread-local storage.

See also

kos/tls.h

wait_callback

```
void(* kthread_t::wait_callback) (void *obj)
```

Wait timeout callback.

If the genwait times out while waiting, this function will be called. This allows hooks for things like fixing up semaphore count values, etc.

Parameters

obj	The object that we were waiting on.

wait_msg

```
const char* kthread_t::wait_msg
```

Generic wait message, if waiting.

See also

kos/genwait.h

wait_obj

```
void* kthread_t::wait_obj
```

Generic wait target, if waiting.

See also

kos/genwait.h

wait_timeout

```
uint64_t kthread_t::wait_timeout
```

Next scheduled time. This value is used for sleep and timed block operations. This value is in milliseconds since the start of timer_ms_gettime(). This should be enough for something like 2 million years of wait time. ;)

The documentation for this struct was generated from the following file:

- include/kos/thread.h

8.43 kthread_tls_kv_t Struct Reference

Thread-local storage key-value pair.

```
#include <tls.h>
```

Public Member Functions

- LIST_ENTRY (kthread_tls_kv) kv_list
  List handle – NOT a function.

Data Fields

- kthread_key_t key
  The key associated with this data.
- void * data
  The value of the data.
- void(* destructor)(void *)
  Optional destructor for the value (set per key).

8.43.1 Detailed Description

Thread-local storage key-value pair.

This is the structure that is actually used to store the specific value for a thread for a single TLS key.

You will not end up using these directly at all in programs, as they are only used internally.

8.43.2 Member Function Documentation

LIST_ENTRY()

```
kthread_tls_kv_t::LIST_ENTRY (
            kthread_tls_kv  )
```

List handle – NOT a function.

8.43.3 Field Documentation

data

```
void* kthread_tls_kv_t::data
```

The value of the data.

destructor

```
void(* kthread_tls_kv_t::destructor) (void *)
```

Optional destructor for the value (set per key).

key

```
kthread_key_t kthread_tls_kv_t::key
```

The key associated with this data.

The documentation for this struct was generated from the following file:

- include/kos/tls.h

8.44 mallinfo Struct Reference

ANSI C functions.

```
#include <malloc.h>
```

Data Fields

- int arena
  non-mmapped space allocated from system
- int ordblks
  number of free chunks
- int smblks
  number of fastbin blocks
- int hblks
  number of mmapped regions
- int hblkhd
  space in mmapped regions
- int usmblks
  maximum total allocated space
- int fsmblks
  space available in freed fastbin blocks
- int uordblks
  total allocated space
- int fordblks
  total free space
- int keepcost
  top-most, releasable (via malloc_trim) space

8.44.1 Detailed Description

ANSI C functions.

8.44.2 Field Documentation

arena

```
int mallinfo::arena
```

non-mmapped space allocated from system

fordblks

```
int mallinfo::fordblks
```

total free space

fsmblks

```
int mallinfo::fsmblks
```

space available in freed fastbin blocks

hblkhd

```
int mallinfo::hblkhd
```

space in mmapped regions

hblks

```
int mallinfo::hblks
```

number of mmapped regions

keepcost

```
int mallinfo::keepcost
```

top-most, releasable (via malloc_trim) space

ordblks

```
int mallinfo::ordblks
```

number of free chunks

smblks

```
int mallinfo::smblks
```

number of fastbin blocks

uordblks

```
int mallinfo::uordblks
```

total allocated space

usmblks

```
int mallinfo::usmblks
```

maximum total allocated space

The documentation for this struct was generated from the following file:

- include/malloc.h

8.45 maple_device_t Struct Reference

One maple device.

```
#include <dc/maple.h>
```

Data Fields

- int valid
  Is this a valid device?
- int port
  Maple bus port connected to.
- int unit
  Unit number, off of the port.
- maple_devinfo_t info
  Device info struct.
- int dev_mask
  Device-present mask for unit 0's.
- maple_frame_t frame
  One rx/tx frame.
- struct maple_driver * drv
  Driver which handles this device.
- volatile int status_valid
  Have we got our first status update?
- uint8 status [1024]
  Status buffer (for pollable devices)

8.45.1 Detailed Description

One maple device.

Note that we duplicate the port/unit info which is normally somewhat implicit so that we can pass around a pointer to a particular device struct.

8.45.2 Field Documentation

dev_mask

```
int maple_device_t::dev_mask
```

Device-present mask for unit 0's.

drv

```
struct maple_driver* maple_device_t::drv
```

Driver which handles this device.

frame

```
maple_frame_t maple_device_t::frame
```

One rx/tx frame.

info

```
maple_devinfo_t maple_device_t::info
```

Device info struct.

port

```
int maple_device_t::port
```

Maple bus port connected to.

status

```
uint8 maple_device_t::status[1024]
```

Status buffer (for pollable devices)

status_valid

```
volatile int maple_device_t::status_valid
```

Have we got our first status update?

unit

```
int maple_device_t::unit
```

Unit number, off of the port.

valid

```
int maple_device_t::valid
```

Is this a valid device?

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple.h

8.46 maple_devinfo_t Struct Reference

Maple device info structure.

`#include <dc/maple.h>`

Data Fields

- uint32 functions
  Function codes supported.
- uint32 function_data [3]
  Additional data per function.
- uint8 area_code
  Region code.
- uint8 connector_direction
  0: UP (most controllers), 1: DOWN (lightgun, microphones)
- char product_name [30]
  Name of device.
- char product_license [60]
  License statement.
- uint16 standby_power
  Power consumption (standby)
- uint16 max_power
  Power consumption (max)

8.46.1 Detailed Description

Maple device info structure.

This structure is used by the hardware to deliver the response to the device info request.

8.46.2 Field Documentation

area_code

`uint8 maple_devinfo_t::area_code`

Region code.

connector_direction

`uint8 maple_devinfo_t::connector_direction`

0: UP (most controllers), 1: DOWN (lightgun, microphones)

function_data

```
uint32 maple_devinfo_t::function_data[3]
```

Additional data per function.

functions

```
uint32 maple_devinfo_t::functions
```

Function codes supported.

max_power

```
uint16 maple_devinfo_t::max_power
```

Power consumption (max)

product_license

```
char maple_devinfo_t::product_license[60]
```

License statement.

product_name

```
char maple_devinfo_t::product_name[30]
```

Name of device.

standby_power

```
uint16 maple_devinfo_t::standby_power
```

Power consumption (standby)

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple.h

8.47 maple_driver_t Struct Reference

A maple device driver.

`#include <dc/maple.h>`

Public Member Functions

- LIST_ENTRY (maple_driver) drv_list
  Driver list handle. NOT A FUNCTION!

Data Fields

- uint32 functions
  One or more MAPLE_FUNCs ORed together.
- const char * name
  The driver name.
- void(* periodic)(struct maple_driver *drv)
  Periodic polling callback.
- int(* attach)(struct maple_driver *drv, maple_device_t *dev)
  Device attached callback.
- void(* detach)(struct maple_driver *drv, maple_device_t *dev)
  Device detached callback.

8.47.1 Detailed Description

A maple device driver.

Anything which is added to this list is capable of handling one or more maple device types. When a device of the given type is connected (includes startup "connection"), the driver is invoked. This same process happens for disconnection, response receipt, and on a periodic interval (for normal updates).

8.47.2 Member Function Documentation

LIST_ENTRY()

```
maple_driver_t::LIST_ENTRY (
            maple_driver  )
```

Driver list handle. NOT A FUNCTION!

8.47.3 Field Documentation

attach

```
int(* maple_driver_t::attach) (struct maple_driver *drv, maple_device_t *dev)
```

Device attached callback.

This callback will be called when a new device of this driver is connected to the system.

Parameters

drv	This structure for the driver.
dev	The device that was connected.

Returns

0 on success, <0 on error.

detach

```
void(* maple_driver_t::detach) (struct maple_driver *drv, maple_device_t *dev)
```

Device detached callback.

This callback will be called when a device of this driver is disconnected from the system.

Parameters

drv	This structure for the driver.
dev	The device that was detached.

functions

```
uint32 maple_driver_t::functions
```

One or more MAPLE_FUNCs ORed together.

name

```
const char* maple_driver_t::name
```

The driver name.

periodic

```
void(* maple_driver_t::periodic) (struct maple_driver *drv)
```

Periodic polling callback.

This callback will be called to update the status of connected devices periodically.

Parameters

drv	This structure for the driver.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple.h

8.48 maple_frame_t Struct Reference

Maple frame to be queued for transport.

```
#include <dc/maple.h>
```

Public Member Functions

- TAILQ_ENTRY (maple_frame) frameq
  Send queue handle. NOT A FUNCTION!

Data Fields

- int cmd
  Command (see Maple commands and responses)
- int dst_port
  Destination port.
- int dst_unit
  Destination unit.
- int length
  Data transfer length in 32-bit words.
- volatile int state
  Has this frame been sent / responded to?
- volatile int queued
  Are we on the queue?
- void * send_buf
  The data which will be sent (if any)
- uint8 * recv_buf
  Points into recv_buf_arr, but 32-byte aligned.
- struct maple_device * dev
  Does this belong to a device?
- void(* callback)(struct maple_frame *)
  Response callback.
- uint8 recv_buf_arr [1024+32]
  Response receive area.

8.48.1 Detailed Description

Maple frame to be queued for transport.

Internal representation of a frame to be queued up for sending.

8.48.2 Member Function Documentation

TAILQ_ENTRY()

```
maple_frame_t::TAILQ_ENTRY (
             maple_frame  )
```

Send queue handle. NOT A FUNCTION!

8.48.3 Field Documentation

callback

```
void(* maple_frame_t::callback) (struct maple_frame *)
```

Response callback.

cmd

```
int maple_frame_t::cmd
```

Command (see Maple commands and responses)

dev

```
struct maple_device* maple_frame_t::dev
```

Does this belong to a device?

dst_port

```
int maple_frame_t::dst_port
```

Destination port.

dst_unit

```
int maple_frame_t::dst_unit
```

Destination unit.

length

```
int maple_frame_t::length
```

Data transfer length in 32-bit words.

queued

```
volatile int maple_frame_t::queued
```

Are we on the queue?

recv_buf

```
uint8* maple_frame_t::recv_buf
```

Points into recv_buf_arr, but 32-byte aligned.

recv_buf_arr

```
uint8 maple_frame_t::recv_buf_arr[1024+32]
```

Response receive area.

send_buf

```
void* maple_frame_t::send_buf
```

The data which will be sent (if any)

state

```
volatile int maple_frame_t::state
```

Has this frame been sent / responded to?

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple.h

8.49 maple_port_t Struct Reference

Internal representation of a Maple port.

`#include <dc/maple.h>`

Data Fields

- int port
  Port ID.
- maple_device_t units [MAPLE_UNIT_COUNT]
  Pointers to active units.

8.49.1 Detailed Description

Internal representation of a Maple port.

Each maple port can contain up to 6 devices, the first one of which is always the port itself.

8.49.2 Field Documentation

port

```
int maple_port_t::port
```

Port ID.

units

```
maple_device_t maple_port_t::units[MAPLE_UNIT_COUNT]
```

Pointers to active units.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple.h

8.50 maple_response_t Struct Reference

Maple response frame structure.

`#include <dc/maple.h>`

Data Fields

- int8 response
  Response.
- uint8 dst_addr
  Destination address.
- uint8 src_addr
  Source address.
- uint8 data_len
  Data length (in 32-bit words)
- uint8 data [ ]
  Data (if any)

8.50.1 Detailed Description

Maple response frame structure.

This structure is used to deliver the actual response to a request placed. The data field is where all the interesting stuff will be.

8.50.2 Field Documentation

data

```
uint8 maple_response_t::data[]
```

Data (if any)

data_len

```
uint8 maple_response_t::data_len
```

Data length (in 32-bit words)

dst_addr

```
uint8 maple_response_t::dst_addr
```

Destination address.

response

```
int8 maple_response_t::response
```

Response.

src_addr

```
uint8 maple_response_t::src_addr
```

Source address.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple.h

8.51 maple_state_t Struct Reference

Maple state structure.

```
#include <dc/maple.h>
```

Data Fields

- struct maple_driver_list driver_list

  Maple device driver list. Do not manipulate directly!
- struct maple_frame_queue frame_queue

  Maple frame submission queue. Do not manipulate directly!
- maple_port_t ports [MAPLE_PORT_COUNT]

  Maple device info structure.
- volatile int dma_cntr

  DMA interrupt counter.
- volatile int vbl_cntr

  VBlank interrupt counter.
- uint8 * dma_buffer

  DMA send buffer.
- volatile int dma_in_progress

  Is a DMA running now?
- int detect_port_next

  Next port that will be auto-detected.
- int detect_unit_next

  Next unit which will be auto-detected.
- volatile int detect_wrapped

  Did the detect wrap?
- int vbl_handle

  Our vblank handler handle.
- int gun_port

  The port to read for lightgun status, if any.
- int gun_x

  The horizontal position of the lightgun signal.
- int gun_y

  The vertical position of the lightgun signal.

8.51.1 Detailed Description

Maple state structure.

We put everything in here to keep from polluting the global namespace too much.

8.51.2 Field Documentation

detect_port_next

```
int maple_state_t::detect_port_next
```

Next port that will be auto-detected.

detect_unit_next

```
int maple_state_t::detect_unit_next
```

Next unit which will be auto-detected.

detect_wrapped

```
volatile int maple_state_t::detect_wrapped
```

Did the detect wrap?

dma_buffer

```
uint8* maple_state_t::dma_buffer
```

DMA send buffer.

dma_cntr

```
volatile int maple_state_t::dma_cntr
```

DMA interrupt counter.

dma_in_progress

```
volatile int maple_state_t::dma_in_progress
```

Is a DMA running now?

driver_list

```
struct maple_driver_list maple_state_t::driver_list
```

Maple device driver list. Do not manipulate directly!

frame_queue

```
struct maple_frame_queue maple_state_t::frame_queue
```

Maple frame submission queue. Do not manipulate directly!

gun_port

```
int maple_state_t::gun_port
```

The port to read for lightgun status, if any.

gun_x

```
int maple_state_t::gun_x
```

The horizontal position of the lightgun signal.

gun_y

```
int maple_state_t::gun_y
```

The vertical position of the lightgun signal.

ports

```
maple_port_t maple_state_t::ports[MAPLE_PORT_COUNT]
```

Maple device info structure.

vbl_cntr

```
volatile int maple_state_t::vbl_cntr
```

VBlank interrupt counter.

vbl_handle

```
int maple_state_t::vbl_handle
```

Our vblank handler handle.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple.h

8.52 mmucontext_t Struct Reference

MMU context type.

```
#include <arch/mmu.h>
```

Data Fields

- mmusubcontext_t * sub [MMU_PAGES]
  1024 sub-contexts
- int asid
  Address Space ID.

8.52.1 Detailed Description

MMU context type.

This type is the top-level context that makes up the page table. There is one of these, with 1024 sub-contexts.

8.52.2 Field Documentation

asid

```
int mmucontext_t::asid
```

Address Space ID.

sub

```
mmusubcontext_t* mmucontext_t::sub[MMU_PAGES]
```

1024 sub-contexts

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/arch/mmu.h

8.53 mmupage_t Struct Reference

MMU TLB entry for a single page.

`#include <arch/mmu.h>`

Data Fields

- uint32 physical: 18
  Physical page ID – 18 bits.
- uint32 prkey: 2
  Protection key data – 2 bits.
- uint32 valid: 1
  Valid mapping – 1 bit.
- uint32 shared: 1
  Shared between procs – 1 bit.
- uint32 cache: 1
  Cacheable – 1 bit.
- uint32 dirty: 1
  Dirty – 1 bit.
- uint32 wthru: 1
  Write-thru enable – 1 bit.
- uint32 blank: 7
  Reserved – 7 bits.
- uint32 pteh
  Pre-built PTEH value.
- uint32 ptel
  Pre-built PTEL value.

8.53.1 Detailed Description

MMU TLB entry for a single page.

The TLB entries on the SH4 are a single 32-bit dword in length. We store some other data here too for ease of use.

8.53.2 Field Documentation

blank

`uint32 mmupage_t::blank`

Reserved – 7 bits.

cache

```
uint32 mmupage_t::cache
```

Cacheable – 1 bit.

dirty

```
uint32 mmupage_t::dirty
```

Dirty – 1 bit.

physical

```
uint32 mmupage_t::physical
```

Physical page ID – 18 bits.

prkey

```
uint32 mmupage_t::prkey
```

Protection key data – 2 bits.

pteh

```
uint32 mmupage_t::pteh
```

Pre-built PTEH value.

ptel

```
uint32 mmupage_t::ptel
```

Pre-built PTEL value.

shared

```
uint32 mmupage_t::shared
```

Shared between procs – 1 bit.

valid

```
uint32 mmupage_t::valid
```

Valid mapping – 1 bit.

wthru

```
uint32 mmupage_t::wthru
```

Write-thru enable – 1 bit.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/arch/mmu.h

8.54 mmusubcontext_t Struct Reference

MMU sub-context type.

```
#include <arch/mmu.h>
```

Data Fields

- mmupage_t page [MMU_SUB_PAGES]
  512 page entries

8.54.1 Detailed Description

MMU sub-context type.

We have two-level page tables on SH4, and each sub-context contains 512 entries.

8.54.2 Field Documentation

page

```
mmupage_t mmusubcontext_t::page[MMU_SUB_PAGES]
```

512 page entries

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/arch/mmu.h

8.55 mouse_cond_t Struct Reference

`#include <mouse.h>`

Data Fields

- uint16 buttons
- uint16 dummy1
- int16 dx
- int16 dy
- int16 dz
- uint16 dummy2
- uint32 dummy3
- uint32 dummy4

8.55.1 Field Documentation

buttons

```
uint16 mouse_cond_t::buttons
```

dummy1

```
uint16 mouse_cond_t::dummy1
```

dummy2

```
uint16 mouse_cond_t::dummy2
```

dummy3

```
uint32 mouse_cond_t::dummy3
```

dummy4

```
uint32 mouse_cond_t::dummy4
```

dx

```
int16 mouse_cond_t::dx
```

dy

```
int16 mouse_cond_t::dy
```

dz

```
int16 mouse_cond_t::dz
```

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple/mouse.h

8.56 mouse_state_t Struct Reference

Mouse status structure.

```
#include <dc/maple/mouse.h>
```

Data Fields

- uint32 buttons
  Buttons pressed bitmask.
- int dx
  X movement value.
- int dy
  Y movement value.
- int dz
  Z movement value.

8.56.1 Detailed Description

Mouse status structure.

This structure contains information about the status of the mouse device, and can be fetched with maple_dev_status().

8.56.2 Field Documentation

buttons

```
uint32 mouse_state_t::buttons
```

Buttons pressed bitmask.

See also

Mouse button codes

dx

```
int mouse_state_t::dx
```

X movement value.

dy

```
int mouse_state_t::dy
```

Y movement value.

dz

```
int mouse_state_t::dz
```

Z movement value.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple/mouse.h

8.57 mutex_t Struct Reference

Mutual exclusion lock type.

```
#include <kos/mutex.h>
```

Data Fields

- int type
- int dynamic
- kthread_t * holder
- int count

8.57.1 Detailed Description

Mutual exclusion lock type.

All members of this structure should be considered to be private. It is unsafe to change anything in here yourself.

8.57.2 Field Documentation

count

```
int mutex_t::count
```

dynamic

```
int mutex_t::dynamic
```

holder

```
kthread_t* mutex_t::holder
```

type

```
int mutex_t::type
```

The documentation for this struct was generated from the following file:

- include/kos/mutex.h

8.58 net_ipv4_stats_t Struct Reference

IPv4 statistics structure.

```
#include <kos/net.h>
```

Data Fields

- uint32 pkt_sent
- uint32 pkt_send_failed

  Packets sent out successfully.
- uint32 pkt_recv

  Packets that failed to send.
- uint32 pkt_recv_bad_size

  Packets received successfully.
- uint32 pkt_recv_bad_chksum

  Packets of a bad size.
- uint32 pkt_recv_bad_proto

  Packets with a bad checksum.

8.58.1 Detailed Description

IPv4 statistics structure.

This structure holds some basic statistics about the IPv4 layer of the stack, and can be retrieved with the appropriate function.

8.58.2 Field Documentation

pkt_recv

```
uint32 net_ipv4_stats_t::pkt_recv
```

Packets that failed to send.

pkt_recv_bad_chksum

```
uint32 net_ipv4_stats_t::pkt_recv_bad_chksum
```

Packets of a bad size.

pkt_recv_bad_proto

```
uint32 net_ipv4_stats_t::pkt_recv_bad_proto
```

Packets with a bad checksum.

pkt_recv_bad_size

```
uint32 net_ipv4_stats_t::pkt_recv_bad_size
```

Packets received successfully.

pkt_send_failed

```
uint32 net_ipv4_stats_t::pkt_send_failed
```

Packets sent out successfully.

pkt_sent

```
uint32 net_ipv4_stats_t::pkt_sent
```

The documentation for this struct was generated from the following file:

- include/kos/net.h

8.59 net_ipv6_stats_t Struct Reference

IPv6 statistics structure.

```
#include <kos/net.h>
```

Data Fields

- uint32 pkt_sent
  Packets sent out successfully.
- uint32 pkt_send_failed
  Packets that failed to send.
- uint32 pkt_recv
  Packets received successfully.
- uint32 pkt_recv_bad_size
  Packets of a bad size.
- uint32 pkt_recv_bad_proto
  Packets with an unknown proto.
- uint32 pkt_recv_bad_ext
  Packets with an unknown hdr.

8.59.1 Detailed Description

IPv6 statistics structure.

This structure holds some basic statistics about the IPv6 layer of the stack, and can be retrieved with the appropriate function.

8.59.2 Field Documentation

pkt_recv

```
uint32 net_ipv6_stats_t::pkt_recv
```

Packets received successfully.

pkt_recv_bad_ext

```
uint32 net_ipv6_stats_t::pkt_recv_bad_ext
```

Packets with an unknown hdr.

pkt_recv_bad_proto

```
uint32 net_ipv6_stats_t::pkt_recv_bad_proto
```

Packets with an unknown proto.

pkt_recv_bad_size

```
uint32 net_ipv6_stats_t::pkt_recv_bad_size
```

Packets of a bad size.

pkt_send_failed

```
uint32 net_ipv6_stats_t::pkt_send_failed
```

Packets that failed to send.

pkt_sent

```
uint32 net_ipv6_stats_t::pkt_sent
```

Packets sent out successfully.

The documentation for this struct was generated from the following file:

- include/kos/net.h

8.60 net_socket_t Struct Reference

Internal representation of a socket for fs_socket.

```
#include <kos/fs_socket.h>
```

Data Fields

- file_t fd

  File handle from the VFS layer.
- struct fs_socket_proto * protocol

  The protocol handler for this socket.
- void * data

  Protocol-specific data.

8.60.1 Detailed Description

Internal representation of a socket for fs_socket.

This structure is the internal representation of a socket "file" that is used within fs_socket. A normal user will never deal with this structure directly (only protocol handlers and fs_socket itself ever sees this structure directly).

8.60.2 Field Documentation

data

```
void* net_socket_t::data
```

Protocol-specific data.

fd

```
file_t net_socket_t::fd
```

File handle from the VFS layer.

protocol

```
struct fs_socket_proto* net_socket_t::protocol
```

The protocol handler for this socket.

The documentation for this struct was generated from the following file:

- include/kos/fs_socket.h

8.61 net_udp_stats_t Struct Reference

UDP statistics structure.

```
#include <kos/net.h>
```

Data Fields

- uint32 pkt_sent
  Packets sent out successfully.
- uint32 pkt_send_failed
  Packets that failed to send.
- uint32 pkt_recv
  Packets received successfully.
- uint32 pkt_recv_bad_size
  Packets of a bad size.
- uint32 pkt_recv_bad_chksum
  Packets with a bad checksum.
- uint32 pkt_recv_no_sock
  Packets with to a closed port.

8.61.1 Detailed Description

UDP statistics structure.

This structure holds some basic statistics about the UDP layer of the stack, and can be retrieved with the appropriate function.

8.61.2 Field Documentation

pkt_recv

```
uint32 net_udp_stats_t::pkt_recv
```

Packets received successfully.

pkt_recv_bad_chksum

```
uint32 net_udp_stats_t::pkt_recv_bad_chksum
```

Packets with a bad checksum.

pkt_recv_bad_size

```
uint32 net_udp_stats_t::pkt_recv_bad_size
```

Packets of a bad size.

pkt_recv_no_sock

```
uint32 net_udp_stats_t::pkt_recv_no_sock
```

Packets with to a closed port.

pkt_send_failed

```
uint32 net_udp_stats_t::pkt_send_failed
```

Packets that failed to send.

pkt_sent

```
uint32 net_udp_stats_t::pkt_sent
```

Packets sent out successfully.

The documentation for this struct was generated from the following file:

- include/kos/net.h

8.62 netcfg_t Struct Reference

Network configuration information.

```
#include <kos/netcfg.h>
```

Data Fields

- int src
  Where was this configuration read from?
- int method
  How should the network be configured?
- uint32 ip
  IPv4 address of the console.
- uint32 gateway
  IPv4 address of the gateway/router.
- uint32 netmask
  Network mask for the local net.
- uint32 broadcast
  Broadcast address for the local net.
- uint32 dns [2]
  IPv4 address of the DNS servers.

- char hostname [64]
  DNS/DHCP hostname.
- char email [64]
  E-Mail address.
- char smtp [64]
  SMTP server address.
- char pop3 [64]
  POP3 server address.
- char pop3_login [64]
  POP3 server username.
- char pop3_passwd [64]
  POP3 server password.
- char proxy_host [64]
  Proxy server address.
- int proxy_port
  Proxy server port.
- char ppp_login [64]
  PPP Username.
- char ppp_passwd [64]
  PPP Password.
- char driver [64]
  Driver program filename (if any).

8.62.1 Detailed Description

Network configuration information.

This structure contains information about the network configuration of the system, as set up by the user.

8.62.2 Field Documentation

broadcast

```
uint32 netcfg_t::broadcast
```

Broadcast address for the local net.

dns

```
uint32 netcfg_t::dns[2]
```

IPv4 address of the DNS servers.

driver

```
char netcfg_t::driver[64]
```

Driver program filename (if any).

email

```
char netcfg_t::email[64]
```

E-Mail address.

gateway

```
uint32 netcfg_t::gateway
```

IPv4 address of the gateway/router.

hostname

```
char netcfg_t::hostname[64]
```

DNS/DHCP hostname.

ip

```
uint32 netcfg_t::ip
```

IPv4 address of the console.

method

```
int netcfg_t::method
```

How should the network be configured?

See also

Network connection methods

netmask

```
uint32 netcfg_t::netmask
```

Network mask for the local net.

pop3

```
char netcfg_t::pop3[64]
```

POP3 server address.

pop3_login

```
char netcfg_t::pop3_login[64]
```

POP3 server username.

pop3_passwd

```
char netcfg_t::pop3_passwd[64]
```

POP3 server password.

ppp_login

```
char netcfg_t::ppp_login[64]
```

PPP Username.

ppp_passwd

```
char netcfg_t::ppp_passwd[64]
```

PPP Password.

proxy_host

```
char netcfg_t::proxy_host[64]
```

Proxy server address.

proxy_port

```
int netcfg_t::proxy_port
```

Proxy server port.

smtp

```
char netcfg_t::smtp[64]
```

SMTP server address.

src

```
int netcfg_t::src
```

Where was this configuration read from?

See also

Network configuration sources

The documentation for this struct was generated from the following file:

- addons/include/kos/netcfg.h

8.63 netif_t Struct Reference

Structure describing one usable network device.

```
#include <kos/net.h>
```

Public Member Functions

- LIST_ENTRY (knetif) if_list
  Device list handle (not a function!)

Data Fields

- const char * name

  Device name ("bba", "la", etc)
- const char * descr

  Long description of the device.
- int index

  Unit index (starts at zero and counts upwards for multiple network devices of the same type)
- uint32 dev_id

  Internal device ID (for whatever the driver wants)
- uint32 flags

  Interface flags.
- uint8 mac_addr [6]

  The device's MAC address.
- uint8 ip_addr [4]

  The device's IP address (if any)
- uint8 netmask [4]

  The device's netmask.
- uint8 gateway [4]

  The device's gateway's IP address.
- uint8 broadcast [4]

  The device's broadcast address.
- uint8 dns [4]

  The device's DNS server address.
- int mtu

  The device's MTU.
- struct in6_addr ip6_lladdr

  The device's Link-local IPv6 address.
- struct in6_addr * ip6_addrs

  Any further IPv6 addresses the device has. The first address in this list will always be used, unless otherwise specified.
- int ip6_addr_count
- struct in6_addr ip6_gateway

  The device's gateway's IPv6 address.
- uint32 mtu6

  Default MTU over IPv6.
- int hop_limit

  Default hop limit over IPv6.
- int(* if_detect)(struct knetif *self)

  Attempt to detect the device.
- int(* if_init)(struct knetif *self)

  Initialize the device.
- int(* if_shutdown)(struct knetif *self)

  Shutdown the device.
- int(* if_start)(struct knetif *self)

  Start the device (after init or stop).
- int(* if_stop)(struct knetif *self)

  Stop (hibernate) the device.

- int(∗ if_tx)(struct knetif ∗self, const uint8 ∗data, int len, int blocking)

  Queue a packet for transmission.
- int(∗ if_tx_commit)(struct knetif ∗self)

  Commit any queued output packets.
- int(∗ if_rx_poll)(struct knetif ∗self)

  Poll for queued receive packets, if neccessary.
- int(∗ if_set_flags)(struct knetif ∗self, uint32 flags_and, uint32 flags_or)

  Set flags; you should generally manipulate flags through here so that the driver gets a chance to act on the info.
- int(∗ if_set_mc)(struct knetif ∗self, const uint8 ∗list, int count)

  Set the device's multicast list.

8.63.1 Detailed Description

Structure describing one usable network device.

Each usable network device should have one of these describing it. These must be registered to the network layer before the device is useable.

8.63.2 Member Function Documentation

LIST_ENTRY()

```
netif_t::LIST_ENTRY (
            knetif  )
```

Device list handle (not a function!)

8.63.3 Field Documentation

broadcast

```
uint8 netif_t::broadcast[4]
```

The device's broadcast address.

descr

```
const char* netif_t::descr
```

Long description of the device.

dev_id

```
uint32 netif_t::dev_id
```

Internal device ID (for whatever the driver wants)

dns

```
uint8 netif_t::dns[4]
```

The device's DNS server address.

flags

```
uint32 netif_t::flags
```

Interface flags.

gateway

```
uint8 netif_t::gateway[4]
```

The device's gateway's IP address.

hop_limit

```
int netif_t::hop_limit
```

Default hop limit over IPv6.

if_detect

```
int(* netif_t::if_detect) (struct knetif *self)
```

Attempt to detect the device.

Parameters

self	The network device in question.

Returns

0 on success, <0 on failure.

if_init

```
int(* netif_t::if_init) (struct knetif *self)
```

Initialize the device.

Parameters

self	The network device in question.

Returns

0 on success, <0 on failure.

if_rx_poll

```
int(* netif_t::if_rx_poll) (struct knetif *self)
```

Poll for queued receive packets, if neccessary.

Parameters

self	The network device in question.

Returns

0 on success, <0 on failure.

if_set_flags

```
int(* netif_t::if_set_flags) (struct knetif *self, uint32 flags_and, uint32 flags_or)
```

Set flags; you should generally manipulate flags through here so that the driver gets a chance to act on the info.

Parameters

self	The network device in question.
flags_and	Bitmask to and with the flags.
flags_or	Bitmask to or with the flags.

if_set_mc

```
int(* netif_t::if_set_mc) (struct knetif *self, const uint8 *list, int count)
```

Set the device's multicast list.

Parameters

self	The network device in question.
list	The list of MAC addresses (6 * count bytes).
count	The number of addresses in list.

if_shutdown

```
int(* netif_t::if_shutdown) (struct knetif *self)
```

Shutdown the device.

Parameters

self	The network device in question.

Returns

0 on success, <0 on failure.

if_start

```
int(* netif_t::if_start) (struct knetif *self)
```

Start the device (after init or stop).

Parameters

self	The network device in question.

Returns

0 on success, <0 on failure.

if_stop

```
int(* netif_t::if_stop) (struct knetif *self)
```

Stop (hibernate) the device.

Parameters

self	The network device in question.

Returns

0 on success, <0 on failure

if_tx

```
int(* netif_t::if_tx) (struct knetif *self, const uint8 *data, int len, int blocking)
```

Queue a packet for transmission.

Parameters

self	The network device in question.
data	The packet to transmit.
len	The length of the packet in bytes.
blocking	1 if we should block if needed, 0 otherwise.

Return values

NETIF_TX_OK	On success.
NETIF_TX_ERROR	On general failure.
NETIF_TX_AGAIN	If non-blocking and we must block to send.

if_tx_commit

```
int(* netif_t::if_tx_commit) (struct knetif *self)
```

Commit any queued output packets.

Parameters

self	The network device in question.

Returns

0 on success, <0 on failure.

index

```
int netif_t::index
```

Unit index (starts at zero and counts upwards for multiple network devices of the same type)

ip6_addr_count

```
int netif_t::ip6_addr_count
```

ip6_addrs

```
struct in6_addr* netif_t::ip6_addrs
```

Any further IPv6 addresses the device has. The first address in this list will always be used, unless otherwise specified.

ip6_gateway

```
struct in6_addr netif_t::ip6_gateway
```

The device's gateway's IPv6 address.

ip6_lladdr

```
struct in6_addr netif_t::ip6_lladdr
```

The device's Link-local IPv6 address.

ip_addr

```
uint8 netif_t::ip_addr[4]
```

The device's IP address (if any)

mac_addr

```
uint8 netif_t::mac_addr[6]
```

The device's MAC address.

mtu

```
int netif_t::mtu
```

The device's MTU.

mtu6

```
uint32 netif_t::mtu6
```

Default MTU over IPv6.

name

```
const char* netif_t::name
```

Device name ("bba", "la", etc)

netmask

```
uint8 netif_t::netmask[4]
```

The device's netmask.

The documentation for this struct was generated from the following file:

- include/kos/net.h

8.64 nmmgr_handler_t Struct Reference

Name handler interface.

```
#include <kos/nmmgr.h>
```

Public Member Functions

- LIST_ENTRY (nmmgr_handler) list_ent

Data Fields

- char pathname [NAME_MAX]
- int pid
- uint32 version
- uint32 flags
- uint32 type

8.64.1 Detailed Description

Name handler interface.

Every name handler must begin its information structures with this header. If the handler conforms to some well-defined interface (such as a VFS), then the struct must more specifically be of that type.

8.64.2 Member Function Documentation

LIST_ENTRY()

```
nmmgr_handler_t::LIST_ENTRY (
            nmmgr_handler  )
```

8.64.3 Field Documentation

flags

```
uint32 nmmgr_handler_t::flags
```

pathname

```
char nmmgr_handler_t::pathname[NAME_MAX]
```

pid

```
int nmmgr_handler_t::pid
```

type

```
uint32 nmmgr_handler_t::type
```

version

```
uint32 nmmgr_handler_t::version
```

The documentation for this struct was generated from the following file:

- include/kos/nmmgr.h

8.65 pollfd Struct Reference

Structure representing a single file descriptor used by poll().

```
#include <poll.h>
```

Data Fields

- int fd

  The file descriptor in question.
- short events

  Events to poll for on input.
- short revents

  Events signalled for output.

8.65.1 Detailed Description

Structure representing a single file descriptor used by poll().

8.65.2 Field Documentation

events

```
short pollfd::events
```

Events to poll for on input.

fd

```
int pollfd::fd
```

The file descriptor in question.

revents

```
short pollfd::revents
```

Events signalled for output.

The documentation for this struct was generated from the following file:

- include/poll.h

8.66 ppp_device_t Struct Reference

PPP device structure.

`#include <ppp/ppp.h>`

Data Fields

- const char * name

  Device name ("modem", "scif", etc).
- const char * descr

  Long description of the device.
- int index

  Unit index (starts at zero and counts upwards for multiple network devices of the same type).
- uint32_t flags

  Device flags. The lowest 16 bits of this value are reserved for use by libppp. You are free to use the other 16 bits as you see fit in your driver.
- void * privdata

  Private, device-specific data. This can be used for whatever the driver deems fit. The PPP code won't touch this data at all. Set to NULL if you don't need anything here.
- int(* detect)(struct ppp_device *self)

  Attempt to detect the device.
- int(* init)(struct ppp_device *self)

  Initialize the device.
- int(* shutdown)(struct ppp_device *self)

  Shutdown the device.
- int(* tx)(struct ppp_device *self, const uint8_t *data, size_t len, uint32_t flags)

  Transmit data on the device.
- const uint8_t *(* rx)(struct ppp_device *self, ssize_t *out_len)

  Poll for queued receive data.

8.66.1 Detailed Description

PPP device structure.

This structure defines a basic output device for PPP packets. This structure is largely modeled after netif_t from the main network stack, with a bit of functionality removed that is irrelevant for PPP.

Note that we only allow one device and one connection in this library.

8.66.2 Field Documentation

descr

```
const char* ppp_device_t::descr
```

Long description of the device.

detect

```
int(* ppp_device_t::detect) (struct ppp_device *self)
```

Attempt to detect the device.

Parameters

self	The network device in question.

Returns

0 on success, <0 on failure.

flags

```
uint32_t ppp_device_t::flags
```

Device flags. The lowest 16 bits of this value are reserved for use by libppp. You are free to use the other 16 bits as you see fit in your driver.

index

```
int ppp_device_t::index
```

Unit index (starts at zero and counts upwards for multiple network devices of the same type).

init

```
int(* ppp_device_t::init) (struct ppp_device *self)
```

Initialize the device.

Parameters

self	The network device in question.

Returns

0 on success, <0 on failure.

name

```
const char* ppp_device_t::name
```

Device name ("modem", "scif", etc).

privdata

```
void* ppp_device_t::privdata
```

Private, device-specific data. This can be used for whatever the driver deems fit. The PPP code won't touch this data at all. Set to NULL if you don't need anything here.

rx

```
const uint8_t *(* ppp_device_t::rx) (struct ppp_device *self, ssize_t *out_len)
```

Poll for queued receive data.

This function will be called periodically by a thread to check the device for any new incoming data.

Parameters

self	The network device in question.

Returns

A pointer to the received data on success. NULL on failure or if no data is waiting.

shutdown

```
int(* ppp_device_t::shutdown) (struct ppp_device *self)
```

Shutdown the device.

Parameters

self	The network device in question.

Returns

0 on success, $<$0 on failure.

tx

```
int(* ppp_device_t::tx) (struct ppp_device *self, const uint8_t *data, size_t len, uint32_t flags)
```

Transmit data on the device.

This function will be called periodically to transmit data on the underlying device. The data passed in may not necessarily be a whole packet (check the flags to see what's being passed in).

Parameters

self	The network device in question.
data	The data to transmit.
len	The length of the data to transmit in bytes.
flags	Flags to describe what data is being sent in.

Returns

0 on success, <0 on failure.

The documentation for this struct was generated from the following file:

- addons/include/ppp/ppp.h

8.67 ppp_protocol_t Struct Reference

PPP Protocol structure.

`#include <ppp/ppp.h>`

Public Member Functions

- TAILQ_ENTRY (ppp_proto) entry
  Protocol list entry (not a function!).

Data Fields

- const char * name
  Protocol name ("lcp", "pap", etc).
- uint16_t code
  Protocol code.
- void * privdata
  Private data (if any).
- int(* init)(struct ppp_proto *self)
  Initialization function.
- int(* shutdown)(struct ppp_proto *self)
  Shutdown function.
- int(* input)(struct ppp_proto *self, const uint8_t *buf, size_t len)
  Protocol packet input function.
- void(* enter_phase)(struct ppp_proto *self, int oldp, int newp)
  Notify the protocol of a PPP phase change.
- void(* check_timeouts)(struct ppp_proto *self, uint64_t tm)
  Check timeouts for resending packets.

8.67.1 Detailed Description

PPP Protocol structure.

Each protocol that the PPP library can handle must have one of these registered. All protocols should be registered BEFORE attempting to actually establish a PPP session to ensure that each protocol can be used in the setup of the connection as needed.

8.67.2 Member Function Documentation

TAILQ_ENTRY()

```
ppp_protocol_t::TAILQ_ENTRY (
             ppp_proto  )
```

Protocol list entry (not a function!).

8.67.3 Field Documentation

check_timeouts

```
void(* ppp_protocol_t::check_timeouts) (struct ppp_proto *self, uint64_t tm)
```

Check timeouts for resending packets.

This function will be called periodically to allow the protocol to check any resend timers that it might have responsibility for.

Parameters

self	The protocol structure for this protocol.
tm	The current system time for checking timeouts against (in milliseconds since system startup).

code

```
uint16_t ppp_protocol_t::code
```

Protocol code.

enter_phase

```
void(* ppp_protocol_t::enter_phase) (struct ppp_proto *self, int oldp, int newp)
```

Notify the protocol of a PPP phase change.

This function will be called by the PPP automaton any time that a phase change is initiated. This is often used for starting up a protocol when appropriate to do so (for instance, LCP uses this to begin negotiating configuration options with the peer when the establish phase is entered by the automaton).

Parameters

self	The protocol structure for this protocol.
oldp	The old phase (the one the automaton is leaving).
newp	The new phase.

See also

PPP automaton phases

init

```
int(* ppp_protocol_t::init) (struct ppp_proto *self)
```

Initialization function.

Parameters

self	The protocol structure for this protocol.

Returns

0 on success, <0 on failure.

Note

Set to NULL if this is not needed in the protocol.

input

```
int(* ppp_protocol_t::input) (struct ppp_proto *self, const uint8_t *buf, size_t len)
```

Protocol packet input function.

This function will be called for each packet delivered to the specified protocol.

Parameters

self	The protocol structure for this protocol.
pkt	The packet being delivered.
len	The length of the packet in bytes.

Returns

0 on success, <0 on failure.

name

```
const char* ppp_protocol_t::name
```

Protocol name ("lcp", "pap", etc).

privdata

```
void* ppp_protocol_t::privdata
```

Private data (if any).

shutdown

```
int(* ppp_protocol_t::shutdown) (struct ppp_proto *self)
```

Shutdown function.

This function should perform any protocol-specific shutdown actions and unregister the protocol from the PPP protocol list.

Parameters

self	The protocol structure for this protocol.

Returns

0 on success, <0 on failure.

The documentation for this struct was generated from the following file:

- addons/include/ppp/ppp.h

8.68 pthread_attr_t Struct Reference

POSIX thread attributes.

```
#include <sys/sched.h>
```

8.68.1 Detailed Description

POSIX thread attributes.

Not implemented in KOS.

The documentation for this struct was generated from the following file:

- include/sys/sched.h

8.69 pthread_condattr_t Struct Reference

POSIX condition variable attributes.

`#include <sys/sched.h>`

8.69.1 Detailed Description

POSIX condition variable attributes.

Not implemented in KOS.

The documentation for this struct was generated from the following file:

- include/sys/sched.h

8.70 pthread_mutexattr_t Struct Reference

POSIX mutex attributes.

`#include <sys/sched.h>`

8.70.1 Detailed Description

POSIX mutex attributes.

Not implemented in KOS.

The documentation for this struct was generated from the following file:

- include/sys/sched.h

8.71 purupuru_effect_t Struct Reference

Effect generation structure.

`#include <purupuru.h>`

Data Fields

- uint8 duration
  The duration of the effect. No idea on units...
- uint8 effect2
  2nd effect field.
- uint8 effect1
  1st effect field.
- uint8 special
  Special effects field.

8.71.1 Detailed Description

Effect generation structure.

This structure is used for convenience to send an effect to the jump pack. This, along with the various macros in this file can give a slightly better idea of the effect being generated than using the raw values.

8.71.2 Field Documentation

duration

`uint8 purupuru_effect_t::duration`

The duration of the effect. No idea on units...

effect1

`uint8 purupuru_effect_t::effect1`

1st effect field.

effect2

`uint8 purupuru_effect_t::effect2`

2nd effect field.

special

```
uint8 purupuru_effect_t::special
```

Special effects field.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple/purupuru.h

8.72 pvr_init_params_t Struct Reference

PVR initialization structure.

```
#include <dc/pvr.h>
```

Data Fields

- int opb_sizes [5]
  Bin sizes.
- int vertex_buf_size
  Vertex buffer size (should be a nice round number)
- int dma_enabled
  Enable vertex DMA?
- int fsaa_enabled
  Enable horizontal scaling?
- int autosort_disabled
  Disable translucent polygon autosort?
- int opb_overflow_count
  OPB Overflow Count.

8.72.1 Detailed Description

PVR initialization structure.

This structure defines how the PVR initializes various parts of the system, including the primitive bin sizes, the vertex buffer size, and whether vertex DMA will be enabled.

You essentially fill one of these in, and pass it to pvr_init().

8.72.2 Field Documentation

autosort_disabled

```
int pvr_init_params_t::autosort_disabled
```

Disable translucent polygon autosort?

Set to non-zero to disable translucent polygon autosorting. By enabling this setting, the PVR acts more like a traditional Z-buffered system when rendering translucent polygons, meaning you must pre-sort them yourself if you want them to appear in the right order.

dma_enabled

```
int pvr_init_params_t::dma_enabled
```

Enable vertex DMA?

Set to non-zero if we want to enable vertex DMA mode. Note that if this is set, then *all* enabled lists need to have a vertex buffer assigned, even if you never use that list for anything.

fsaa_enabled

```
int pvr_init_params_t::fsaa_enabled
```

Enable horizontal scaling?

Set to non-zero if horizontal scaling is to be enabled. By enabling this setting and stretching your image to double the native screen width, you can get horizontal full-screen anti-aliasing.

opb_overflow_count

```
int pvr_init_params_t::opb_overflow_count
```

OPB Overflow Count.

Preallocates this many extra OPBs (sets of tile bins), allowing the PVR to use the extra space when there's too much geometry in the first OPB.

Increasing this value can eliminate artifacts where pieces of geometry flicker in and out of existence along the tile boundaries.

opb_sizes

```
int pvr_init_params_t::opb_sizes[5]
```

Bin sizes.

The bins go in the following order: opaque polygons, opaque modifiers, translucent polygons, translucent modifiers, punch-thrus

vertex_buf_size

```
int pvr_init_params_t::vertex_buf_size
```

Vertex buffer size (should be a nice round number)

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.73 pvr_mod_hdr_t Struct Reference

Modifier volume header.

```
#include <dc/pvr.h>
```

Data Fields

- uint32_t cmd
  TA command.
- uint32_t mode1
  Parameter word 1.
- uint32_t d1
  Dummy value.
- uint32_t d2
  Dummy value.
- uint32_t d3
  Dummy value.
- uint32_t d4
  Dummy value.
- uint32_t d5
  Dummy value.
- uint32_t d6
  Dummy value.

8.73.1 Detailed Description

Modifier volume header.

This is the header that should be submitted when dealing with setting a modifier volume.

8.73.2 Field Documentation

cmd

```
uint32_t pvr_mod_hdr_t::cmd
```

TA command.

d1

```
uint32_t pvr_mod_hdr_t::d1
```

Dummy value.

d2

```
uint32_t pvr_mod_hdr_t::d2
```

Dummy value.

d3

```
uint32_t pvr_mod_hdr_t::d3
```

Dummy value.

d4

```
uint32_t pvr_mod_hdr_t::d4
```

Dummy value.

d5

```
uint32_t pvr_mod_hdr_t::d5
```

Dummy value.

d6

```
uint32_t pvr_mod_hdr_t::d6
```

Dummy value.

mode1

```
uint32_t pvr_mod_hdr_t::mode1
```

Parameter word 1.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.74 pvr_modifier_vol_t Struct Reference

PVR vertex type: Modifier volume.

```
#include <pvr.h>
```

Data Fields

- uint32_t flags
  TA command (vertex flags)
- float ax
  First X coordinate.
- float ay
  First Y coordinate.
- float az
  First Z coordinate.
- float bx
  Second X coordinate.
- float by
  Second Y coordinate.
- float bz
  Second Z coordinate.
- float cx
  Third X coordinate.
- float cy
  Third Y coordinate.
- float cz
  Third Z coordinate.
- uint32_t d1
  Dummy value.
- uint32_t d2
  Dummy value.
- uint32_t d3
  Dummy value.
- uint32_t d4
  Dummy value.
- uint32_t d5
  Dummy value.
- uint32_t d6
  Dummy value.

8.74.1 Detailed Description

PVR vertex type: Modifier volume.

This vertex type is to be used with the modifier volume header to specify triangular modifier areas.

8.74.2 Field Documentation

ax

```
float pvr_modifier_vol_t::ax
```

First X coordinate.

ay

```
float pvr_modifier_vol_t::ay
```

First Y coordinate.

az

```
float pvr_modifier_vol_t::az
```

First Z coordinate.

bx

```
float pvr_modifier_vol_t::bx
```

Second X coordinate.

by

```
float pvr_modifier_vol_t::by
```

Second Y coordinate.

bz

```
float pvr_modifier_vol_t::bz
```

Second Z coordinate.

cx

```
float pvr_modifier_vol_t::cx
```

Third X coordinate.

cy

```
float pvr_modifier_vol_t::cy
```

Third Y coordinate.

cz

```
float pvr_modifier_vol_t::cz
```

Third Z coordinate.

d1

```
uint32_t pvr_modifier_vol_t::d1
```

Dummy value.

d2

```
uint32_t pvr_modifier_vol_t::d2
```

Dummy value.

d3

```
uint32_t pvr_modifier_vol_t::d3
```

Dummy value.

d4

```
uint32_t pvr_modifier_vol_t::d4
```

Dummy value.

d5

```
uint32_t pvr_modifier_vol_t::d5
```

Dummy value.

d6

```
uint32_t pvr_modifier_vol_t::d6
```

Dummy value.

flags

```
uint32_t pvr_modifier_vol_t::flags
```

TA command (vertex flags)

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.75 pvr_poly_cxt_t Struct Reference

PVR polygon context.

```
#include <dc/pvr.h>
```

Data Fields

- int list_type
  Primitive list.
- struct {
  int alpha
  Enable or disable alpha outside modifier.
  int shading
  Shading type.
  int fog_type
  Fog type outside modifier.
  int culling
  Culling mode.
  int color_clamp
  Color clamp enable/disable outside modifier.
  int clip_mode
  Clipping mode.
  int modifier_mode

Modifier mode.

int specular

Offset color enable/disable outside modifier.

int alpha2

Enable/disable alpha inside modifier.

int fog_type2

Fog type inside modifier.

int color_clamp2

Color clamp enable/disable inside modifier.

} gen

General parameters.

- struct {

int src

Source blending mode outside modifier.

int dst

Dest blending mode outside modifier.

int src_enable

Source blending enable outside modifier.

int dst_enable

Dest blending enable outside modifier.

int src2

Source blending mode inside modifier.

int dst2

Dest blending mode inside modifier.

int src_enable2

Source blending mode inside modifier.

int dst_enable2

Dest blending mode inside modifier.

} blend

Blending parameters.

- struct {

int color

Color format in vertex.

int uv

U/V data format in vertex.

int modifier

Enable or disable modifier effect.

} fmt

Format control.

- struct {

int comparison

Depth comparison mode.

int write

Enable or disable depth writes.

} depth

Depth comparison/write modes.

- struct {

int enable

Enable/disable texturing.

int filter
Filtering mode.
int mipmap
Enable/disable mipmaps.
int mipmap_bias
Mipmap bias.
int uv_flip
Enable/disable U/V flipping.
int uv_clamp
Enable/disable U/V clamping.
int alpha
Enable/disable texture alpha.
int env
Texture color contribution.
int width
Texture width (requires a power of 2)
int height
Texture height (requires a power of 2)
int format
Texture format.
pvr_ptr_t base
Texture pointer.
} txr

Texturing params outside modifier.

- struct {
  int enable
  Enable/disable texturing.
  int filter
  Filtering mode.
  int mipmap
  Enable/disable mipmaps.
  int mipmap_bias
  Mipmap bias.
  int uv_flip
  Enable/disable U/V flipping.
  int uv_clamp
  Enable/disable U/V clamping.
  int alpha
  Enable/disable texture alpha.
  int env
  Texture color contribution.
  int width
  Texture width (requires a power of 2)
  int height
  Texture height (requires a power of 2)
  int format
  Texture format.
  pvr_ptr_t base
  Texture pointer.
  } txr2

Texturing params inside modifier.

8.75.1 Detailed Description

PVR polygon context.

You should use this more human readable format for specifying your polygon contexts, and then compile them into polygon headers when you are ready to start using them.

This has embedded structures in it for two reasons; the first reason is to make it easier for me to add new stuff later without breaking existing code. The second reason is to make it more readable and usable.

Unfortunately, it seems that Doxygen chokes up a little bit on this structure, and others like it. The documentation should still be mostly understandable though...

8.75.2 Field Documentation

alpha

```
int pvr_poly_cxt_t::alpha
```

Enable or disable alpha outside modifier.

Enable/disable texture alpha.

See also

Enable or disable alpha blending

Enable or disable texture alpha blending

alpha2

```
int pvr_poly_cxt_t::alpha2
```

Enable/disable alpha inside modifier.

See also

Enable or disable alpha blending

base

```
pvr_ptr_t pvr_poly_cxt_t::base
```

Texture pointer.

[struct]

```
struct { ... } pvr_poly_cxt_t::blend
```

Blending parameters.

clip_mode

```
int pvr_poly_cxt_t::clip_mode
```

Clipping mode.

See also

PVR clipping modes

color

```
int pvr_poly_cxt_t::color
```

Color format in vertex.

See also

PVR vertex color formats

color_clamp

```
int pvr_poly_cxt_t::color_clamp
```

Color clamp enable/disable outside modifier.

See also

Enable or disable color clamping

color_clamp2

```
int pvr_poly_cxt_t::color_clamp2
```

Color clamp enable/disable inside modifier.

See also

Enable or disable color clamping

comparison

```
int pvr_poly_cxt_t::comparison
```

Depth comparison mode.

See also

PVR depth comparison modes

culling

```
int pvr_poly_cxt_t::culling
```

Culling mode.

See also

PVR culling modes

[struct]

```
struct { ... } pvr_poly_cxt_t::depth
```

Depth comparison/write modes.

dst

```
int pvr_poly_cxt_t::dst
```

Dest blending mode outside modifier.

See also

PVR blending modes

dst2

```
int pvr_poly_cxt_t::dst2
```

Dest blending mode inside modifier.

See also

PVR blending modes

dst_enable

```
int pvr_poly_cxt_t::dst_enable
```

Dest blending enable outside modifier.

See also

Enable or disable blending

dst_enable2

```
int pvr_poly_cxt_t::dst_enable2
```

Dest blending mode inside modifier.

See also

Enable or disable blending

enable

```
int pvr_poly_cxt_t::enable
```

Enable/disable texturing.

See also

Enable or disable texturing on polygons

env

```
int pvr_poly_cxt_t::env
```

Texture color contribution.

See also

Texture color calculation modes

filter

```
int pvr_poly_cxt_t::filter
```

Filtering mode.

See also

PVR texture sampling modes

[struct]

```
struct { ... } pvr_poly_cxt_t::fmt
```

Format control.

fog_type

```
int pvr_poly_cxt_t::fog_type
```

Fog type outside modifier.

See also

PVR fog modes

fog_type2

```
int pvr_poly_cxt_t::fog_type2
```

Fog type inside modifier.

See also

PVR fog modes

format

```
int pvr_poly_cxt_t::format
```

Texture format.

See also

PVR texture formats

[struct]

```
struct { ... } pvr_poly_cxt_t::gen
```

General parameters.

height

```
int pvr_poly_cxt_t::height
```

Texture height (requires a power of 2)

list_type

```
int pvr_poly_cxt_t::list_type
```

Primitive list.

See also

PVR primitive list types

mipmap

```
int pvr_poly_cxt_t::mipmap
```

Enable/disable mipmaps.

See also

Enable or disable PVR mipmap processing

mipmap_bias

```
int pvr_poly_cxt_t::mipmap_bias
```

Mipmap bias.

See also

PVR mipmap bias modes

modifier

```
int pvr_poly_cxt_t::modifier
```

Enable or disable modifier effect.

See also

Enable or disable modifier effects

modifier_mode

```
int pvr_poly_cxt_t::modifier_mode
```

Modifier mode.

shading

```
int pvr_poly_cxt_t::shading
```

Shading type.

See also

PVR shading modes

specular

```
int pvr_poly_cxt_t::specular
```

Offset color enable/disable outside modifier.

See also

Enable or disable offset color

src

```
int pvr_poly_cxt_t::src
```

Source blending mode outside modifier.

See also

PVR blending modes

src2

```
int pvr_poly_cxt_t::src2
```

Source blending mode inside modifier.

See also

PVR blending modes

src_enable

```
int pvr_poly_cxt_t::src_enable
```

Source blending enable outside modifier.

See also

Enable or disable blending

src_enable2

```
int pvr_poly_cxt_t::src_enable2
```

Source blending mode inside modifier.

See also

Enable or disable blending

[struct]

```
struct { ... } pvr_poly_cxt_t::txr
```

Texturing params outside modifier.

[struct]

```
struct { ... } pvr_poly_cxt_t::txr2
```

Texturing params inside modifier.

uv

`int pvr_poly_cxt_t::uv`

U/V data format in vertex.

See also

PVR U/V data format control

uv_clamp

`int pvr_poly_cxt_t::uv_clamp`

Enable/disable U/V clamping.

See also

Enable or disable clamping of U/V on the PVR

uv_flip

`int pvr_poly_cxt_t::uv_flip`

Enable/disable U/V flipping.

See also

Enable or disable U/V flipping on the PVR

width

`int pvr_poly_cxt_t::width`

Texture width (requires a power of 2)

write

`int pvr_poly_cxt_t::write`

Enable or disable depth writes.

See also

Enable or disable PVR depth writes

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.76 pvr_poly_hdr_t Struct Reference

PVR polygon header.

`#include <dc/pvr.h>`

Data Fields

- uint32_t cmd
  TA command.
- uint32_t mode1
  Parameter word 1.
- uint32_t mode2
  Parameter word 2.
- uint32_t mode3
  Parameter word 3.
- uint32_t d1
  Dummy value.
- uint32_t d2
  Dummy value.
- uint32_t d3
  Dummy value.
- uint32_t d4
  Dummy value.

8.76.1 Detailed Description

PVR polygon header.

This is the hardware equivalent of a rendering context; you'll create one of these from your pvr_poly_cxt_t and use it for submission to the hardware.

8.76.2 Field Documentation

cmd

`uint32_t pvr_poly_hdr_t::cmd`

TA command.

d1

`uint32_t pvr_poly_hdr_t::d1`

Dummy value.

d2

`uint32_t pvr_poly_hdr_t::d2`

Dummy value.

d3

`uint32_t pvr_poly_hdr_t::d3`

Dummy value.

d4

`uint32_t pvr_poly_hdr_t::d4`

Dummy value.

mode1

`uint32_t pvr_poly_hdr_t::mode1`

Parameter word 1.

mode2

`uint32_t pvr_poly_hdr_t::mode2`

Parameter word 2.

mode3

`uint32_t pvr_poly_hdr_t::mode3`

Parameter word 3.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.77 pvr_poly_ic_hdr_t Struct Reference

PVR polygon header with intensity color.

`#include <dc/pvr.h>`

Data Fields

- uint32_t cmd
  TA command.
- uint32_t mode1
  Parameter word 1.
- uint32_t mode2
  Parameter word 2.
- uint32_t mode3
  Parameter word 3.
- float a
  Face color alpha component.
- float r
  Face color red component.
- float g
  Face color green component.
- float b
  Face color blue component.

8.77.1 Detailed Description

PVR polygon header with intensity color.

This is the equivalent of pvr_poly_hdr_t, but for use with intensity color.

8.77.2 Field Documentation

a

`float pvr_poly_ic_hdr_t::a`

Face color alpha component.

b

`float pvr_poly_ic_hdr_t::b`

Face color blue component.

cmd

```
uint32_t pvr_poly_ic_hdr_t::cmd
```

TA command.

g

```
float pvr_poly_ic_hdr_t::g
```

Face color green component.

mode1

```
uint32_t pvr_poly_ic_hdr_t::mode1
```

Parameter word 1.

mode2

```
uint32_t pvr_poly_ic_hdr_t::mode2
```

Parameter word 2.

mode3

```
uint32_t pvr_poly_ic_hdr_t::mode3
```

Parameter word 3.

r

```
float pvr_poly_ic_hdr_t::r
```

Face color red component.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.78 pvr_poly_mod_hdr_t Struct Reference

PVR polygon header to be used with modifier volumes.

`#include <dc/pvr.h>`

Data Fields

- uint32_t cmd
  TA command.
- uint32_t mode1
  Parameter word 1.
- uint32_t mode2_0
  Parameter word 2 (outside volume)
- uint32_t mode3_0
  Parameter word 3 (outside volume)
- uint32_t mode2_1
  Parameter word 2 (inside volume)
- uint32_t mode3_1
  Parameter word 3 (inside volume)
- uint32_t d1
  Dummy value.
- uint32_t d2
  Dummy value.

8.78.1 Detailed Description

PVR polygon header to be used with modifier volumes.

This is the equivalent of a pvr_poly_hdr_t for use when a polygon is to be used with modifier volumes.

8.78.2 Field Documentation

cmd

`uint32_t pvr_poly_mod_hdr_t::cmd`

TA command.

d1

`uint32_t pvr_poly_mod_hdr_t::d1`

Dummy value.

d2

`uint32_t pvr_poly_mod_hdr_t::d2`

Dummy value.

mode1

`uint32_t pvr_poly_mod_hdr_t::mode1`

Parameter word 1.

mode2_0

`uint32_t pvr_poly_mod_hdr_t::mode2_0`

Parameter word 2 (outside volume)

mode2_1

`uint32_t pvr_poly_mod_hdr_t::mode2_1`

Parameter word 2 (inside volume)

mode3_0

`uint32_t pvr_poly_mod_hdr_t::mode3_0`

Parameter word 3 (outside volume)

mode3_1

`uint32_t pvr_poly_mod_hdr_t::mode3_1`

Parameter word 3 (inside volume)

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.79 pvr_sprite_col_t Struct Reference

PVR vertex type: Untextured sprite.

`#include <pvr.h>`

Data Fields

- uint32_t flags
  TA command (vertex flags)
- float ax
  First X coordinate.
- float ay
  First Y coordinate.
- float az
  First Z coordinate.
- float bx
  Second X coordinate.
- float by
  Second Y coordinate.
- float bz
  Second Z coordinate.
- float cx
  Third X coordinate.
- float cy
  Third Y coordinate.
- float cz
  Third Z coordinate.
- float dx
  Fourth X coordinate.
- float dy
  Fourth Y coordinate.
- uint32_t d1
  Dummy value.
- uint32_t d2
  Dummy value.
- uint32_t d3
  Dummy value.
- uint32_t d4
  Dummy value.

8.79.1 Detailed Description

PVR vertex type: Untextured sprite.

This vertex type is to be used with the sprite polygon header and the sprite related commands to draw untextured sprites (aka, quads).

8.79.2 Field Documentation

ax

```
float pvr_sprite_col_t::ax
```

First X coordinate.

ay

```
float pvr_sprite_col_t::ay
```

First Y coordinate.

az

```
float pvr_sprite_col_t::az
```

First Z coordinate.

bx

```
float pvr_sprite_col_t::bx
```

Second X coordinate.

by

```
float pvr_sprite_col_t::by
```

Second Y coordinate.

bz

```
float pvr_sprite_col_t::bz
```

Second Z coordinate.

cx

```
float pvr_sprite_col_t::cx
```

Third X coordinate.

cy

```
float pvr_sprite_col_t::cy
```

Third Y coordinate.

cz

```
float pvr_sprite_col_t::cz
```

Third Z coordinate.

d1

```
uint32_t pvr_sprite_col_t::d1
```

Dummy value.

d2

```
uint32_t pvr_sprite_col_t::d2
```

Dummy value.

d3

```
uint32_t pvr_sprite_col_t::d3
```

Dummy value.

d4

```
uint32_t pvr_sprite_col_t::d4
```

Dummy value.

dx

```
float pvr_sprite_col_t::dx
```

Fourth X coordinate.

dy

```
float pvr_sprite_col_t::dy
```

Fourth Y coordinate.

flags

```
uint32_t pvr_sprite_col_t::flags
```

TA command (vertex flags)

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.80 pvr_sprite_cxt_t Struct Reference

PVR sprite context.

```
#include <dc/pvr.h>
```

Data Fields

- int list_type
  Primitive list.
- struct {
  int alpha
  Enable or disable alpha.
  int fog_type
  Fog type.
  int culling
  Culling mode.
  int color_clamp
  Color clamp enable/disable.
  int clip_mode
  Clipping mode.
  int specular
  Offset color enable/disable.
  } gen

  General parameters.

- struct {
  int src
  Source blending mode.
  int dst
  Dest blending mode.
  int src_enable
  Source blending enable.
  int dst_enable
  Dest blending enable.
  } blend

- struct {
  int comparison
  Depth comparison mode.
  int write
  Enable or disable depth writes.
  } depth

  Depth comparison/write modes.

- struct {
  int enable
  Enable/disable texturing.
  int filter
  Filtering mode.
  int mipmap
  Enable/disable mipmaps.
  int mipmap_bias
  Mipmap bias.
  int uv_flip
  Enable/disable U/V flipping.
  int uv_clamp
  Enable/disable U/V clamping.
  int alpha
  Enable/disable texture alpha.
  int env
  Texture color contribution.
  int width
  Texture width (requires a power of 2)
  int height
  Texture height (requires a power of 2)
  int format
  Texture format.
  pvr_ptr_t base
  Texture pointer.
  } txr

  Texturing params.

8.80.1 Detailed Description

PVR sprite context.

You should use this more human readable format for specifying your sprite contexts, and then compile them into sprite headers when you are ready to start using them.

Unfortunately, it seems that Doxygen chokes up a little bit on this structure, and others like it. The documentation should still be mostly understandable though...

8.80.2 Field Documentation

alpha

```
int pvr_sprite_cxt_t::alpha
```

Enable or disable alpha.

Enable/disable texture alpha.

See also

Enable or disable alpha blending

Enable or disable texture alpha blending

base

```
pvr_ptr_t pvr_sprite_cxt_t::base
```

Texture pointer.

[struct]

```
struct { ... } pvr_sprite_cxt_t::blend
```

clip_mode

```
int pvr_sprite_cxt_t::clip_mode
```

Clipping mode.

See also

PVR clipping modes

color_clamp

```
int pvr_sprite_cxt_t::color_clamp
```

Color clamp enable/disable.

See also

Enable or disable color clamping

comparison

```
int pvr_sprite_cxt_t::comparison
```

Depth comparison mode.

See also

PVR depth comparison modes

culling

```
int pvr_sprite_cxt_t::culling
```

Culling mode.

See also

PVR culling modes

[struct]

```
struct { ... } pvr_sprite_cxt_t::depth
```

Depth comparison/write modes.

dst

```
int pvr_sprite_cxt_t::dst
```

Dest blending mode.

See also

PVR blending modes

dst_enable

```
int pvr_sprite_cxt_t::dst_enable
```

Dest blending enable.

See also

Enable or disable blending

enable

```
int pvr_sprite_cxt_t::enable
```

Enable/disable texturing.

See also

Enable or disable texturing on polygons

env

```
int pvr_sprite_cxt_t::env
```

Texture color contribution.

See also

Texture color calculation modes

filter

```
int pvr_sprite_cxt_t::filter
```

Filtering mode.

See also

PVR texture sampling modes

fog_type

```
int pvr_sprite_cxt_t::fog_type
```

Fog type.

See also

PVR fog modes

format

```
int pvr_sprite_cxt_t::format
```

Texture format.

See also

PVR texture formats

[struct]

```
struct { ... } pvr_sprite_cxt_t::gen
```

General parameters.

height

```
int pvr_sprite_cxt_t::height
```

Texture height (requires a power of 2)

list_type

```
int pvr_sprite_cxt_t::list_type
```

Primitive list.

See also

PVR primitive list types

mipmap

```
int pvr_sprite_cxt_t::mipmap
```

Enable/disable mipmaps.

See also

Enable or disable PVR mipmap processing

mipmap_bias

```
int pvr_sprite_cxt_t::mipmap_bias
```

Mipmap bias.

See also

PVR mipmap bias modes

specular

```
int pvr_sprite_cxt_t::specular
```

Offset color enable/disable.

See also

Enable or disable offset color

src

```
int pvr_sprite_cxt_t::src
```

Source blending mode.

See also

PVR blending modes

src_enable

```
int pvr_sprite_cxt_t::src_enable
```

Source blending enable.

See also

Enable or disable blending

[struct]

```
struct { ... } pvr_sprite_cxt_t::txr
```

Texturing params.

uv_clamp

```
int pvr_sprite_cxt_t::uv_clamp
```

Enable/disable U/V clamping.

See also

Enable or disable clamping of U/V on the PVR

uv_flip

```
int pvr_sprite_cxt_t::uv_flip
```

Enable/disable U/V flipping.

See also

Enable or disable U/V flipping on the PVR

width

```
int pvr_sprite_cxt_t::width
```

Texture width (requires a power of 2)

write

```
int pvr_sprite_cxt_t::write
```

Enable or disable depth writes.

See also

Enable or disable PVR depth writes

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.81 pvr_sprite_hdr_t Struct Reference

PVR polygon header specifically for sprites.

```
#include <dc/pvr.h>
```

Data Fields

- uint32_t cmd
  TA command.
- uint32_t mode1
  Parameter word 1.
- uint32_t mode2
  Parameter word 2.
- uint32_t mode3
  Parameter word 3.
- uint32_t argb
  Sprite face color.
- uint32_t oargb
  Sprite offset color.
- uint32_t d1
  Dummy value.
- uint32_t d2
  Dummy value.

8.81.1 Detailed Description

PVR polygon header specifically for sprites.

This is the equivalent of a pvr_poly_hdr_t for use when a quad/sprite is to be rendered. Note that the color data is here, not in the vertices.

8.81.2 Field Documentation

argb

```
uint32_t pvr_sprite_hdr_t::argb
```

Sprite face color.

cmd

```
uint32_t pvr_sprite_hdr_t::cmd
```

TA command.

d1

```
uint32_t pvr_sprite_hdr_t::d1
```

Dummy value.

d2

```
uint32_t pvr_sprite_hdr_t::d2
```

Dummy value.

mode1

```
uint32_t pvr_sprite_hdr_t::mode1
```

Parameter word 1.

mode2

```
uint32_t pvr_sprite_hdr_t::mode2
```

Parameter word 2.

mode3

```
uint32_t pvr_sprite_hdr_t::mode3
```

Parameter word 3.

oargb

`uint32_t pvr_sprite_hdr_t::oargb`

Sprite offset color.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.82 pvr_sprite_txr_t Struct Reference

PVR vertex type: Textured sprite.

`#include <dc/pvr.h>`

Data Fields

- uint32_t flags
  TA command (vertex flags)
- float ax
  First X coordinate.
- float ay
  First Y coordinate.
- float az
  First Z coordinate.
- float bx
  Second X coordinate.
- float by
  Second Y coordinate.
- float bz
  Second Z coordinate.
- float cx
  Third X coordinate.
- float cy
  Third Y coordinate.
- float cz
  Third Z coordinate.
- float dx
  Fourth X coordinate.
- float dy
  Fourth Y coordinate.
- uint32_t dummy
  Dummy value.
- uint32_t auv
  First U/V texture coordinates.
- uint32_t buv
  Second U/V texture coordinates.
- uint32_t cuv
  Third U/V texture coordinates.

8.82.1 Detailed Description

PVR vertex type: Textured sprite.

This vertex type is to be used with the sprite polygon header and the sprite related commands to draw textured sprites. Note that there is no fourth Z coordinate. I suppose it just gets interpolated?

The U/V coordinates in here are in the 16-bit per coordinate form. Also, like the fourth Z value, there is no fourth U or V, so it must get interpolated from the others.

8.82.2 Field Documentation

auv

```
uint32_t pvr_sprite_txr_t::auv
```

First U/V texture coordinates.

ax

```
float pvr_sprite_txr_t::ax
```

First X coordinate.

ay

```
float pvr_sprite_txr_t::ay
```

First Y coordinate.

az

```
float pvr_sprite_txr_t::az
```

First Z coordinate.

buv

```
uint32_t pvr_sprite_txr_t::buv
```

Second U/V texture coordinates.

bx

```
float pvr_sprite_txr_t::bx
```

Second X coordinate.

by

```
float pvr_sprite_txr_t::by
```

Second Y coordinate.

bz

```
float pvr_sprite_txr_t::bz
```

Second Z coordinate.

cuv

```
uint32_t pvr_sprite_txr_t::cuv
```

Third U/V texture coordinates.

cx

```
float pvr_sprite_txr_t::cx
```

Third X coordinate.

cy

```
float pvr_sprite_txr_t::cy
```

Third Y coordinate.

cz

```
float pvr_sprite_txr_t::cz
```

Third Z coordinate.

dummy

```
uint32_t pvr_sprite_txr_t::dummy
```

Dummy value.

dx

```
float pvr_sprite_txr_t::dx
```

Fourth X coordinate.

dy

```
float pvr_sprite_txr_t::dy
```

Fourth Y coordinate.

flags

```
uint32_t pvr_sprite_txr_t::flags
```

TA command (vertex flags)

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.83 pvr_stats_t Struct Reference

PVR statistics structure.

```
#include <dc/pvr.h>
```

Data Fields

- uint32_t enabled_list_mask

  Which lists are enabled?
- uint32_t vbl_count

  VBlank count.
- int frame_last_time

  Ready-to-Ready length for the last frame in milliseconds.
- float frame_rate

  Current frame rate (per second)
- int reg_last_time

  Registration time for the last frame in milliseconds.
- int rnd_last_time

  Rendering time for the last frame in milliseconds.
- int vtx_buffer_used

  Number of bytes used in the vertex buffer for the last frame.
- int vtx_buffer_used_max

  Number of bytes used in the vertex buffer for the largest frame.
- int buf_last_time

  DMA buffer file time for the last frame in milliseconds.
- uint32_t frame_count

  Total number of rendered/viewed frames.

8.83.1 Detailed Description

PVR statistics structure.

This structure is used to hold various statistics about the operation of the PVR since initialization.

8.83.2 Field Documentation

buf_last_time

```
int pvr_stats_t::buf_last_time
```

DMA buffer file time for the last frame in milliseconds.

enabled_list_mask

```
uint32_t pvr_stats_t::enabled_list_mask
```

Which lists are enabled?

frame_count

```
uint32_t pvr_stats_t::frame_count
```

Total number of rendered/viewed frames.

frame_last_time

```
int pvr_stats_t::frame_last_time
```

Ready-to-Ready length for the last frame in milliseconds.

frame_rate

```
float pvr_stats_t::frame_rate
```

Current frame rate (per second)

reg_last_time

```
int pvr_stats_t::reg_last_time
```

Registration time for the last frame in milliseconds.

rnd_last_time

```
int pvr_stats_t::rnd_last_time
```

Rendering time for the last frame in milliseconds.

vbl_count

```
uint32_t pvr_stats_t::vbl_count
```

VBlank count.

vtx_buffer_used

```
int pvr_stats_t::vtx_buffer_used
```

Number of bytes used in the vertex buffer for the last frame.

vtx_buffer_used_max

```
int pvr_stats_t::vtx_buffer_used_max
```

Number of bytes used in the vertex buffer for the largest frame.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.84 pvr_vertex_pcm_t Struct Reference

PVR vertex type: Non-textured, packed color, affected by modifier volume.

```
#include <dc/pvr.h>
```

Data Fields

- uint32_t flags
  TA command (vertex flags)
- float x
  X coordinate.
- float y
  Y coordinate.
- float z
  Z coordinate.
- uint32_t argb0
  Vertex color (outside volume)
- uint32_t argb1
  Vertex color (inside volume)
- uint32_t d1
  Dummy value.
- uint32_t d2
  Dummy value.

8.84.1 Detailed Description

PVR vertex type: Non-textured, packed color, affected by modifier volume.

This vertex type has two copies of colors. The second color is used when enclosed within a modifier volume.

8.84.2 Field Documentation

argb0

```
uint32_t pvr_vertex_pcm_t::argb0
```

Vertex color (outside volume)

argb1

```
uint32_t pvr_vertex_pcm_t::argb1
```

Vertex color (inside volume)

d1

```
uint32_t pvr_vertex_pcm_t::d1
```

Dummy value.

d2

```
uint32_t pvr_vertex_pcm_t::d2
```

Dummy value.

flags

```
uint32_t pvr_vertex_pcm_t::flags
```

TA command (vertex flags)

x

```
float pvr_vertex_pcm_t::x
```

X coordinate.

y

```
float pvr_vertex_pcm_t::y
```

Y coordinate.

z

`float pvr_vertex_pcm_t::z`

Z coordinate.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.85 pvr_vertex_t Struct Reference

Generic PVR vertex type.

`#include <dc/pvr.h>`

Data Fields

- uint32_t flags
  TA command (vertex flags)
- float x
  X coordinate.
- float y
  Y coordinate.
- float z
  Z coordinate.
- float u
  Texture U coordinate.
- float v
  Texture V coordinate.
- uint32_t argb
  Vertex color.
- uint32_t oargb
  Vertex offset color.

8.85.1 Detailed Description

Generic PVR vertex type.

The PVR chip itself supports many more vertex types, but this is the main one that can be used with both textured and non-textured polygons, and is fairly fast.

8.85.2 Field Documentation

argb

```
uint32_t pvr_vertex_t::argb
```

Vertex color.

flags

```
uint32_t pvr_vertex_t::flags
```

TA command (vertex flags)

oargb

```
uint32_t pvr_vertex_t::oargb
```

Vertex offset color.

u

```
float pvr_vertex_t::u
```

Texture U coordinate.

v

```
float pvr_vertex_t::v
```

Texture V coordinate.

x

```
float pvr_vertex_t::x
```

X coordinate.

y

```
float pvr_vertex_t::y
```

Y coordinate.

z

```
float pvr_vertex_t::z
```

Z coordinate.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.86 pvr_vertex_tpcm_t Struct Reference

PVR vertex type: Textured, packed color, affected by modifer volume.

```
#include <dc/pvr.h>
```

Data Fields

- uint32_t flags
  TA command (vertex flags)
- float x
  X coordinate.
- float y
  Y coordinate.
- float z
  Z coordinate.
- float u0
  Texture U coordinate (outside)
- float v0
  Texture V coordinate (outside)
- uint32_t argb0
  Vertex color (outside)
- uint32_t oargb0
  Vertex offset color (outside)
- float u1
  Texture U coordinate (inside)
- float v1
  Texture V coordinate (inside)
- uint32_t argb1
  Vertex color (inside)
- uint32_t oargb1
  Vertex offset color (inside)
- uint32_t d1
  Dummy value.
- uint32_t d2
  Dummy value.
- uint32_t d3
  Dummy value.
- uint32_t d4
  Dummy value.

8.86.1 Detailed Description

PVR vertex type: Textured, packed color, affected by modifer volume.

Note that this vertex type has two copies of colors, offset colors, and texture coords. The second set of texture coords, colors, and offset colors are used when enclosed within a modifer volume.

8.86.2 Field Documentation

argb0

```
uint32_t pvr_vertex_tpcm_t::argb0
```

Vertex color (outside)

argb1

```
uint32_t pvr_vertex_tpcm_t::argb1
```

Vertex color (inside)

d1

```
uint32_t pvr_vertex_tpcm_t::d1
```

Dummy value.

d2

```
uint32_t pvr_vertex_tpcm_t::d2
```

Dummy value.

d3

```
uint32_t pvr_vertex_tpcm_t::d3
```

Dummy value.

d4

```
uint32_t pvr_vertex_tpcm_t::d4
```

Dummy value.

flags

```
uint32_t pvr_vertex_tpcm_t::flags
```

TA command (vertex flags)

oargb0

```
uint32_t pvr_vertex_tpcm_t::oargb0
```

Vertex offset color (outside)

oargb1

```
uint32_t pvr_vertex_tpcm_t::oargb1
```

Vertex offset color (inside)

u0

```
float pvr_vertex_tpcm_t::u0
```

Texture U coordinate (outside)

u1

```
float pvr_vertex_tpcm_t::u1
```

Texture U coordinate (inside)

v0

```
float pvr_vertex_tpcm_t::v0
```

Texture V coordinate (outside)

v1

```
float pvr_vertex_tpcm_t::v1
```

Texture V coordinate (inside)

x

```
float pvr_vertex_tpcm_t::x
```

X coordinate.

y

```
float pvr_vertex_tpcm_t::y
```

Y coordinate.

z

```
float pvr_vertex_tpcm_t::z
```

Z coordinate.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/pvr.h

8.87 rw_semaphore_t Struct Reference

Reader/writer semaphore structure.

```
#include <kos/rwsem.h>
```

Data Fields

- int dynamic
  Was this structure created with rwsem_create()?
- int read_count
  The number of readers that are currently holding the lock.
- kthread_t * write_lock
  The thread holding the write lock.
- kthread_t * reader_waiting
  Space for one reader who's trying to upgrade to a writer.

8.87.1 Detailed Description

Reader/writer semaphore structure.

All members of this structure should be considered to be private, it is not safe to change anything in here yourself.

8.87.2 Field Documentation

dynamic

```
int rw_semaphore_t::dynamic
```

Was this structure created with rwsem_create()?

read_count

```
int rw_semaphore_t::read_count
```

The number of readers that are currently holding the lock.

reader_waiting

```
kthread_t* rw_semaphore_t::reader_waiting
```

Space for one reader who's trying to upgrade to a writer.

write_lock

```
kthread_t* rw_semaphore_t::write_lock
```

The thread holding the write lock.

The documentation for this struct was generated from the following file:

- include/kos/rwsem.h

8.88 sched_param Struct Reference

Scheduling Parameters, P1003.1b-1993, p. 249.

```
#include <sys/sched.h>
```

Data Fields

- int sched_priority

  Process execution scheduling priority.

8.88.1 Detailed Description

Scheduling Parameters, P1003.1b-1993, p. 249.

Note

Fields whose name begins with "ss_" added by P1003.4b/D8, p. 33.

8.88.2 Field Documentation

sched_priority

```
int sched_param::sched_priority
```

Process execution scheduling priority.

The documentation for this struct was generated from the following file:

- include/sys/sched.h

8.89 semaphore_t Struct Reference

Semaphore type.

```
#include <kos/sem.h>
```

Data Fields

- int initialized

  Are we initialized?
- int count

  The semaphore count.

8.89.1 Detailed Description

Semaphore type.

This structure defines a semaphore. There are no public members of this structure for you to actually do anything with in your code, so don't try.

8.89.2 Field Documentation

count

```
int semaphore_t::count
```

The semaphore count.

initialized

```
int semaphore_t::initialized
```

Are we initialized?

The documentation for this struct was generated from the following file:

- include/kos/sem.h

8.90 sip_state_t Struct Reference

SIP status structure.

```
#include <dc/maple/sip.h>
```

Data Fields

- int amp_gain
  The gain value for the microphone amp.
- int sample_type
  The type of samples that are being recorded.
- int frequency
  What frequency are we sampling at?
- int is_sampling
  Is the mic currently sampling?
- sip_sample_cb callback
  Sampling callback.

8.90.1 Detailed Description

SIP status structure.

This structure contains information about the status of the microphone device and can be fetched with maple_dev_status(). You should not modify any of the values in here, it is all "read-only" to your programs. Modifying any of this, especially while the microphone is sampling could really screw things up.

8.90.2 Field Documentation

amp_gain

```
int sip_state_t::amp_gain
```

The gain value for the microphone amp.

callback

```
sip_sample_cb sip_state_t::callback
```

Sampling callback.

frequency

```
int sip_state_t::frequency
```

What frequency are we sampling at?

is_sampling

```
int sip_state_t::is_sampling
```

Is the mic currently sampling?

sample_type

```
int sip_state_t::sample_type
```

The type of samples that are being recorded.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple/sip.h

8.91 sockaddr Struct Reference

Socket address structure.

```
#include <sys/socket.h>
```

Data Fields

- sa_family_t sa_family
  Address family.
- char sa_data [ ]
  Address data.

8.91.1 Detailed Description

Socket address structure.

8.91.2 Field Documentation

sa_data

```
char sockaddr::sa_data[]
```

Address data.

sa_family

```
sa_family_t sockaddr::sa_family
```

Address family.

The documentation for this struct was generated from the following file:

- include/sys/socket.h

8.92 sockaddr_in Struct Reference

Structure used to store an IPv4 address for a socket.

```
#include <netinet/in.h>
```

Data Fields

- sa_family_t sin_family
  Family for the socket. Must be AF_INET.
- in_port_t sin_port
  Port for the socket. Must be in network byte order.
- struct in_addr sin_addr
  Address for the socket. Must be in network byte order.
- unsigned char sin_zero [8]
  Empty space, ignored for all intents and purposes.

8.92.1 Detailed Description

Structure used to store an IPv4 address for a socket.

This structure is the standard way to set up addresses for sockets in the AF_INET address family. Generally you will not send one of these directly to a function, but rather will cast it to a struct sockaddr. Also, this structure contains the old sin_zero member which is no longer required by the standard (for compatibility with applications that expect it).

8.92.2 Field Documentation

sin_addr

```
struct in_addr sockaddr_in::sin_addr
```

Address for the socket. Must be in network byte order.

sin_family

```
sa_family_t sockaddr_in::sin_family
```

Family for the socket. Must be AF_INET.

sin_port

```
in_port_t sockaddr_in::sin_port
```

Port for the socket. Must be in network byte order.

sin_zero

```
unsigned char sockaddr_in::sin_zero[8]
```

Empty space, ignored for all intents and purposes.

The documentation for this struct was generated from the following file:

- include/netinet/in.h

8.93 sockaddr_in6 Struct Reference

Structure used to store an IPv6 address for a socket.

```
#include <netinet/in.h>
```

Data Fields

- sa_family_t sin6_family

  Family for the socket. Must be AF_INET6.
- in_port_t sin6_port

  Port for the socket. Must be in network byte order.
- uint32_t sin6_flowinfo

  Traffic class and flow information.
- struct in6_addr sin6_addr

  Address for the socket. Must be in network byte order.
- uint32_t sin6_scope_id

  Set of interfaces for a scope.

8.93.1 Detailed Description

Structure used to store an IPv6 address for a socket.

This structure is the standard way to set up addresses for sockets in the AF_INET6 address family. Generally you will not send one of these directly to a function, but rather will cast it to a struct sockaddr.

8.93.2 Field Documentation

sin6_addr

```
struct in6_addr sockaddr_in6::sin6_addr
```

Address for the socket. Must be in network byte order.

sin6_family

```
sa_family_t sockaddr_in6::sin6_family
```

Family for the socket. Must be AF_INET6.

sin6_flowinfo

```
uint32_t sockaddr_in6::sin6_flowinfo
```

Traffic class and flow information.

sin6_port

```
in_port_t sockaddr_in6::sin6_port
```

Port for the socket. Must be in network byte order.

sin6_scope_id

```
uint32_t sockaddr_in6::sin6_scope_id
```

Set of interfaces for a scope.

The documentation for this struct was generated from the following file:

- include/netinet/in.h

8.94 sockaddr_storage Struct Reference

Socket address structure of appropriate size to hold any supported socket type's addresses.

```
#include <sys/socket.h>
```

Data Fields

- sa_family_t ss_family
  Address family.
- char _ss_pad1 [_SS_PAD1SIZE]
  First padding field.
- __uint64_t _ss_align
  Used to force alignment.
- char _ss_pad2 [_SS_PAD2SIZE]
  Second padding field to fill up the space required.

8.94.1 Detailed Description

Socket address structure of appropriate size to hold any supported socket type's addresses.

8.94.2 Field Documentation

_ss_align

```
__uint64_t sockaddr_storage::_ss_align
```

Used to force alignment.

_ss_pad1

```
char sockaddr_storage::_ss_pad1[_SS_PAD1SIZE]
```

First padding field.

_ss_pad2

```
char sockaddr_storage::_ss_pad2[_SS_PAD2SIZE]
```

Second padding field to fill up the space required.

ss_family

```
sa_family_t sockaddr_storage::ss_family
```

Address family.

The documentation for this struct was generated from the following file:

- include/sys/socket.h

8.95 symtab_handler_t Struct Reference

A symbol table "handler" for nmmgr.

```
#include <kos/exports.h>
```

Data Fields

- struct nmmgr_handler nmmgr
  Name manager handler header.
- export_sym_t * table
  Location of the first entry.

8.95.1 Detailed Description

A symbol table "handler" for nmmgr.

8.95.2 Field Documentation

nmmgr

```
struct nmmgr_handler symtab_handler_t::nmmgr
```

Name manager handler header.

table

```
export_sym_t* symtab_handler_t::table
```

Location of the first entry.

The documentation for this struct was generated from the following file:

- include/kos/exports.h

8.96 tcbhead_t Struct Reference

Control Block Header.

```
#include <thread.h>
```

Data Fields

- void * dtv
  Dynamic TLS vector (unused)
- uintptr_t pointer_guard
  Pointer guard (unused)

8.96.1 Detailed Description

Control Block Header.

Header preceeding the static TLS data segments as defined by the SH-ELF TLS ABI (version 1). This is what the thread pointer (GBR) points to for compiler access to thread-local data.

8.96.2 Field Documentation

dtv

```
void* tcbhead_t::dtv
```

Dynamic TLS vector (unused)

pointer_guard

```
uintptr_t tcbhead_t::pointer_guard
```

Pointer guard (unused)

The documentation for this struct was generated from the following file:

- include/kos/thread.h

8.97 utsname Struct Reference

Kernel name/information structure.

`#include <sys/utsname.h>`

Data Fields

- char sysname [_UTSNAME_LENGTH]
  OS Name ("KallistiOS").
- char nodename [_UTSNAME_LENGTH]
  Name on network, if any.
- char release [_UTSNAME_LENGTH]
  Kernel release ("2.1.0").
- char version [_UTSNAME_LENGTH]
  Kernel version string.
- char machine [_UTSNAME_LENGTH]
  Hardware identifier.

8.97.1 Detailed Description

Kernel name/information structure.

This structure contains information about the kernel and is used by the uname() function for returning that information to a program.

8.97.2 Field Documentation

machine

`char utsname::machine[_UTSNAME_LENGTH]`

Hardware identifier.

nodename

`char utsname::nodename[_UTSNAME_LENGTH]`

Name on network, if any.

release

`char utsname::release[_UTSNAME_LENGTH]`

Kernel release ("2.1.0").

sysname

```
char utsname::sysname[_UTSNAME_LENGTH]
```

OS Name ("KallistiOS").

version

```
char utsname::version[_UTSNAME_LENGTH]
```

Kernel version string.

The documentation for this struct was generated from the following file:

- include/sys/utsname.h

8.98 vec3f_t Struct Reference

```
#include <vec3f.h>
```

Data Fields

- float x
- float y
- float z

8.98.1 Field Documentation

x

```
float vec3f_t::x
```

y

```
float vec3f_t::y
```

z

```
float vec3f_t::z
```

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/vec3f.h

8.99 vector_t Struct Reference

4-part vector type.

`#include <dc/vector.h>`

Data Fields

- float x
- float y
- float z
- float w

8.99.1 Detailed Description

4-part vector type.

8.99.2 Field Documentation

w

`float vector_t::w`

x

`float vector_t::x`

y

`float vector_t::y`

z

`float vector_t::z`

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/vector.h

8.100 vfs_handler_t Struct Reference

VFS handler interface.

`#include <kos/fs.h>`

Data Fields

- nmmgr_handler_t nmmgr

  Name manager handler header.
- int cache

  Allow VFS cacheing; 0=no, 1=yes.
- void * privdata

  Pointer to private data for the handler.
- void *(* open)(struct vfs_handler *vfs, const char *fn, int mode)

  Open a file on the given VFS; return a unique identifier.
- int(* close)(void *hnd)

  Close a previously opened file.
- ssize_t(* read)(void *hnd, void *buffer, size_t cnt)

  Read from a previously opened file.
- ssize_t(* write)(void *hnd, const void *buffer, size_t cnt)

  Write to a previously opened file.
- off_t(* seek)(void *hnd, off_t offset, int whence)

  Seek in a previously opened file.
- off_t(* tell)(void *hnd)

  Return the current position in a previously opened file.
- size_t(* total)(void *hnd)

  Return the total size of a previously opened file.
- dirent_t *(* readdir)(void *hnd)

  Read the next directory entry in a directory opened with O_DIR.
- int(* ioctl)(void *hnd, int cmd, va_list ap)

  Execute a device-specific call on a previously opened file.
- int(* rename)(struct vfs_handler *vfs, const char *fn1, const char *fn2)

  Rename/move a file on the given VFS.
- int(* unlink)(struct vfs_handler *vfs, const char *fn)

  Delete a file from the given VFS.
- void *(* mmap)(void *fd)

  "Memory map" a previously opened file
- int(* complete)(void *fd, ssize_t *rv)

  Perform an I/O completion (async I/O) for a previously opened file.
- int(* stat)(struct vfs_handler *vfs, const char *path, struct stat *buf, int flag)

  Get status information on a file on the given VFS.
- int(* mkdir)(struct vfs_handler *vfs, const char *fn)

  Make a directory on the given VFS.
- int(* rmdir)(struct vfs_handler *vfs, const char *fn)

  Remove a directory from the given VFS.
- int(* fcntl)(void *fd, int cmd, va_list ap)

Manipulate file control flags on the given file.

- short(∗ poll)(void ∗fd, short events)

  Check if an event is pending on the given file.
- int(∗ link)(struct vfs_handler ∗vfs, const char ∗path1, const char ∗path2)

  Create a hard link.
- int(∗ symlink)(struct vfs_handler ∗vfs, const char ∗path1, const char ∗path2)

  Create a symbolic link.
- _off64_t(∗ seek64)(void ∗hnd, _off64_t offset, int whence)

  Seek in a previously opened file (64-bit offsets)
- _off64_t(∗ tell64)(void ∗hnd)

  Return the current position in an opened file (64-bit offset)
- uint64(∗ total64)(void ∗hnd)

  Return the size of an opened file as a 64-bit integer.
- ssize_t(∗ readlink)(struct vfs_handler ∗vfs, const char ∗path, char ∗buf, size_t bufsize)

  Read the value of a symbolic link.
- int(∗ rewinddir)(void ∗hnd)

  Rewind a directory stream to the start.
- int(∗ fstat)(void ∗hnd, struct stat ∗st)

  Get status information on an already opened file.

8.100.1 Detailed Description

VFS handler interface.

All VFS handlers must implement this interface.

8.100.2 Field Documentation

cache

```
int vfs_handler_t::cache
```

Allow VFS cacheing; 0=no, 1=yes.

close

```
int(* vfs_handler_t::close) (void *hnd)
```

Close a previously opened file.

complete

```
int(* vfs_handler_t::complete) (void *fd, ssize_t *rv)
```

Perform an I/O completion (async I/O) for a previously opened file.

fcntl

```
int(* vfs_handler_t::fcntl) (void *fd, int cmd, va_list ap)
```

Manipulate file control flags on the given file.

fstat

```
int(* vfs_handler_t::fstat) (void *hnd, struct stat *st)
```

Get status information on an already opened file.

ioctl

```
int(* vfs_handler_t::ioctl) (void *hnd, int cmd, va_list ap)
```

Execute a device-specific call on a previously opened file.

link

```
int(* vfs_handler_t::link) (struct vfs_handler *vfs, const char *path1, const char *path2)
```

Create a hard link.

mkdir

```
int(* vfs_handler_t::mkdir) (struct vfs_handler *vfs, const char *fn)
```

Make a directory on the given VFS.

mmap

```
void *(* vfs_handler_t::mmap) (void *fd)
```

"Memory map" a previously opened file

nmmgr

```
nmmgr_handler_t vfs_handler_t::nmmgr
```

Name manager handler header.

open

```
void *(* vfs_handler_t::open) (struct vfs_handler *vfs, const char *fn, int mode)
```

Open a file on the given VFS; return a unique identifier.

poll

```
short(* vfs_handler_t::poll) (void *fd, short events)
```

Check if an event is pending on the given file.

privdata

```
void* vfs_handler_t::privdata
```

Pointer to private data for the handler.

read

```
ssize_t(* vfs_handler_t::read) (void *hnd, void *buffer, size_t cnt)
```

Read from a previously opened file.

readdir

```
dirent_t *(* vfs_handler_t::readdir) (void *hnd)
```

Read the next directory entry in a directory opened with O_DIR.

readlink

```
ssize_t(* vfs_handler_t::readlink) (struct vfs_handler *vfs, const char *path, char *buf, size_t
bufsize)
```

Read the value of a symbolic link.

Note

path will not be passed through realpath() before calling the filesystem-level function. It is also important to not call realpath() in any implementation of this function as it is possible that realpath() will call this function.

rename

```
int(* vfs_handler_t::rename) (struct vfs_handler *vfs, const char *fn1, const char *fn2)
```

Rename/move a file on the given VFS.

rewinddir

```
int(* vfs_handler_t::rewinddir) (void *hnd)
```

Rewind a directory stream to the start.

rmdir

```
int(* vfs_handler_t::rmdir) (struct vfs_handler *vfs, const char *fn)
```

Remove a directory from the given VFS.

seek

```
off_t(* vfs_handler_t::seek) (void *hnd, off_t offset, int whence)
```

Seek in a previously opened file.

seek64

```
_off64_t(* vfs_handler_t::seek64) (void *hnd, _off64_t offset, int whence)
```

Seek in a previously opened file (64-bit offsets)

stat

```
int(* vfs_handler_t::stat) (struct vfs_handler *vfs, const char *path, struct stat *buf, int flag)
```

Get status information on a file on the given VFS.

Note

path will not be passed through realpath() before calling the filesystem-level function. It is also important to not call realpath() in any implementation of this function as it is possible that realpath() will call this function.

symlink

```
int(* vfs_handler_t::symlink) (struct vfs_handler *vfs, const char *path1, const char *path2)
```

Create a symbolic link.

tell

```
off_t(* vfs_handler_t::tell) (void *hnd)
```

Return the current position in a previously opened file.

tell64

```
_off64_t(* vfs_handler_t::tell64) (void *hnd)
```

Return the current position in an opened file (64-bit offset)

total

```
size_t(* vfs_handler_t::total) (void *hnd)
```

Return the total size of a previously opened file.

total64

```
uint64(* vfs_handler_t::total64) (void *hnd)
```

Return the size of an opened file as a 64-bit integer.

unlink

```
int(* vfs_handler_t::unlink) (struct vfs_handler *vfs, const char *fn)
```

Delete a file from the given VFS.

write

```
ssize_t(* vfs_handler_t::write) (void *hnd, const void *buffer, size_t cnt)
```

Write to a previously opened file.

The documentation for this struct was generated from the following file:

- include/kos/fs.h

8.101 vid_mode_t Struct Reference

Video mode structure.

`#include <dc/video.h>`

Data Fields

- int generic
  Generic mode type for vid_set_mode()
- uint16 width
  Width of the display, in pixels.
- uint16 height
  Height of the display, in pixels.
- uint32 flags
  Combination of one or more VID_∗ flags.
- int16 cable_type
  Allowed cable type.
- uint16 pm
  Pixel mode.
- uint16 scanlines
  Number of scanlines.
- uint16 clocks
  Clocks per scanline.
- uint16 bitmapx
  Bitmap window X position.
- uint16 bitmapy
  Bitmap window Y position (automatically increased for PAL)
- uint16 scanint1
  First scanline interrupt position.
- uint16 scanint2
  Second scanline interrupt position (automatically doubled for VGA)
- uint16 borderx1
  Border X starting position.
- uint16 borderx2
  Border X stop position.
- uint16 bordery1
  Border Y starting position.
- uint16 bordery2
  Border Y stop position.
- uint16 fb_curr
  Current framebuffer.
- uint16 fb_count
  Number of framebuffers.
- uint32 fb_base [VID_MAX_FB]
  Offset to framebuffers.

8.101.1 Detailed Description

Video mode structure.

KOS maintains a list of valid video modes internally that correspond to the specific display modes enumeration. Each of them is built of one of these.

8.101.2 Field Documentation

bitmapx

```
uint16 vid_mode_t::bitmapx
```

Bitmap window X position.

bitmapy

```
uint16 vid_mode_t::bitmapy
```

Bitmap window Y position (automatically increased for PAL)

borderx1

```
uint16 vid_mode_t::borderx1
```

Border X starting position.

borderx2

```
uint16 vid_mode_t::borderx2
```

Border X stop position.

bordery1

```
uint16 vid_mode_t::bordery1
```

Border Y starting position.

bordery2

```
uint16 vid_mode_t::bordery2
```

Border Y stop position.

cable_type

```
int16 vid_mode_t::cable_type
```

Allowed cable type.

clocks

```
uint16 vid_mode_t::clocks
```

Clocks per scanline.

fb_base

```
uint32 vid_mode_t::fb_base[VID_MAX_FB]
```

Offset to framebuffers.

fb_count

```
uint16 vid_mode_t::fb_count
```

Number of framebuffers.

fb_curr

```
uint16 vid_mode_t::fb_curr
```

Current framebuffer.

flags

```
uint32 vid_mode_t::flags
```

Combination of one or more VID_* flags.

generic

```
int vid_mode_t::generic
```

Generic mode type for vid_set_mode()

height

```
uint16 vid_mode_t::height
```

Height of the display, in pixels.

pm

```
uint16 vid_mode_t::pm
```

Pixel mode.

scanint1

```
uint16 vid_mode_t::scanint1
```

First scanline interrupt position.

scanint2

```
uint16 vid_mode_t::scanint2
```

Second scanline interrupt position (automatically doubled for VGA)

scanlines

```
uint16 vid_mode_t::scanlines
```

Number of scanlines.

width

```
uint16 vid_mode_t::width
```

Width of the display, in pixels.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/video.h

8.102 vmu_dir_t Struct Reference

VMU FS Directory entries, 32 bytes each.

`#include <dc/vmufs.h>`

Data Fields

- uint8 filetype
  0x00 = no file; 0x33 = data; 0xcc = a game
- uint8 copyprotect
  0x00 = copyable; 0xff = copy protected
- uint16 firstblk
  Location of the first block in the file.
- char filename [12]
  Note: there is no null terminator.
- vmu_timestamp_t timestamp
  File time.
- uint16 filesize
  Size of the file in blocks.
- uint16 hdroff
  Offset of header, in blocks from start of file.
- uint8 dirty
  See header notes.
- uint8 pad1 [3]
  All zeros.

8.102.1 Detailed Description

VMU FS Directory entries, 32 bytes each.

8.102.2 Field Documentation

copyprotect

`uint8 vmu_dir_t::copyprotect`

0x00 = copyable; 0xff = copy protected

dirty

`uint8 vmu_dir_t::dirty`

See header notes.

filename

```
char vmu_dir_t::filename[12]
```

Note: there is no null terminator.

filesize

```
uint16 vmu_dir_t::filesize
```

Size of the file in blocks.

filetype

```
uint8 vmu_dir_t::filetype
```

0x00 = no file; 0x33 = data; 0xcc = a game

firstblk

```
uint16 vmu_dir_t::firstblk
```

Location of the first block in the file.

hdroff

```
uint16 vmu_dir_t::hdroff
```

Offset of header, in blocks from start of file.

pad1

```
uint8 vmu_dir_t::pad1[3]
```

All zeros.

timestamp

```
vmu_timestamp_t vmu_dir_t::timestamp
```

File time.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/vmufs.h

8.103 vmu_hdr_t Struct Reference

Final VMU package type.

```
#include <dc/vmu_pkg.h>
```

Data Fields

- char desc_short [16]
  Space-padded short description.
- char desc_long [32]
  Space-padded long description.
- char app_id [16]
  Null-padded application ID.
- uint16 icon_cnt
  Number of icons.
- uint16 icon_anim_speed
  Icon animation speed.
- uint16 eyecatch_type
  Eyecatch type.
- uint16 crc
  CRC of the file.
- uint32 data_len
  Payload size.
- uint8 reserved [20]
  Reserved (all zero)
- uint16 icon_pal [16]
  Icon palette (ARGB4444)

8.103.1 Detailed Description

Final VMU package type.

This structure will be written into the file itself, not vmu_pkg_t.

8.103.2 Field Documentation

app_id

```
char vmu_hdr_t::app_id[16]
```

Null-padded application ID.

crc

```
uint16 vmu_hdr_t::crc
```

CRC of the file.

data_len

```
uint32 vmu_hdr_t::data_len
```

Payload size.

desc_long

```
char vmu_hdr_t::desc_long[32]
```

Space-padded long description.

desc_short

```
char vmu_hdr_t::desc_short[16]
```

Space-padded short description.

eyecatch_type

```
uint16 vmu_hdr_t::eyecatch_type
```

Eyecatch type.

icon_anim_speed

```
uint16 vmu_hdr_t::icon_anim_speed
```

Icon animation speed.

icon_cnt

```
uint16 vmu_hdr_t::icon_cnt
```

Number of icons.

icon_pal

```
uint16 vmu_hdr_t::icon_pal[16]
```

Icon palette (ARGB4444)

reserved

```
uint8 vmu_hdr_t::reserved[20]
```

Reserved (all zero)

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/vmu_pkg.h

8.104 vmu_pkg_t Struct Reference

VMU Package type.

```
#include <dc/vmu_pkg.h>
```

Data Fields

- char desc_short [20]
  Short file description.
- char desc_long [36]
  Long file description.
- char app_id [20]
  Application ID.
- int icon_cnt
  Number of icons.
- int icon_anim_speed
  Icon animation speed.
- int eyecatch_type
  "Eyecatch" type
- int data_len
  Number of data (payload) bytes.
- uint16 icon_pal [16]
  Icon palette (ARGB4444)
- const uint8 * icon_data
  *512*n bytes of icon data*
- const uint8 * eyecatch_data
  Eyecatch data.
- const uint8 * data
  Payload data.

8.104.1 Detailed Description

VMU Package type.

Anyone wanting to package a VMU file should create one of these somewhere; eventually it will be turned into a flat file that you can save using fs_vmu.

8.104.2 Field Documentation

app_id

```
char vmu_pkg_t::app_id[20]
```

Application ID.

data

```
const uint8* vmu_pkg_t::data
```

Payload data.

data_len

```
int vmu_pkg_t::data_len
```

Number of data (payload) bytes.

desc_long

```
char vmu_pkg_t::desc_long[36]
```

Long file description.

desc_short

```
char vmu_pkg_t::desc_short[20]
```

Short file description.

eyecatch_data

```
const uint8* vmu_pkg_t::eyecatch_data
```

Eyecatch data.

eyecatch_type

```
int vmu_pkg_t::eyecatch_type
```

"Eyecatch" type

icon_anim_speed

```
int vmu_pkg_t::icon_anim_speed
```

Icon animation speed.

icon_cnt

```
int vmu_pkg_t::icon_cnt
```

Number of icons.

icon_data

```
const uint8* vmu_pkg_t::icon_data
```

512∗n bytes of icon data

icon_pal

```
uint16 vmu_pkg_t::icon_pal[16]
```

Icon palette (ARGB4444)

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/vmu_pkg.h

8.105 vmu_root_t Struct Reference

VMU FS Root block layout.

```
#include <dc/vmufs.h>
```

Data Fields

- uint8 magic [16]
  All should contain 0x55.
- uint8 use_custom
  0 = standard, 1 = custom
- uint8 custom_color [4]
  blue, green, red, alpha
- uint8 pad1 [27]
  All zeros.
- vmu_timestamp_t timestamp
  BCD timestamp.
- uint8 pad2 [8]
  All zeros.
- uint8 unk1 [6]
  ???
- uint16 fat_loc
  FAT location.
- uint16 fat_size
  FAT size in blocks.
- uint16 dir_loc
  Directory location.
- uint16 dir_size
  Directory size in blocks.
- uint16 icon_shape
  Icon shape for this VMS.
- uint16 blk_cnt
  Number of user blocks.
- uint8 unk2 [430]
  ???

8.105.1 Detailed Description

VMU FS Root block layout.

8.105.2 Field Documentation

blk_cnt

```
uint16 vmu_root_t::blk_cnt
```

Number of user blocks.

custom_color

```
uint8 vmu_root_t::custom_color[4]
```

blue, green, red, alpha

dir_loc

```
uint16 vmu_root_t::dir_loc
```

Directory location.

dir_size

```
uint16 vmu_root_t::dir_size
```

Directory size in blocks.

fat_loc

```
uint16 vmu_root_t::fat_loc
```

FAT location.

fat_size

```
uint16 vmu_root_t::fat_size
```

FAT size in blocks.

icon_shape

```
uint16 vmu_root_t::icon_shape
```

Icon shape for this VMS.

magic

```
uint8 vmu_root_t::magic[16]
```

All should contain 0x55.

pad1

```
uint8 vmu_root_t::pad1[27]
```

All zeros.

pad2

```
uint8 vmu_root_t::pad2[8]
```

All zeros.

timestamp

```
vmu_timestamp_t vmu_root_t::timestamp
```

BCD timestamp.

unk1

```
uint8 vmu_root_t::unk1[6]
```

???

unk2

```
uint8 vmu_root_t::unk2[430]
```

???

use_custom

```
uint8 vmu_root_t::use_custom
```

0 = standard, 1 = custom

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/vmufs.h

8.106 vmu_state_t Union Reference

VMU's "civilized" state data: 0 = RELEASED, 1 = PRESSED.

```
#include <vmu.h>
```

Data Fields

- uint8_t buttons
  Combined button state mask.
- struct {
  uint8_t dpad_up: 1
  Dpad Up button state.
  uint8_t dpad_down: 1
  Dpad Down button state.
  uint8_t dpad_left: 1
  Dpad Left button state.
  uint8_t dpad_right: 1
  Dpad Right button state.
  uint8_t a: 1
  'A' button state
  uint8_t b: 1
  'B' button state
  uint8_t mode: 1
  Mode button state.
  uint8_t sleep: 1
  Sleep button state.
  };

8.106.1 Detailed Description

VMU's "civilized" state data: 0 = RELEASED, 1 = PRESSED.

Note

The Dpad buttons are automatically reoriented for you depending on which direction the VMU is facing in a particular type of controller.

8.106.2 Field Documentation

[struct]

```
struct { ... } vmu_state_t
```

a

`uint8_t vmu_state_t::a`

'A' button state

b

`uint8_t vmu_state_t::b`

'B' button state

buttons

`uint8_t vmu_state_t::buttons`

Combined button state mask.

dpad_down

`uint8_t vmu_state_t::dpad_down`

Dpad Down button state.

dpad_left

`uint8_t vmu_state_t::dpad_left`

Dpad Left button state.

dpad_right

`uint8_t vmu_state_t::dpad_right`

Dpad Right button state.

dpad_up

`uint8_t vmu_state_t::dpad_up`

Dpad Up button state.

mode

```
uint8_t vmu_state_t::mode
```

Mode button state.

sleep

```
uint8_t vmu_state_t::sleep
```

Sleep button state.

The documentation for this union was generated from the following file:

- kernel/arch/dreamcast/include/dc/maple/vmu.h

8.107 vmu_timestamp_t Struct Reference

BCD timestamp, used several places in the vmufs.

```
#include <dc/vmufs.h>
```

Data Fields

- uint8 cent
  Century.
- uint8 year
  Year, within century.
- uint8 month
  Month of the year.
- uint8 day
  Day of the month.
- uint8 hour
  Hour of the day.
- uint8 min
  Minutes.
- uint8 sec
  Seconds.
- uint8 dow
  Day of week (0 = monday, etc)

8.107.1 Detailed Description

BCD timestamp, used several places in the vmufs.

8.107.2 Field Documentation

cent

```
uint8 vmu_timestamp_t::cent
```

Century.

day

```
uint8 vmu_timestamp_t::day
```

Day of the month.

dow

```
uint8 vmu_timestamp_t::dow
```

Day of week (0 = monday, etc)

hour

```
uint8 vmu_timestamp_t::hour
```

Hour of the day.

min

```
uint8 vmu_timestamp_t::min
```

Minutes.

month

```
uint8 vmu_timestamp_t::month
```

Month of the year.

sec

```
uint8 vmu_timestamp_t::sec
```

Seconds.

year

```
uint8 vmu_timestamp_t::year
```

Year, within century.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/vmufs.h

8.108 vmufb_font_t Struct Reference

VMU framebuffer font meta-data.

```
#include <dc/vmu_fb.h>
```

Data Fields

- unsigned int w
  Character width in pixels.
- unsigned int h
  Character height in pixels.
- unsigned int stride
  Size of one character in bytes.
- const char * fontdata
  Pointer to the font data.

8.108.1 Detailed Description

VMU framebuffer font meta-data.

8.108.2 Field Documentation

fontdata

```
const char* vmufb_font_t::fontdata
```

Pointer to the font data.

h

```
unsigned int vmufb_font_t::h
```

Character height in pixels.

stride

```
unsigned int vmufb_font_t::stride
```

Size of one character in bytes.

w

```
unsigned int vmufb_font_t::w
```

Character width in pixels.

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/vmu_fb.h

8.109 vmufb_t Struct Reference

VMU framebuffer.

```
#include <dc/vmu_fb.h>
```

Data Fields

- uint32_t data [48]

8.109.1 Detailed Description

VMU framebuffer.

This object contains a 48x32 monochrome framebuffer. It can be painted to, or displayed one the VMUs connected to the system, using the API below.

8.109.2 Field Documentation

data

```
uint32_t vmufb_t::data[48]
```

The documentation for this struct was generated from the following file:

- kernel/arch/dreamcast/include/dc/vmu_fb.h

9 File Documentation

9.1 addons/include/ext2/fs_ext2.h File Reference

VFS interface for an ext2 filesystem.

```
#include <sys/cdefs.h>
#include <stdint.h>
#include <kos/blockdev.h>
```

Macros

- #define FS_EXT2_MOUNT_READONLY 0x00000000

  Mount read-only.
- #define FS_EXT2_MOUNT_READWRITE 0x00000001

  Mount read-write.

Functions

- int fs_ext2_init (void)

  Initialize fs_ext2.
- int fs_ext2_shutdown (void)

  Shut down fs_ext2.
- int fs_ext2_mount (const char *mp, kos_blockdev_t *dev, uint32_t flags)

  Mount an ext2 filesystem in the VFS.
- int fs_ext2_unmount (const char *mp)

  Unmount an ext2 filesystem from the VFS.
- int fs_ext2_sync (const char *mp)

  Sync an ext2 filesystem, flushing all pending writes to the block device.

9.1.1 Detailed Description

VFS interface for an ext2 filesystem.

This file defines the public interface to add support for the Second Extended Filesystem (ext2) to KOS' VFS. ext2 is one of the many filesystems that is natively supported by Linux, and was the main filesystem used by most Linux installations pretty much until the creation of the ext3 filesystem.

The KOS ext2 driver was designed with two purposes. First of all, this fs was added to provide a filesystem for use on SD cards used with the Dreamcast SD adapter. ext2 was chosen for this purpose for a bunch of reasons, but probably the biggest one was the non-patent-encumbered nature of ext2 and the availability of programs/drivers to read ext2 on most major OSes available for PCs today. The second purpose of this filesystem driver is to provide an alternative for fs_romdisk when swapping out disk images at runtime. Basically, if a disk image is useful to you, but cacheing it fully in memory is not important, then you could rig up a relatively simple interface with this filesystem driver.

Note that there is a lower-level interface sitting underneath of this layer. This lower-level interface (simply called ext2fs) should not generally be used by any normal applications. As of this point, it is completely non thread-safe and the fs_ext2 layer takes extreme care to overcome those issues with the lower-level interface. Over time, I may fix the thread-safety issues in ext2fs, but that is not particularly high on my priority list at the moment. There shouldn't really be a reason to work directly with the ext2fs layer anyway, as this layer should give you everything you need by interfacing with the VFS in the normal fashion.

There's one final note that I should make. Everything in fs_ext2 and ext2fs is licensed under the same license as the rest of KOS. None of it was derived from GPLed sources. Pretty much all of what's in ext2fs was written based on the documentation at http://www.nongnu.org/ext2-doc/ .

Author

Lawrence Sebald

9.1.2 Function Documentation

fs_ext2_init()

```
int fs_ext2_init (
            void  )
```

Initialize fs_ext2.

This function initializes fs_ext2, preparing various internal structures for use.

Return values

0	On success. No error conditions currently defined.

fs_ext2_mount()

```
int fs_ext2_mount (
            const char * mp,
            kos_blockdev_t * dev,
            uint32_t flags )
```

Mount an ext2 filesystem in the VFS.

This function mounts an ext2 filesystem to the specified mount point on the VFS. This function will detect whether or not an ext2 filesystem exists on the given block device and mount it only if there is actually an ext2 filesystem.

Parameters

mp	The path to mount the filesystem at.
dev	The block device containing the filesystem.
flags	Mount flags. Bitwise OR of values from ext2_mount_flags

Return values

0	On success.
-1	On error.

fs_ext2_shutdown()

```
int fs_ext2_shutdown (
            void  )
```

Shut down fs_ext2.

This function shuts down fs_ext2, basically undoing what fs_ext2_init() did.

Return values

0	On success. No error conditions currently defined.

fs_ext2_sync()

```
int fs_ext2_sync (
            const char * mp )
```

Sync an ext2 filesystem, flushing all pending writes to the block device.

This function completes all pending writes on the filesystem, making sure all data and metadata are in a consistent state on the block device. As both inode and block writes are normally postponed until they are either evicted from the cache or the filesystem is unmounted, doing this periodically may be a good idea if there is a chance that the filesystem will not be unmounted cleanly.

Parameters

mp	The mount point of the filesystem to be synced.

Return values

0	On success.
-1	On error.

Note

This function has no effect if the filesystem was mounted read-only.

fs_ext2_unmount()

```
int fs_ext2_unmount (
            const char * mp )
```

Unmount an ext2 filesystem from the VFS.

This function unmoutns an ext2 filesystem that was previously mounted by the fs_ext2_mount() function.

Parameters

mp	The mount point of the filesystem to be unmounted.

Return values

0	On success.
-1	On error.

9.2 fs_ext2.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    ext2/fs_ext2.h
00004    Copyright (C) 2012, 2013 Lawrence Sebald
00005 */
00006
00007 #ifndef __EXT2_FS_EXT2_H
00008 #define __EXT2_FS_EXT2_H
00009
00010 #include <sys/cdefs.h>
00011 __BEGIN_DECLS
00012
00013 #include <stdint.h>
00014 #include <kos/blockdev.h>
00015
00016 /** \file   ext2/fs_ext2.h
00017     \brief  VFS interface for an ext2 filesystem.
00018
00019     This file defines the public interface to add support for the Second
00020     Extended Filesystem (ext2) to KOS' VFS. ext2 is one of the many filesystems
00021     that is natively supported by Linux, and was the main filesystem used by
00022     most Linux installations pretty much until the creation of the ext3
00023     filesystem.
00024
00025     The KOS ext2 driver was designed with two purposes. First of all, this fs
00026     was added to provide a filesystem for use on SD cards used with the
00027     Dreamcast SD adapter. ext2 was chosen for this purpose for a bunch of
00028     reasons, but probably the biggest one was the non-patent-encumbered nature
00029     of ext2 and the availability of programs/drivers to read ext2 on most major
00030     OSes available for PCs today. The second purpose of this filesystem driver
00031     is to provide an alternative for fs_romdisk when swapping out disk images at
00032     runtime. Basically, if a disk image is useful to you, but cacheing it fully
00033     in memory is not important, then you could rig up a relatively simple
00034     interface with this filesystem driver.
00035
00036     Note that there is a lower-level interface sitting underneath of this layer.
00037     This lower-level interface (simply called ext2fs) should not generally be
00038     used by any normal applications. As of this point, it is completely non
00039     thread-safe and the fs_ext2 layer takes extreme care to overcome those
00040     issues with the lower-level interface. Over time, I may fix the thread-
00041     safety issues in ext2fs, but that is not particularly high on my priority
00042     list at the moment. There shouldn't really be a reason to work directly with
```

```
00043     the ext2fs layer anyway, as this layer should give you everything you need
00044     by interfacing with the VFS in the normal fashion.
00045
00046     There's one final note that I should make. Everything in fs_ext2 and ext2fs
00047     is licensed under the same license as the rest of KOS. None of it was
00048     derived from GPLed sources. Pretty much all of what's in ext2fs was written
00049     based on the documentation at http://www.nongnu.org/ext2-doc/ .
00050
00051     \author Lawrence Sebald
00052 */
00053
00054 /** \brief  Initialize fs_ext2.
00055
00056     This function initializes fs_ext2, preparing various internal structures for
00057     use.
00058
00059     \retval 0           On success. No error conditions currently defined.
00060 */
00061 int fs_ext2_init(void);
00062
00063 /** \brief  Shut down fs_ext2.
00064
00065     This function shuts down fs_ext2, basically undoing what fs_ext2_init() did.
00066
00067     \retval 0           On success. No error conditions currently defined.
00068 */
00069 int fs_ext2_shutdown(void);
00070
00071 /** \defgroup ext2_mount_flags          Mount flags for fs_ext2
00072
00073     These values are the valid flags that can be passed for the flags parameter
00074     to the fs_ext2_mount() function. Note that these can be combined, except for
00075     the read-only flag.
00076
00077     Also, it is not possible to mount some filesystems as read-write. For
00078     instance, if the filesystem was marked as not cleanly unmounted (from Linux
00079     itself), the driver will fail to mount the device as read-write. Also, if
00080     the block device does not support writing, then the filesystem will not be
00081     mounted as read-write (for obvious reasons).
00082
00083     These should stay synchronized with the ones in ext2fs.h.
00084
00085     @{
00086 */
00087 #define FS_EXT2_MOUNT_READONLY      0x00000000  /**< \brief Mount read-only */
00088 #define FS_EXT2_MOUNT_READWRITE     0x00000001  /**< \brief Mount read-write */
00089 /** @} */
00090
00091 /** \brief  Mount an ext2 filesystem in the VFS.
00092
00093     This function mounts an ext2 filesystem to the specified mount point on the
00094     VFS. This function will detect whether or not an ext2 filesystem exists on
00095     the given block device and mount it only if there is actually an ext2
00096     filesystem.
00097
00098     \param  mp          The path to mount the filesystem at.
00099     \param  dev         The block device containing the filesystem.
00100     \param  flags       Mount flags. Bitwise OR of values from ext2_mount_flags
00101     \retval 0           On success.
00102     \retval -1          On error.
00103 */
00104 int fs_ext2_mount(const char *mp, kos_blockdev_t *dev, uint32_t flags);
00105
00106 /** \brief  Unmount an ext2 filesystem from the VFS.
00107
00108     This function unmoutns an ext2 filesystem that was previously mounted by the
00109     fs_ext2_mount() function.
00110
00111     \param  mp          The mount point of the filesystem to be unmounted.
00112     \retval 0           On success.
00113     \retval -1          On error.
00114 */
00115 int fs_ext2_unmount(const char *mp);
00116
00117 /** \brief  Sync an ext2 filesystem, flushing all pending writes to the block
00118             device.
00119
00120     This function completes all pending writes on the filesystem, making sure
00121     all data and metadata are in a consistent state on the block device. As both
00122     inode and block writes are normally postponed until they are either evicted
00123     from the cache or the filesystem is unmounted, doing this periodically may
```

```
00124     be a good idea if there is a chance that the filesystem will not be
00125     unmounted cleanly.
00126
00127     \param  mp          The mount point of the filesystem to be synced.
00128     \retval 0           On success.
00129     \retval -1          On error.
00130
00131     \note   This function has no effect if the filesystem was mounted read-only.
00132 */
00133 int fs_ext2_sync(const char *mp);
00134
00135 __END_DECLS
00136 #endif /* !__EXT2_FS_EXT2_H */
```

9.3 addons/include/fat/fs_fat.h File Reference

VFS interface for a FAT filesystem.

```
#include <sys/cdefs.h>
#include <stdint.h>
#include <kos/blockdev.h>
```

Macros

- #define FS_FAT_MOUNT_READONLY 0x00000000

  Mount read-only.
- #define FS_FAT_MOUNT_READWRITE 0x00000001

  Mount read-write.

Functions

- int fs_fat_init (void)

  Initialize fs_fat.
- int fs_fat_shutdown (void)

  Shut down fs_fat.
- int fs_fat_mount (const char *mp, kos_blockdev_t *dev, uint32_t flags)

  Mount a FAT filesystem in the VFS.
- int fs_fat_unmount (const char *mp)

  Unmount a FAT filesystem from the VFS.
- int fs_fat_sync (const char *mp)

  Sync a FAT filesystem, flushing all pending writes to the block device.

9.3.1 Detailed Description

VFS interface for a FAT filesystem.

This file defines the public interface to add support for the FAT filesystem, as in common use on all kinds of systems and popularized by MS-DOS and Windows. This interface supports FAT12, FAT16, and FAT32, with both short and long names.

Note that there is a lower-level interface sitting underneath of this layer. This lower-level interface (simply called fatfs) should not generally be used by any normal applications. As of this point, it is completely non thread-safe and the fs_fat layer takes extreme care to overcome those issues with the lower-level interface. Over time, I may fix the thread- safety issues in fatfs, but that is not particularly high on my priority list at the moment. There shouldn't really be a reason to work directly with the fatfs layer anyway, as this layer should give you everything you need by interfacing with the VFS in the normal fashion.

Author

Lawrence Sebald

9.3.2 Function Documentation

fs_fat_init()

```
int fs_fat_init (
            void  )
```

Initialize fs_fat.

This function initializes fs_fat, preparing various internal structures for use.

Return values

0	On success. No error conditions currently defined.

fs_fat_mount()

```
int fs_fat_mount (
            const char * mp,
            kos_blockdev_t * dev,
            uint32_t flags )
```

Mount a FAT filesystem in the VFS.

This function mounts an fat filesystem to the specified mount point on the VFS. This function will detect whether or not an FAT filesystem exists on the given block device and mount it only if there is actually an FAT filesystem.

Parameters

mp	The path to mount the filesystem at.
dev	The block device containing the filesystem.
flags	Mount flags. Bitwise OR of values from fat_mount_flags

Return values

0	On success.
-1	On error.

fs_fat_shutdown()

```
int fs_fat_shutdown (
            void  )
```

Shut down fs_fat.

This function shuts down fs_fat, basically undoing what fs_fat_init() did.

Return values

0	On success. No error conditions currently defined.

fs_fat_sync()

```
int fs_fat_sync (
            const char * mp )
```

Sync a FAT filesystem, flushing all pending writes to the block device.

This function completes all pending writes on the filesystem, making sure all data and metadata are in a consistent state on the block device. As both inode and block writes are normally postponed until they are either evicted from the cache or the filesystem is unmounted, doing this periodically may be a good idea if there is a chance that the filesystem will not be unmounted cleanly.

Parameters

mp	The mount point of the filesystem to be synced.

Return values

0	On success.
-1	On error.

Note

This function has no effect if the filesystem was mounted read-only.

fs_fat_unmount()

```
int fs_fat_unmount (
            const char * mp )
```

Unmount a FAT filesystem from the VFS.

This function unmoutns an FAT filesystem that was previously mounted by the fs_fat_mount() function.

Parameters

mp	The mount point of the filesystem to be unmounted.

Return values

0	On success.
-1	On error.

9.4 fs_fat.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    fat/fs_fat.h
00004    Copyright (C) 2012, 2013, 2019 Lawrence Sebald
00005 */
00006
00007 #ifndef __FAT_FS_FAT_H
00008 #define __FAT_FS_FAT_H
00009
00010 #include <sys/cdefs.h>
00011 __BEGIN_DECLS
00012
00013 #include <stdint.h>
00014 #include <kos/blockdev.h>
00015
00016 /** \file   fat/fs_fat.h
00017     \brief  VFS interface for a FAT filesystem.
00018
00019     This file defines the public interface to add support for the FAT
00020     filesystem, as in common use on all kinds of systems and popularized by
00021     MS-DOS and Windows. This interface supports FAT12, FAT16, and FAT32, with
00022     both short and long names.
00023
00024     Note that there is a lower-level interface sitting underneath of this layer.
00025     This lower-level interface (simply called fatfs) should not generally be
00026     used by any normal applications. As of this point, it is completely non
00027     thread-safe and the fs_fat layer takes extreme care to overcome those
00028     issues with the lower-level interface. Over time, I may fix the thread-
00029     safety issues in fatfs, but that is not particularly high on my priority
00030     list at the moment. There shouldn't really be a reason to work directly with
00031     the fatfs layer anyway, as this layer should give you everything you need
00032     by interfacing with the VFS in the normal fashion.
00033
00034     \author Lawrence Sebald
```

```
00035 */
00036
00037 /** \brief  Initialize fs_fat.
00038
00039     This function initializes fs_fat, preparing various internal structures for
00040     use.
00041
00042     \retval 0           On success. No error conditions currently defined.
00043 */
00044 int fs_fat_init(void);
00045
00046 /** \brief  Shut down fs_fat.
00047
00048     This function shuts down fs_fat, basically undoing what fs_fat_init() did.
00049
00050     \retval 0           On success. No error conditions currently defined.
00051 */
00052 int fs_fat_shutdown(void);
00053
00054 /** \defgroup fat_mount_flags          Mount flags for fs_fat
00055
00056     These values are the valid flags that can be passed for the flags parameter
00057     to the fs_fat_mount() function. Note that these can be combined, except for
00058     the read-only flag.
00059
00060     Also, it is not possible to mount some filesystems as read-write. For
00061     instance, if the filesystem was marked as not cleanly unmounted the driver
00062     will fail to mount the device as read-write. Also, if the block device does
00063     not support writing, then the filesystem will not be mounted as read-write
00064     (for obvious reasons).
00065
00066     These should stay synchronized with the ones in fatfs.h.
00067
00068     @{
00069 */
00070 #define FS_FAT_MOUNT_READONLY       0x00000000  /**< \brief Mount read-only */
00071 #define FS_FAT_MOUNT_READWRITE      0x00000001  /**< \brief Mount read-write */
00072 /** @} */
00073
00074 /** \brief  Mount a FAT filesystem in the VFS.
00075
00076     This function mounts an fat filesystem to the specified mount point on the
00077     VFS. This function will detect whether or not an FAT filesystem exists on
00078     the given block device and mount it only if there is actually an FAT
00079     filesystem.
00080
00081     \param  mp          The path to mount the filesystem at.
00082     \param  dev         The block device containing the filesystem.
00083     \param  flags       Mount flags. Bitwise OR of values from fat_mount_flags
00084     \retval 0           On success.
00085     \retval -1          On error.
00086 */
00087 int fs_fat_mount(const char *mp, kos_blockdev_t *dev, uint32_t flags);
00088
00089 /** \brief  Unmount a FAT filesystem from the VFS.
00090
00091     This function unmoutns an FAT filesystem that was previously mounted by the
00092     fs_fat_mount() function.
00093
00094     \param  mp          The mount point of the filesystem to be unmounted.
00095     \retval 0           On success.
00096     \retval -1          On error.
00097 */
00098 int fs_fat_unmount(const char *mp);
00099
00100 /** \brief  Sync a FAT filesystem, flushing all pending writes to the block
00101             device.
00102
00103     This function completes all pending writes on the filesystem, making sure
00104     all data and metadata are in a consistent state on the block device. As both
00105     inode and block writes are normally postponed until they are either evicted
00106     from the cache or the filesystem is unmounted, doing this periodically may
00107     be a good idea if there is a chance that the filesystem will not be
00108     unmounted cleanly.
00109
00110     \param  mp          The mount point of the filesystem to be synced.
00111     \retval 0           On success.
00112     \retval -1          On error.
00113
00114     \note   This function has no effect if the filesystem was mounted read-only.
00115 */
```

```
00116 int fs_fat_sync(const char *mp);
00117
00118 __END_DECLS
00119 #endif /* !__FAT_FS_FAT_H */
```

9.5 addons/include/kos/bspline.h File Reference

B-Spline curve support.

```
#include <sys/cdefs.h>
#include <dc/vector.h>
```

Functions

- void bspline_coeff (const point_t *pnt)

  Calculate and set b-spline coefficients.
- void bspline_get_point (float t, point_t *p)

  Generate the next point for the current set of coefficients.

9.5.1 Detailed Description

B-Spline curve support.

This module provides utility functions to generate b-spline curves in your program. It is used by passing in a set of control points to bspline_coeff(), and then querying for individual points using bspline_get_point().

Note that this module is NOT thread-safe.

Author

Megan Potter

9.5.2 Function Documentation

bspline_coeff()

```
void bspline_coeff (
            const point_t * pnt )
```

Calculate and set b-spline coefficients.

This function performs the initial setup work of calculating the coefficients needed to generate a b-spline curve for the specified set of points. The calculation is based on a total of 4 points: one previous point, the current point, and two points that occur after the current point.

The current point should be at pnt[0], the previous at pnt[-1], and the future points should be at pnt[1], and pnt[2]. I repeat: pnt[-1] must be a valid point for this to work properly.

Parameters

pnt	The array of points used to calculate the b-spline coefficients.

bspline_get_point()

```
void bspline_get_point (
            float t,
            point_t * p )
```

Generate the next point for the current set of coefficients.

Given a 't' (between 0.0f and 1.0f) this will generate the next point value for the current set of coefficients.

Parameters

t	The "t" value for the b-spline generation function.
p	Storage for the generated point.

9.6 bspline.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    bspline.h
00004    Copyright (C) 2000 Megan Potter
00005
00006 */
00007
00008 #ifndef __KOS_BSPLINE_H
00009 #define __KOS_BSPLINE_H
00010
00011 /** \file   kos/bspline.h
00012     \brief  B-Spline curve support.
00013
00014     This module provides utility functions to generate b-spline curves in your
00015     program. It is used by passing in a set of control points to
00016     bspline_coeff(), and then querying for individual points using
00017     bspline_get_point().
00018
00019     Note that this module is NOT thread-safe.
00020
00021     \author Megan Potter
00022 */
00023
00024 #include <sys/cdefs.h>
00025 __BEGIN_DECLS
00026
00027 #include <dc/vector.h>
00028
00029 /** \brief  Calculate and set b-spline coefficients.
00030
00031     This function performs the initial setup work of calculating the
00032     coefficients needed to generate a b-spline curve for the specified set of
00033     points. The calculation is based on a total of 4 points: one previous point,
00034     the current point, and two points that occur after the current point.
00035
00036     The current point should be at pnt[0], the previous at pnt[-1], and the
00037     future points should be at pnt[1], and pnt[2]. I repeat: pnt[-1] must be a
00038     valid point for this to work properly.
```

```
00039
00040     \param  pnt         The array of points used to calculate the b-spline
00041                         coefficients.
00042 */
00043 void bspline_coeff(const point_t *pnt);
00044
00045 /** \brief  Generate the next point for the current set of coefficients.
00046
00047     Given a 't' (between 0.0f and 1.0f) this will generate the next point value
00048     for the current set of coefficients.
00049
00050     \param  t           The "t" value for the b-spline generation function.
00051     \param  p           Storage for the generated point.
00052 */
00053 void bspline_get_point(float t, point_t *p);
00054
00055 __END_DECLS
00056
00057 #endif  /* __KOS_BSPLINE_H */
```

9.7 addons/include/kos/img.h File Reference

Platform-independent image type.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Data Structures

- struct kos_img_t

  Platform-indpendent image type.

Macros

- #define KOS_IMG_FMT_I(x) ((x) & 0xffff)

  Read the platform-independent half of the format.
- #define KOS_IMG_FMT_D(x) (((x) >> 16) & 0xffff)

  Read the platform-specific half of the format.
- #define KOS_IMG_FMT(i, d) (((i) & 0xffff) | (((d) & 0xffff) << 16))

  Build a format value from a platform-independent half and a platform-specific half of the value.
- #define KOS_IMG_FMT_NONE 0x00

  Undefined or uninitialized format.
- #define KOS_IMG_FMT_RGB888 0x01

  24-bpp interleaved R/G/B bytes.
- #define KOS_IMG_FMT_ARGB8888 0x02

  32-bpp interleaved A/R/G/B bytes.
- #define KOS_IMG_FMT_RGB565 0x03

  16-bpp interleaved R (5 bits), G (6 bits), B (5 bits).
- #define KOS_IMG_FMT_ARGB4444 0x04

  16-bpp interleaved A/R/G/B (4 bits each).
- #define KOS_IMG_FMT_ARGB1555 0x05

  16-bpp interleaved A (1 bit), R (5 bits), G (5 bits), B (5 bits).

- #define KOS_IMG_FMT_PAL4BPP 0x06
  Paletted, 4 bits per pixel (16 colors).
- #define KOS_IMG_FMT_PAL8BPP 0x07
  Paletted, 8 bits per pixel (256 colors).
- #define KOS_IMG_FMT_YUV422 0x08
  8-bit Y (4 bits), U (2 bits), V (2 bits).
- #define KOS_IMG_FMT_BGR565 0x09
  15-bpp interleaved B (5 bits), G (6 bits), R (5 bits).
- #define KOS_IMG_FMT_RGBA8888 0x10
  32-bpp interleaved R/G/B/A bytes.
- #define KOS_IMG_FMT_MASK 0xff
  Basic format mask (not an actual format value).
- #define KOS_IMG_INVERTED_X 0x0100
  X axis of image data is inverted (stored right to left).
- #define KOS_IMG_INVERTED_Y 0x0200
  Y axis of image data is inverted (stored bottom to top).
- #define KOS_IMG_NOT_OWNER 0x0400
  The image is not the owner of the image data buffer.

Functions

- void kos_img_free (kos_img_t *img, int struct_also)
  Free a kos_img_t object.

9.7.1 Detailed Description

Platform-independent image type.

This file provides a platform-independent image type that is designed to hold any sort of textures or other image data. This type contains a very basic description of the image data (width, height, pixel format), as well as the image data itself.

All of the image-loading libraries in kos-ports should provide a function to load the image data into one of these types.

Author

Megan Potter

9.7.2 Function Documentation

kos_img_free()

```
void kos_img_free (
            kos_img_t * img,
            int struct_also )
```

Free a kos_img_t object.

This function frees the data in a kos_img_t object, returning any memory to the heap as appropriate. Optionally, this can also free the object itself, if required.

Parameters

img	The image object to free.
struct_also	Set to non-zero to free the image object itself, as well as any data contained therein.

9.8 img.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/img.h
00004    Copyright (C) 2002 Megan Potter
00005
00006 */
00007
00008 #ifndef __KOS_IMG_H
00009 #define __KOS_IMG_H
00010
00011 /** \file   kos/img.h
00012     \brief  Platform-independent image type.
00013
00014     This file provides a platform-independent image type that is designed to
00015     hold any sort of textures or other image data. This type contains a very
00016     basic description of the image data (width, height, pixel format), as well
00017     as the image data itself.
00018
00019     All of the image-loading libraries in kos-ports should provide a function
00020     to load the image data into one of these types.
00021
00022     \author Megan Potter
00023 */
00024
00025 #include <sys/cdefs.h>
00026 __BEGIN_DECLS
00027
00028 #include <arch/types.h>
00029
00030 /** \brief  Platform-indpendent image type.
00031
00032     You can use this type for textures or whatever you feel it's appropriate
00033     for. "width" and "height" are as you would expect. "format" has a lower-half
00034     which is platform-independent and used to basically describe the contained
00035     data; the upper-half is platform-dependent and can hold anything (so AND it
00036     off if you only want the bottom part).
00037
00038     Note that in some of the more obscure formats (like the paletted formats)
00039     the data interpretation may be platform dependent. Thus we also provide a
00040     data length field.
00041
00042     \headerfile kos/img.h
00043 */
00044 typedef struct kos_img {
00045     void *data;         /**< \brief Image data in the specified format. */
00046     uint32 w;           /**< \brief Width of the image. */
00047     uint32 h;           /**< \brief Height of the image. */
00048     uint32 fmt;         /**< \brief Format of the image data.
00049                              \see   kos_img_fmts
00050                              \see   kos_img_fmt_macros */
00051     uint32 byte_count;  /**< \brief Length of the image data, in bytes. */
00052 } kos_img_t;
00053
00054 /** \defgroup   kos_img_fmt_macros  Macros for accessing the format of an image
00055
00056     These macros provide easy access to the fmt field of a kos_img_t object.
00057
00058     @{
00059 */
00060 /** \brief  Read the platform-independent half of the format.
00061
00062     This macro masks the format of a kos_img_t to give you just the lower half
00063     of the value, which contains the platform-independent half of the format.
00064
```

```
00065     \param  x           An image format (fmt field of a kos_img_t).
00066     \return             The platform-independent half of the format.
00067 */
00068 #define KOS_IMG_FMT_I(x) ((x) & 0xffff)
00069
00070 /** \brief  Read the platform-specific half of the format.
00071
00072     This macro masks the format of a kos_img_t to give you just the upper half
00073     of the value, which contains the platform-specific half of the format.
00074
00075     \param  x           An image format (fmt field of a kos_img_t).
00076     \return             The platform-specific half of the format.
00077 */
00078 #define KOS_IMG_FMT_D(x) (((x) » 16) & 0xffff)
00079
00080 /** \brief  Build a format value from a platform-independent half and a
00081             platform-specific half of the value.
00082
00083     This macro combines the platform-independent and platform-specific portions
00084     of an image format into a value suitable for storing as the fmt field of a
00085     kos_img_t object.
00086
00087     \param  i           The platform-independent half of the format.
00088     \param  d           The platform-specific half of the format. This should
00089                         not be pre-shifted.
00090     \return             A complete image format value, suitable for placing in
00091                         the fmt variable of a kos_img_t.
00092 */
00093 #define KOS_IMG_FMT(i, d) ( ((i) & 0xffff) | (((d) & 0xffff) « 16) )
00094
00095 /** @} */
00096
00097 /** \defgroup   kos_img_fmts        Image format types
00098
00099     This is the list of platform-independent image types that can be used as the
00100     lower-half of the fmt value for a kos_img_t.
00101
00102     @{
00103 */
00104 /** \brief  Undefined or uninitialized format. */
00105 #define KOS_IMG_FMT_NONE        0x00
00106
00107 /** \brief  24-bpp interleaved R/G/B bytes. */
00108 #define KOS_IMG_FMT_RGB888      0x01
00109
00110 /** \brief  32-bpp interleaved A/R/G/B bytes. */
00111 #define KOS_IMG_FMT_ARGB8888    0x02
00112
00113 /** \brief  16-bpp interleaved R (5 bits), G (6 bits), B (5 bits). */
00114 #define KOS_IMG_FMT_RGB565      0x03
00115
00116 /** \brief  16-bpp interleaved A/R/G/B (4 bits each). */
00117 #define KOS_IMG_FMT_ARGB4444    0x04
00118
00119 /** \brief  16-bpp interleaved A (1 bit), R (5 bits), G (5 bits), B (5 bits).
00120     \note   This can also be used for RGB555 (with the top bit ignored). */
00121 #define KOS_IMG_FMT_ARGB1555    0x05
00122
00123 /** \brief  Paletted, 4 bits per pixel (16 colors). */
00124 #define KOS_IMG_FMT_PAL4BPP     0x06
00125
00126 /** \brief  Paletted, 8 bits per pixel (256 colors). */
00127 #define KOS_IMG_FMT_PAL8BPP     0x07
00128
00129 /** \brief  8-bit Y (4 bits), U (2 bits), V (2 bits). */
00130 #define KOS_IMG_FMT_YUV422      0x08
00131
00132 /** \brief  15-bpp interleaved B (5 bits), G (6 bits), R (5 bits). */
00133 #define KOS_IMG_FMT_BGR565      0x09
00134
00135 /** \brief  32-bpp interleaved R/G/B/A bytes. */
00136 #define KOS_IMG_FMT_RGBA8888    0x10
00137
00138 /** \brief  Basic format mask (not an actual format value). */
00139 #define KOS_IMG_FMT_MASK        0xff
00140
00141 /** \brief  X axis of image data is inverted (stored right to left). */
00142 #define KOS_IMG_INVERTED_X      0x0100
00143
00144 /** \brief  Y axis of image data is inverted (stored bottom to top). */
00145 #define KOS_IMG_INVERTED_Y      0x0200
```

```
00146
00147 /** \brief  The image is not the owner of the image data buffer.
00148
00149     This generally implies that the image data is stored in ROM and thus cannot
00150     be freed.
00151 */
00152 #define KOS_IMG_NOT_OWNER       0x0400
00153
00154 /** @} */
00155
00156 /** \brief  Free a kos_img_t object.
00157
00158     This function frees the data in a kos_img_t object, returning any memory to
00159     the heap as appropriate. Optionally, this can also free the object itself,
00160     if required.
00161
00162     \param  img             The image object to free.
00163     \param  struct_also     Set to non-zero to free the image object itself,
00164                             as well as any data contained therein.
00165 */
00166 void kos_img_free(kos_img_t *img, int struct_also);
00167
00168 __END_DECLS
00169
00170 #endif  /* __KOS_IMG_H */
00171
```

9.9 addons/include/kos/md5.h File Reference

Message Digest 5 (MD5) hashing support.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Data Structures

- struct kos_md5_cxt_t

  MD5 context.

Functions

- void kos_md5_start (kos_md5_cxt_t *cxt)

  Initialize a MD5 context.
- void kos_md5_hash_block (kos_md5_cxt_t *cxt, const uint8 *input, uint32 size)

  Hash a block of data with MD5.
- void kos_md5_finish (kos_md5_cxt_t *cxt, uint8 output[16])

  Complete a MD5 hash.
- void kos_md5 (const uint8 *input, uint32 size, uint8 output[16])

  Compute the hash of a block of data with MD5.

9.9.1 Detailed Description

Message Digest 5 (MD5) hashing support.

This file provides the functionality to compute MD5 hashes over any data buffer. While MD5 isn't considered a safe cryptographic hash any more, it still has its uses.

Author

Lawrence Sebald

9.9.2 Function Documentation

kos_md5()

```
void kos_md5 (
            const uint8 * input,
            uint32 size,
            uint8 output[16] )
```

Compute the hash of a block of data with MD5.

This function is used to hash a full block of data without messing around with any contexts or anything else of the sort. This is appropriate if you have all the data you want to hash readily available. It takes care of all of the context setup and teardown for you.

Parameters

input	The data to hash.
size	The number of bytes of input data passed in.
output	Where to store the final message digest.

kos_md5_finish()

```
void kos_md5_finish (
            kos_md5_cxt_t * cxt,
            uint8 output[16] )
```

Complete a MD5 hash.

This function computes the final MD5 hash of the context passed in, returning the completed digest in the output parameter.

Parameters

cxt	The MD5 context to finalize.
output	Where to store the final digest.

kos_md5_hash_block()

```
void kos_md5_hash_block (
            kos_md5_cxt_t * cxt,
            const uint8 * input,
            uint32 size )
```

Hash a block of data with MD5.

This function is used to hash the block of data input into the function with MD5, updating the state context as appropriate. If the data does not fill an entire block of 64-bytes (or there is left-over data), it will be stored in the context for hashing with a future block. Thus, do not attempt to read the intermediate hash value, as it will not be complete.

Parameters

cxt	The MD5 context to use.
input	The block of data to hash.
size	The number of bytes of input data passed in.

kos_md5_start()

```
void kos_md5_start (
            kos_md5_cxt_t * cxt )
```

Initialize a MD5 context.

This function initializes the context passed in to the initial state needed for computing a MD5 hash. You must call this function to initialize the state variables before attempting to hash any blocks of data.

Parameters

cxt	The MD5 context to initialize.

9.10 md5.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/md5.h
00004    Copyright (C) 2010 Lawrence Sebald
00005 */
00006
00007 #ifndef __KOS_MD5_H
00008 #define __KOS_MD5_H
00009
00010 /** \file   kos/md5.h
00011     \brief  Message Digest 5 (MD5) hashing support.
00012
00013     This file provides the functionality to compute MD5 hashes over any data
00014     buffer. While MD5 isn't considered a safe cryptographic hash any more, it
00015     still has its uses.
00016
00017     \author Lawrence Sebald
00018 */
00019
00020 #include <sys/cdefs.h>
00021 __BEGIN_DECLS
00022
00023 #include <arch/types.h>
00024
00025 /** \brief  MD5 context.
00026
00027     This structure contains the variables needed to maintain the internal state
00028     of the MD5 code. You should not manipulate these variables manually, but
00029     rather use the kos_md5_* functions to do everything you need.
00030
```

```
00031     \headerfile kos/md5.h
00032 */
00033 typedef struct kos_md5_cxt {
00034     uint64 size;        /**< \brief Size of the data in buf. */
00035     uint32 hash[4];     /**< \brief Intermediate hash value. */
00036     uint8  buf[64];     /**< \brief Temporary storage of values to be hashed. */
00037 } kos_md5_cxt_t;
00038
00039 /** \brief  Initialize a MD5 context.
00040
00041     This function initializes the context passed in to the initial state needed
00042     for computing a MD5 hash. You must call this function to initialize the
00043     state variables before attempting to hash any blocks of data.
00044
00045     \param  cxt         The MD5 context to initialize.
00046 */
00047 void kos_md5_start(kos_md5_cxt_t *cxt);
00048
00049 /** \brief  Hash a block of data with MD5.
00050
00051     This function is used to hash the block of data input into the function with
00052     MD5, updating the state context as appropriate. If the data does not fill an
00053     entire block of 64-bytes (or there is left-over data), it will be stored in
00054     the context for hashing with a future block. Thus, do not attempt to read
00055     the intermediate hash value, as it will not be complete.
00056
00057     \param  cxt         The MD5 context to use.
00058     \param  input       The block of data to hash.
00059     \param  size        The number of bytes of input data passed in.
00060 */
00061 void kos_md5_hash_block(kos_md5_cxt_t *cxt, const uint8 *input, uint32 size);
00062
00063 /** \brief  Complete a MD5 hash.
00064
00065     This function computes the final MD5 hash of the context passed in,
00066     returning the completed digest in the output parameter.
00067
00068     \param  cxt         The MD5 context to finalize.
00069     \param  output      Where to store the final digest.
00070 */
00071 void kos_md5_finish(kos_md5_cxt_t *cxt, uint8 output[16]);
00072
00073 /** \brief  Compute the hash of a block of data with MD5.
00074
00075     This function is used to hash a full block of data without messing around
00076     with any contexts or anything else of the sort. This is appropriate if you
00077     have all the data you want to hash readily available. It takes care of all
00078     of the context setup and teardown for you.
00079
00080     \param  input       The data to hash.
00081     \param  size        The number of bytes of input data passed in.
00082     \param  output      Where to store the final message digest.
00083 */
00084 void kos_md5(const uint8 *input, uint32 size, uint8 output[16]);
00085
00086 __END_DECLS
00087
00088 #endif /* !__KOS_MD5_H */
```

9.11 addons/include/kos/netcfg.h File Reference

Network configuration interface.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Data Structures

- struct netcfg_t

  Network configuration information.

Macros

- #define NETCFG_METHOD_DHCP 0
  Use DHCP to configure.
- #define NETCFG_METHOD_STATIC 1
  Static network configuration.
- #define NETCFG_METHOD_PPPOE 4
  Use PPPoE.
- #define NETCFG_SRC_VMU 0
  Read from a VMU.
- #define NETCFG_SRC_FLASH 1
  Read from the flashrom.
- #define NETCFG_SRC_CWD 2
  Read from the working directory.
- #define NETCFG_SRC_CDROOT 3
  Read from the root of the CD.

Functions

- int netcfg_load_from (const char *fn, netcfg_t *out)
  Load network configuration from a file.
- int netcfg_load_flash (netcfg_t *out)
  Load network configuration from the Dreamcast's flashrom.
- int netcfg_load (netcfg_t *out)
  Load network configuration.
- int netcfg_save_to (const char *fn, const netcfg_t *cfg)
  Save network configuration to a file.
- int netcfg_save (const netcfg_t *cfg)
  Save network configuration to the first available VMU.

9.11.1 Detailed Description

Network configuration interface.

This file provides a common interface for reading and writing the network configuration on KOS. The interface can read from the flashrom on the Dreamcast or from a file (such as on a VMU or the like), and can write data back to a file.

The data that is written out by this code is written in a relatively easy to parse text-based format.

Author

Megan Potter

9.11.2 Function Documentation

netcfg_load()

```
int netcfg_load (
            netcfg_t * out )
```

Load network configuration.

This function loads the network configuration data, searching in multiple locations to attempt to find a file with a stored configurtion. This function will attempt to read the configuration from each VMU first (from a file named net.cfg), then it will try the flashrom, followed by the current working directory, and lastly the root of the CD.

Parameters

out	Buffer to store the parsed configuraiton.

Returns

0 on success, <0 on failure.

netcfg_load_flash()

```
int netcfg_load_flash (
            netcfg_t * out )
```

Load network configuration from the Dreamcast's flashrom.

This function loads the network configuration that is stored in flashrom of the Dreamcast, parsing it into a netcfg_t.

Parameters

out	Buffer to store the parsed configuration.

Returns

0 on success, <0 on failure.

Note

This currently does not read the configuration stored by the PlanetWeb browser at all.

netcfg_load_from()

```
int netcfg_load_from (
            const char * fn,
            netcfg_t * out )
```

Load network configuration from a file.

This function loads the network configuration that is stored in the given file to the network configuration structure passed in. This function will also handle files on the VMU with the VMU specific header attached without any extra work required.

Parameters

fn	The file to read the configuration from.
out	Buffer to store the parsed configuration.

Returns

0 on success, <0 on failure.

netcfg_save()

```
int netcfg_save (
            const netcfg_t * cfg )
```

Save network configuration to the first available VMU.

This function saves the network configuration to first VMU that it finds. It will not retry if the first VMU doesn't have enough space to hold the file.

Parameters

cfg	The configuration to save.

Returns

0 on success, <0 on failure.

netcfg_save_to()

```
int netcfg_save_to (
            const char * fn,
            const netcfg_t * cfg )
```

Save network configuration to a file.

This function saves the network configuration to the specified file. This function will automatically prepend the appropriate header if it is saved to a VMU.

Parameters

fn	The file to save to.
cfg	The configuration to save.

Returns

0 on success, <0 on failure.

9.12 netcfg.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/netcfg.h
00004    Copyright (C) 2003 Megan Potter
00005
00006 */
00007
00008 #ifndef __KOS_NETCFG_H
00009 #define __KOS_NETCFG_H
00010
00011 /** \file   kos/netcfg.h
00012     \brief  Network configuration interface.
00013
00014     This file provides a common interface for reading and writing the network
00015     configuration on KOS. The interface can read from the flashrom on the
00016     Dreamcast or from a file (such as on a VMU or the like), and can write data
00017     back to a file.
00018
00019     The data that is written out by this code is written in a relatively easy to
00020     parse text-based format.
00021
00022     \author Megan Potter
00023 */
00024
00025 #include <sys/cdefs.h>
00026 __BEGIN_DECLS
00027
00028 #include <arch/types.h>
00029
00030 /** \defgroup   netcfg_methods  Network connection methods
00031
00032     These constants give the list of network connection methods that are
00033     supported by the netcfg_t type. One of these will be in the method field of
00034     objects of that type.
00035
00036     @{
00037 */
00038 #define NETCFG_METHOD_DHCP      0   /**< \brief Use DHCP to configure. */
00039 #define NETCFG_METHOD_STATIC    1   /**< \brief Static network configuration. */
00040 #define NETCFG_METHOD_PPPOE     4   /**< \brief Use PPPoE. */
00041 /** @} */
00042
00043 /** \defgroup   netcfg_srcs     Network configuration sources
00044
00045     These constants give the list of places that the network configuration might
00046     be read from. One of these constants should be in the src field of objects
00047     of type netcfg_t.
00048
00049     @{
00050 */
00051 #define NETCFG_SRC_VMU      0   /**< \brief Read from a VMU. */
00052 #define NETCFG_SRC_FLASH    1   /**< \brief Read from the flashrom. */
00053 #define NETCFG_SRC_CWD      2   /**< \brief Read from the working directory. */
00054 #define NETCFG_SRC_CDROOT   3   /**< \brief Read from the root of the CD. */
00055 /** @} */
00056
00057 /** \brief  Network configuration information.
00058
00059     This structure contains information about the network configuration of the
00060     system, as set up by the user.
00061
00062     \headerfile kos/netcfg.h
00063 */
00064 typedef struct netcfg {
00065     /** \brief  Where was this configuration read from?
00066         \see    netcfg_srcs
00067     */
00068     int src;
00069
00070     /** \brief  How should the network be configured?
00071         \see    netcfg_methods
00072     */
00073     int method;
00074
00075     uint32 ip;              /**< \brief IPv4 address of the console */
00076     uint32 gateway;         /**< \brief IPv4 address of the gateway/router. */
00077     uint32 netmask;         /**< \brief Network mask for the local net. */
00078     uint32 broadcast;       /**< \brief Broadcast address for the local net. */
00079     uint32 dns[2];          /**< \brief IPv4 address of the DNS servers. */
00080     char hostname[64];      /**< \brief DNS/DHCP hostname. */
00081     char email[64];         /**< \brief E-Mail address. */
```

```
00082     char smtp[64];          /**< \brief SMTP server address. */
00083     char pop3[64];          /**< \brief POP3 server address. */
00084     char pop3_login[64];    /**< \brief POP3 server username. */
00085     char pop3_passwd[64];   /**< \brief POP3 server password. */
00086     char proxy_host[64];    /**< \brief Proxy server address. */
00087     int proxy_port;         /**< \brief Proxy server port. */
00088     char ppp_login[64];     /**< \brief PPP Username. */
00089     char ppp_passwd[64];    /**< \brief PPP Password. */
00090     char driver[64];        /**< \brief Driver program filename (if any). */
00091 } netcfg_t;
00092
00093 /** \brief  Load network configuration from a file.
00094
00095     This function loads the network configuration that is stored in the given
00096     file to the network configuration structure passed in. This function will
00097     also handle files on the VMU with the VMU specific header attached without
00098     any extra work required.
00099
00100     \param  fn          The file to read the configuration from.
00101     \param  out         Buffer to store the parsed configuration.
00102     \return             0 on success, <0 on failure.
00103 */
00104 int netcfg_load_from(const char * fn, netcfg_t * out);
00105
00106 /** \brief  Load network configuration from the Dreamcast's flashrom.
00107
00108     This function loads the network configuration that is stored in flashrom of
00109     the Dreamcast, parsing it into a netcfg_t.
00110
00111     \param  out         Buffer to store the parsed configuration.
00112     \return             0 on success, <0 on failure.
00113     \note               This currently does not read the configuration stored by
00114                         the PlanetWeb browser at all.
00115 */
00116 int netcfg_load_flash(netcfg_t * out);
00117
00118 /** \brief  Load network configuration.
00119
00120     This function loads the network configuration data, searching in multiple
00121     locations to attempt to find a file with a stored configurtion. This
00122     function will attempt to read the configuration from each VMU first (from
00123     a file named net.cfg), then it will try the flashrom, followed by the
00124     current working directory, and lastly the root of the CD.
00125
00126     \param  out         Buffer to store the parsed configuraiton.
00127     \return             0 on success, <0 on failure.
00128 */
00129 int netcfg_load(netcfg_t * out);
00130
00131 /** \brief  Save network configuration to a file.
00132
00133     This function saves the network configuration to the specified file. This
00134     function will automatically prepend the appropriate header if it is saved
00135     to a VMU.
00136
00137     \param  fn          The file to save to.
00138     \param  cfg         The configuration to save.
00139     \return             0 on success, <0 on failure.
00140 */
00141 int netcfg_save_to(const char * fn, const netcfg_t * cfg);
00142
00143 /** \brief  Save network configuration to the first available VMU.
00144
00145     This function saves the network configuration to first VMU that it finds. It
00146     will not retry if the first VMU doesn't have enough space to hold the file.
00147
00148     \param  cfg         The configuration to save.
00149     \return             0 on success, <0 on failure.
00150 */
00151 int netcfg_save(const netcfg_t * cfg);
00152
00153 __END_DECLS
00154
00155 #endif  /* __KOS_NETCFG_H */
00156
```

9.13 addons/include/kos/pcx.h File Reference

Small PCX Loader.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Functions

- int pcx_load_flat (const char *fn, int *w_out, int *h_out, void *pic_out)

  Load a PCX file into a buffer as flat 15-bit BGR data.
- int pcx_load_palette (const char *fn, int *w_out, int *h_out, void *pic_out, void *pal_out)

  Load a PCX file into a buffer as paletted data.

9.13.1 Detailed Description

Small PCX Loader.

This module provides a few functions used for loading PCX files. These funcions were mainly for use on the GBA port of KallistiOS (which has been removed from the tree), although they can be used pretty much anywhere. That said, libpcx is generally more useful than these functions for use on the Dreamcast.

Author

Megan Potter

9.13.2 Function Documentation

pcx_load_flat()

```
int pcx_load_flat (
            const char * fn,
            int * w_out,
            int * h_out,
            void * pic_out )
```

Load a PCX file into a buffer as flat 15-bit BGR data.

This function loads the specified PCX file into the buffer provided, decoding the image to 15-bit BGR pixels. The width and height of the buffer should match that of the image itself.

Parameters

fn	The file to load.
w_out	Buffer to return the width of the image in.
h_out	Buffer to return the height of the image in.
pic_out	Buffer to store image data in.

Returns

0 on succes, < 0 on failure.

pcx_load_palette()

```
int pcx_load_palette (
            const char * fn,
            int * w_out,
            int * h_out,
            void * pic_out,
            void * pal_out )
```

Load a PCX file into a buffer as paletted data.

This function is similar to the pcx_load_flat() function, but instead of decoding the image into a flat buffer, it decodes the image into a paletted format. The image itself will be 8-bit paletted pixels, and the palette is in a 15-bit BGR format. The width and height of the buffer should match that of the image itself.

Parameters

fn	The file to load.
w_out	Buffer to return the width of the image in.
h_out	Buffer to return the height of the image in.
pic_out	Buffer to store image data in. This should be width * height bytes in size.
pal_out	Buffer to store the palette data in. This should be allocated to hold 256 uint16_t values.

Returns

0 on succes, < 0 on failure.

9.14 pcx.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/pcx.h
00004    Copyright (C) 2000-2001 Megan Potter
00005
00006 */
00007
00008 #ifndef __KOS_PCX_H
00009 #define __KOS_PCX_H
00010
00011 /** \file   kos/pcx.h
00012     \brief  Small PCX Loader.
00013
00014     This module provides a few functions used for loading PCX files. These
00015     funcions were mainly for use on the GBA port of KallistiOS (which has been
00016     removed from the tree), although they can be used pretty much anywhere.
00017     That said, libpcx is generally more useful than these functions for use on
00018     the Dreamcast.
00019
00020     \author Megan Potter
00021 */
```

```
00022
00023 #include <sys/cdefs.h>
00024 __BEGIN_DECLS
00025
00026 #include <arch/types.h>
00027
00028 /* These first two versions are mainly for use with the GBA, and they are
00029    defined in the "pcx_small" file. They can be used on any architecture of
00030    course. */
00031
00032 /** \brief  Load a PCX file into a buffer as flat 15-bit BGR data.
00033
00034     This function loads the specified PCX file into the buffer provided,
00035     decoding the image to 15-bit BGR pixels. The width and height of the buffer
00036     should match that of the image itself.
00037
00038     \param  fn          The file to load.
00039     \param  w_out       Buffer to return the width of the image in.
00040     \param  h_out       Buffer to return the height of the image in.
00041     \param  pic_out     Buffer to store image data in.
00042     \return             0 on succes, < 0 on failure.
00043 */
00044 int pcx_load_flat(const char *fn, int *w_out, int *h_out, void *pic_out);
00045
00046 /** \brief  Load a PCX file into a buffer as paletted data.
00047
00048     This function is similar to the pcx_load_flat() function, but instead of
00049     decoding the image into a flat buffer, it decodes the image into a paletted
00050     format. The image itself will be 8-bit paletted pixels, and the palette is
00051     in a 15-bit BGR format. The width and height of the buffer should match that
00052     of the image itself.
00053
00054     \param  fn          The file to load.
00055     \param  w_out       Buffer to return the width of the image in.
00056     \param  h_out       Buffer to return the height of the image in.
00057     \param  pic_out     Buffer to store image data in. This should be width *
00058                         height bytes in size.
00059     \param  pal_out     Buffer to store the palette data in. This should be
00060                         allocated to hold 256 uint16_t values.
00061     \return             0 on succes, < 0 on failure.
00062 */
00063 int pcx_load_palette(const char *fn, int *w_out, int *h_out, void *pic_out,
00064                      void *pal_out);
00065
00066 __END_DECLS
00067
00068 #endif  /* __KOS_PCX_H */
00069
```

9.15 addons/include/navi/flash.h File Reference

BIOS replacement flashrom support.

```
#include <arch/types.h>
```

Functions

- int nvflash_detect (void)

  Try to detect a compatible flashrom.
- int nvflash_erase_block (uint32 addr)

  Erase a single block of flashrom.
- int nvflash_write_block (uint32 addr, void *data, uint32 len)

  Write data to the flashrom.
- int nvflash_erase_all (void)

  Erase the whole flashrom.

9.15.1 Detailed Description

BIOS replacement flashrom support.

This file is involved with accessing an flashrom chip soldered in place to replace the BIOS ROM. Specifically, this driver is for accessing a STMicro M29W800B, but should also work with other chips like the M29W160B or equivalent AMD chips.

Author

Megan Potter

9.15.2 Function Documentation

nvflash_detect()

```
int nvflash_detect (
            void  )
```

Try to detect a compatible flashrom.

Returns

0 if a compatible flashrom is detected, <0 if the normal Dreamcast BIOS is detected.

nvflash_erase_all()

```
int nvflash_erase_all (
            void  )
```

Erase the whole flashrom.

Returns

0 on success, <0 on error.

nvflash_erase_block()

```
int nvflash_erase_block (
            uint32 addr )
```

Erase a single block of flashrom.

Parameters

addr	The block of the flashrom to erase.

Returns

0 on success, <0 on error.

nvflash_write_block()

```
int nvflash_write_block (
              uint32 addr,
              void * data,
              uint32 len )
```

Write data to the flashrom.

Parameters

addr	The block of the flashrom to write to.
data	The data to write.
len	The length of the data, in bytes.

Returns

0 on success, <0 on error.

9.16 flash.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    navi/flash.h
00004    Copyright (C) 2002 Megan Potter
00005
00006 */
00007
00008 /** \file   navi/flash.h
00009     \brief  BIOS replacement flashrom support.
00010
00011     This file is involved with accessing an flashrom chip soldered in place to
00012     replace the BIOS ROM. Specifically, this driver is for accessing a STMicro
00013     M29W800B, but should also work with other chips like the M29W160B or
00014     equivalent AMD chips.
00015
00016     \author Megan Potter
00017 */
00018
00019 #ifndef __NAVI_FLASH_H
00020 #define __NAVI_FLASH_H
00021
00022 #include <arch/types.h>
00023
00024 /** \brief  Try to detect a compatible flashrom.
00025     \return                 0 if a compatible flashrom is detected, <0 if the
```

```
00026                           normal Dreamcast BIOS is detected.
00027 */
00028 int nvflash_detect(void);
00029
00030 /** \brief  Erase a single block of flashrom.
00031     \param  addr            The block of the flashrom to erase.
00032     \return                 0 on success, <0 on error.
00033 */
00034 int nvflash_erase_block(uint32 addr);
00035
00036 /** \brief  Write data to the flashrom.
00037     \param  addr            The block of the flashrom to write to.
00038     \param  data            The data to write.
00039     \param  len             The length of the data, in bytes.
00040     \return                 0 on success, <0 on error.
00041 */
00042 int nvflash_write_block(uint32 addr, void * data, uint32 len);
00043
00044 /* Erase the whole flash chip */
00045 /** \brief  Erase the whole flashrom.
00046     \return                 0 on success, <0 on error.
00047 */
00048 int nvflash_erase_all(void);
00049
00050 #endif  /* __NAVI_FLASH_H */
```

9.17 addons/include/navi/ide.h File Reference

External G2 Bus-based IDE support.

```
#include <arch/types.h>
```

Functions

- int ide_read (uint32 linear, uint32 numsects, void *bufptr)

  Read sectors from the hard disk via PIO.
- int ide_write (uint32 linear, uint32 numsects, void *bufptr)

  Write sectors from the hard disk via PIO.
- uint32 ide_num_sectors (void)

  Retrieve the number of sectors from the hard disk.
- int ide_init (void)

  Initialize Navi IDE.
- void ide_shutdown (void)

  Shutdown Navi IDE.

9.17.1 Detailed Description

External G2 Bus-based IDE support.

This file is involved with accessing an IDE controller that is attached to the G2 Bus expansion port. Exact details of how to build such a device have been posted in various places around the Internet. This driver refers to the device built by Megan as a part of the Navi project.

Author

Megan Potter

9.17.2 Function Documentation

ide_init()

```
int ide_init (
            void  )
```

Initialize Navi IDE.

Returns

0 on success (no error conditions defined).

ide_num_sectors()

```
uint32 ide_num_sectors (
            void  )
```

Retrieve the number of sectors from the hard disk.

Returns

The total number of linear sectors.

ide_read()

```
int ide_read (
            uint32 linear,
            uint32 numsects,
            void * bufptr )
```

Read sectors from the hard disk via PIO.

Parameters

linear	The address to begin reading from.
numsects	The number of sectors to read.
bufptr	The buffer to read into.

Returns

0 on success, $<$0 on error.

ide_shutdown()

```
void ide_shutdown (
            void  )
```

Shutdown Navi IDE.

ide_write()

```
int ide_write (
            uint32 linear,
            uint32 numsects,
            void * bufptr )
```

Write sectors from the hard disk via PIO.

Parameters

linear	The address to begin writing to.
numsects	The number of sectors to write.
bufptr	The buffer to write out of.

Returns

0 on success, <0 on error.

9.18 ide.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    navi/ide.h
00004    Copyright (C) 2002 Megan Potter
00005
00006 */
00007
00008 /** \file   navi/ide.h
00009     \brief  External G2 Bus-based IDE support.
00010
00011     This file is involved with accessing an IDE controller that is attached to
00012     the G2 Bus expansion port. Exact details of how to build such a device have
00013     been posted in various places around the Internet. This driver refers to the
00014     device built by Megan as a part of the Navi project.
00015
00016     \author Megan Potter
00017 */
00018
00019 #ifndef __NAVI_IDE_H
00020 #define __NAVI_IDE_H
00021
00022 #include <arch/types.h>
00023
00024 /** \brief  Read sectors from the hard disk via PIO.
00025     \param  linear          The address to begin reading from.
00026     \param  numsects        The number of sectors to read.
00027     \param  bufptr          The buffer to read into.
00028     \return                 0 on success, <0 on error.
00029 */
```

```
00030 int ide_read(uint32 linear, uint32 numsects, void *bufptr);
00031
00032 /** \brief  Write sectors from the hard disk via PIO.
00033     \param  linear          The address to begin writing to.
00034     \param  numsects        The number of sectors to write.
00035     \param  bufptr          The buffer to write out of.
00036     \return                 0 on success, <0 on error.
00037 */
00038 int ide_write(uint32 linear, uint32 numsects, void *bufptr);
00039
00040 /** \brief  Retrieve the number of sectors from the hard disk.
00041     \returns                The total number of linear sectors.
00042 */
00043 uint32 ide_num_sectors(void);
00044
00045 /** \brief  Initialize Navi IDE.
00046     \return                 0 on success (no error conditions defined).
00047 */
00048 int ide_init(void);
00049
00050 /** \brief  Shutdown Navi IDE. */
00051 void ide_shutdown(void);
00052
00053 #endif  /* __NAVI_IDE_H */
```

9.19 addons/include/ppp/ppp.h File Reference

PPP interface for network communications.

```
#include <sys/cdefs.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/queue.h>
```

Data Structures

- struct ppp_device_t

  PPP device structure.
- struct ppp_protocol_t

  PPP Protocol structure.

Macros

- #define PPP_TX_END_OF_PKT 0x00000001

  End of packet flag.
- #define PPP_PROTO_ENTRY_INIT { NULL, NULL }

  Static initializer for protocol list entry.
- #define PPP_PHASE_DEAD 0x01

  Pre-connection.
- #define PPP_PHASE_ESTABLISH 0x02

  Establishing connection.
- #define PPP_PHASE_AUTHENTICATE 0x03

  Authentication to peer.
- #define PPP_PHASE_NETWORK 0x04

Established and working.

- #define PPP_PHASE_TERMINATE 0x05

  Tearing down the link.
- #define PPP_FLAG_AUTH_PAP 0x00000001

  PAP authentication.
- #define PPP_FLAG_AUTH_CHAP 0x00000002

  CHAP authentication.
- #define PPP_FLAG_PCOMP 0x00000004

  Protocol compression.
- #define PPP_FLAG_ACCOMP 0x00000008

  Addr/ctrl compression.
- #define PPP_FLAG_MAGIC_NUMBER 0x00000010

  Use magic numbers.
- #define PPP_FLAG_WANT_MRU 0x00000020

  Specify MRU.
- #define PPP_FLAG_NO_ACCM 0x00000040

  No ctl character map.

Functions

- int ppp_set_device (ppp_device_t *dev)

  Set the device used to do PPP communications.
- int ppp_set_login (const char *username, const char *password)

  Set the login credentials used to authenticate to the peer.
- int ppp_send (const uint8_t *data, size_t len, uint16_t proto)

  Send a packet on the PPP link.
- int ppp_add_protocol (ppp_protocol_t *hnd)

  Register a protocol with the PPP stack.
- int ppp_del_protocol (ppp_protocol_t *hnd)

  Unregister a protocol from the PPP stack.
- int ppp_lcp_send_proto_reject (uint16_t proto, const uint8_t *pkt, size_t len)

  Send a Protocol Reject packet on the link.
- uint32_t ppp_get_flags (void)

  Get the flags set for our side of the link.
- uint32_t ppp_get_peer_flags (void)

  Get the flags set for the peer's side of the link.
- void ppp_set_flags (uint32_t flags)

  Set the flags set for our side of the link.
- int ppp_connect (void)

  Establish a point-to-point link across a previously set-up device.
- int ppp_scif_init (int bps)

  Initialize the Dreamcast serial port for a PPP link.
- int ppp_modem_init (const char *number, int blind, int *conn_rate)

  Initialize the Dreamcast modem for a PPP link.
- int ppp_init (void)

  Initialize the PPP library.
- int ppp_shutdown (void)

  Shut down the PPP library.

9.19.1 Detailed Description

PPP interface for network communications.

This file defines the API provided by libppp to interact with the PPP stack. PPP is a network communication protocol used to establish a direct link between two peers. It is most commonly used as the data link layer protocol for dialup internet access, but can also be potentially used on broadband connections (PPP over Ethernet or PPPoE) or on a direct serial line to a computer.

The API presented by this library is designed to be extensible to whatever devices you might want to use it with, and was designed to integrate fairly simply into the rest of KOS' network stack.

Author

Lawrence Sebald

9.19.2 Macro Definition Documentation

PPP_PROTO_ENTRY_INIT

```
#define PPP_PROTO_ENTRY_INIT { NULL, NULL }
```

Static initializer for protocol list entry.

PPP_TX_END_OF_PKT

```
#define PPP_TX_END_OF_PKT 0x00000001
```

End of packet flag.

9.19.3 Function Documentation

ppp_add_protocol()

```
int ppp_add_protocol (
            ppp_protocol_t * hnd )
```

Register a protocol with the PPP stack.

This function adds a new protocol to the PPP stack, allowing the stack to communicate packets for the given protocol across the link. Generally, you should not have any reason to call this function, as the library includes a set of protocols to do normal communications.

Parameters

hnd	A protocol handler structure.

Returns

0 on success, <0 on failure.

ppp_connect()

```
int ppp_connect (
            void  )
```

Establish a point-to-point link across a previously set-up device.

This function establishes a point-to-point link to the peer across a device that was previously set up with ppp_set_device(). Before calling this function, the device must be ready to communicate with the peer. That is to say, any handshaking needed to establish the underlying hardware link must have already completed.

Returns

0 on success, <0 on failure.

Note

This function will block until the link is established.

ppp_del_protocol()

```
int ppp_del_protocol (
            ppp_protocol_t * hnd )
```

Unregister a protocol from the PPP stack.

This function removes protocol from the PPP stack. This should be done at shutdown time of any protocols added with ppp_add_protocol().

Parameters

hnd	A protocol handler structure.

Returns

0 on success, <0 on failure.

ppp_get_flags()

```
uint32_t ppp_get_flags (
            void  )
```

Get the flags set for our side of the link.

This function retrieves the connection flags set for our side of the PPP link. Before link establishment, this indicates the flags we would like to establish on the link and after establishment it represents the flags that were actually negotiated during link establishment.

Returns

Bitwise OR of PPP link configuration flags.

ppp_get_peer_flags()

```
uint32_t ppp_get_peer_flags (
            void  )
```

Get the flags set for the peer's side of the link.

This function retrieves the connection flags set for the other side of the PPP link. This value is only valid after link establishment.

Returns

Bitwise OR of PPP link configuration flags.

ppp_init()

```
int ppp_init (
            void  )
```

Initialize the PPP library.

This function initializes the PPP library, preparing internal structures for use and initializing the PPP protocols needed for normal IP communications.

Returns

0 on success, <0 on failure.

ppp_lcp_send_proto_reject()

```
int ppp_lcp_send_proto_reject (
            uint16_t proto,
            const uint8_t * pkt,
            size_t len )
```

Send a Protocol Reject packet on the link.

This function sends a LCP protocol reject packet on the link for the specified packet. Generally, you should not have to call this function, as the library will handle doing so internally.

Parameters

proto	The PPP protocol number of the invalid packet.
pkt	The packet itself.
len	The length of the packet, in bytes.

Returns

0 on success, <0 on failure.

ppp_modem_init()

```
int ppp_modem_init (
            const char * number,
            int blind,
            int * conn_rate )
```

Initialize the Dreamcast modem for a PPP link.

This function sets up the Dreamcast modem to act as a communications link for a point-to-point connection. This includes dialing the specified phone number and establishing the low-level link.

Parameters

number	The phone number to dial out.
blind	Non-zero to blind dial (don't wait for a dial tone).
conn_rate	Storage for the connection rate, in bits per second. Set to NULL if you do not need this value back from the function.

Return values

0	On success.
-1	If modem initialization fails.
-2	If not using blind dial and no dial tone is detected within 5 seconds of opening the line.
-3	If dialing the modem fails.
-4	If a connection is not established in 60 seconds after dialing.

ppp_scif_init()

```
int ppp_scif_init (
            int bps )
```

Initialize the Dreamcast serial port for a PPP link.

This function sets up the Dreamcast serial port to act as a communications link for a point-to-point connection. This can be used in conjunction with a coder's cable (or similar) to connect the Dreamcast to a PC and the internet if the target system is set up properly.

Parameters

bps	The speed to initialize the serial port at.

Returns

0 on success, <0 on failure.

ppp_send()

```
int ppp_send (
              const uint8_t * data,
              size_t len,
              uint16_t proto )
```

Send a packet on the PPP link.

This function sends a single packet to the peer on the PPP link. Generally, you should not use this function directly, but rather use the facilities provided by KOS' network stack.

Parameters

data	The packet to send.
len	The length of the packet, in bytes.
proto	The PPP protocol number for the packet.

Returns

0 on success, <0 on failure.

ppp_set_device()

```
int ppp_set_device (
              ppp_device_t * dev )
```

Set the device used to do PPP communications.

This function sets the device that further communications over a point-to-point link will take place over. The device need not be ready to communicate immediately upon calling this function.

Unless you are adding support for a new device, you will probably never have to call this function. For instance, if you want to use the Dreamcast serial port to establish a link, the ppp_scif_init() function will call this for you.

Parameters

dev	The device to use for communication.

Returns

0 on success, <0 on failure.

Note

Calling this function after establishing a PPP link will fail.

ppp_set_flags()

```
void ppp_set_flags (
            uint32_t flags )
```

Set the flags set for our side of the link.

This function sets the connection flags for our side of the PPP link.

ppp_set_login()

```
int ppp_set_login (
            const char * username,
            const char * password )
```

Set the login credentials used to authenticate to the peer.

This function sets the login credentials that will be used to authenticate to the peer, if the peer requests authentication. The specifics of how the authentication takes place depends on what options are configured when establishing the link.

Parameters

username	The username to authenticate as.
password	The password to use to authenticate.

Returns

0 on success, <0 on failure.

Note

Calling this function after establishing a PPP link will fail.

ppp_shutdown()

```
int ppp_shutdown (
            void  )
```

Shut down the PPP library.

This function cleans up the PPP library, shutting down any connections and deinitializing any protocols that have bene registered previously.

Returns

0 on success, <0 on failure.

9.20 ppp.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    ppp/ppp.h
00004    Copyright (C) 2014 Lawrence Sebald
00005 */
00006
00007 #ifndef __PPP_PPP_H
00008 #define __PPP_PPP_H
00009
00010 #include <sys/cdefs.h>
00011 __BEGIN_DECLS
00012
00013 #include <stdint.h>
00014 #include <sys/types.h>
00015 #include <sys/queue.h>
00016
00017 /** \file   ppp/ppp.h
00018     \brief  PPP interface for network communications.
00019
00020     This file defines the API provided by libppp to interact with the PPP stack.
00021     PPP is a network communication protocol used to establish a direct link
00022     between two peers. It is most commonly used as the data link layer protocol
00023     for dialup internet access, but can also be potentially used on broadband
00024     connections (PPP over Ethernet or PPPoE) or on a direct serial line to
00025     a computer.
00026
00027     The API presented by this library is designed to be extensible to whatever
00028     devices you might want to use it with, and was designed to integrate fairly
00029     simply into the rest of KOS' network stack.
00030
00031     \author Lawrence Sebald
00032 */
00033
00034 /** \brief  PPP device structure.
00035
00036     This structure defines a basic output device for PPP packets. This structure
00037     is largely modeled after netif_t from the main network stack, with a bit of
00038     functionality removed that is irrelevant for PPP.
00039
00040     Note that we only allow one device and one connection in this library.
00041
00042     \headerfile ppp/ppp.h
00043 */
00044 typedef struct ppp_device {
00045     /** \brief  Device name ("modem", "scif", etc). */
00046     const char *name;
00047
00048     /** \brief  Long description of the device. */
00049     const char *descr;
00050
00051     /** \brief  Unit index (starts at zero and counts upwards for multiple
00052                 network devices of the same type). */
00053     int index;
00054
00055     /** \brief  Device flags.
00056         The lowest 16 bits of this value are reserved for use by libppp. You are
00057         free to use the other 16 bits as you see fit in your driver. */
00058     uint32_t flags;
00059
00060     /** \brief  Private, device-specific data.
00061         This can be used for whatever the driver deems fit. The PPP code won't
```

```
00062         touch this data at all. Set to NULL if you don't need anything here. */
00063     void *privdata;
00064
00065     /** \brief  Attempt to detect the device.
00066         \param  self        The network device in question.
00067         \return             0 on success, <0 on failure.
00068     */
00069     int (*detect)(struct ppp_device *self);
00070
00071     /** \brief  Initialize the device.
00072         \param  self        The network device in question.
00073         \return             0 on success, <0 on failure.
00074     */
00075     int (*init)(struct ppp_device *self);
00076
00077     /** \brief  Shutdown the device.
00078         \param  self        The network device in question.
00079         \return             0 on success, <0 on failure.
00080     */
00081     int (*shutdown)(struct ppp_device *self);
00082
00083     /** \brief  Transmit data on the device.
00084
00085         This function will be called periodically to transmit data on the
00086         underlying device. The data passed in may not necessarily be a whole
00087         packet (check the flags to see what's being passed in).
00088
00089         \param  self        The network device in question.
00090         \param  data        The data to transmit.
00091         \param  len         The length of the data to transmit in bytes.
00092         \param  flags       Flags to describe what data is being sent in.
00093         \return             0 on success, <0 on failure.
00094     */
00095     int (*tx)(struct ppp_device *self, const uint8_t *data, size_t len,
00096               uint32_t flags);
00097
00098     /** \brief  Poll for queued receive data.
00099
00100         This function will be called periodically by a thread to check the
00101         device for any new incoming data.
00102
00103         \param  self        The network device in question.
00104         \return             A pointer to the received data on success. NULL on
00105                             failure or if no data is waiting.
00106     */
00107     const uint8_t *(*rx)(struct ppp_device *self, ssize_t *out_len);
00108 } ppp_device_t;
00109
00110 /** \brief  End of packet flag. */
00111 #define PPP_TX_END_OF_PKT    0x00000001
00112
00113 /** \brief  PPP Protocol structure.
00114
00115     Each protocol that the PPP library can handle must have one of these
00116     registered. All protocols should be registered BEFORE attempting to actually
00117     establish a PPP session to ensure that each protocol can be used in the
00118     setup of the connection as needed.
00119
00120     \headerfile ppp/ppp.h
00121 */
00122 typedef struct ppp_proto {
00123     /** \brief  Protocol list entry (not a function!). */
00124     TAILQ_ENTRY(ppp_proto) entry;
00125
00126     /** \brief  Protocol name ("lcp", "pap", etc). */
00127     const char *name;
00128
00129     /** \brief  Protocol code. */
00130     uint16_t code;
00131
00132     /** \brief  Private data (if any). */
00133     void *privdata;
00134
00135     /** \brief  Initialization function.
00136
00137         \param  self        The protocol structure for this protocol.
00138         \return             0 on success, <0 on failure.
00139         \note               Set to NULL if this is not needed in the protocol.
00140     */
00141     int (*init)(struct ppp_proto *self);
00142
```

```
00143     /** \brief  Shutdown function.
00144
00145         This function should perform any protocol-specific shutdown actions and
00146         unregister the protocol from the PPP protocol list.
00147
00148         \param  self        The protocol structure for this protocol.
00149         \return             0 on success, <0 on failure.
00150     */
00151     int (*shutdown)(struct ppp_proto *self);
00152
00153     /** \brief  Protocol packet input function.
00154
00155         This function will be called for each packet delivered to the specified
00156         protocol.
00157
00158         \param  self        The protocol structure for this protocol.
00159         \param  pkt         The packet being delivered.
00160         \param  len         The length of the packet in bytes.
00161         \return             0 on success, <0 on failure.
00162     */
00163     int (*input)(struct ppp_proto *self, const uint8_t *buf, size_t len);
00164
00165     /** \brief  Notify the protocol of a PPP phase change.
00166
00167         This function will be called by the PPP automaton any time that a phase
00168         change is initiated. This is often used for starting up a protocol when
00169         appropriate to do so (for instance, LCP uses this to begin negotiating
00170         configuration options with the peer when the establish phase is entered
00171         by the automaton).
00172
00173         \param  self        The protocol structure for this protocol.
00174         \param  oldp        The old phase (the one the automaton is leaving).
00175         \param  newp        The new phase.
00176         \see                ppp_phases
00177     */
00178     void (*enter_phase)(struct ppp_proto *self, int oldp, int newp);
00179
00180     /** \brief  Check timeouts for resending packets.
00181
00182         This function will be called periodically to allow the protocol to check
00183         any resend timers that it might have responsibility for.
00184
00185         \param  self        The protocol structure for this protocol.
00186         \param  tm          The current system time for checking timeouts
00187                             against (in milliseconds since system startup).
00188     */
00189     void (*check_timeouts)(struct ppp_proto *self, uint64_t tm);
00190 } ppp_protocol_t;
00191
00192 /** \brief  Static initializer for protocol list entry. */
00193 #define PPP_PROTO_ENTRY_INIT { NULL, NULL }
00194
00195 /** \defgroup ppp_phases                PPP automaton phases
00196
00197     This list defines the phases of the PPP automaton, as described in Section
00198     3.2 of RFC 1661.
00199
00200     @{
00201 */
00202 #define PPP_PHASE_DEAD          0x01    /**< \brief Pre-connection. */
00203 #define PPP_PHASE_ESTABLISH     0x02    /**< \brief Establishing connection. */
00204 #define PPP_PHASE_AUTHENTICATE  0x03    /**< \brief Authentication to peer. */
00205 #define PPP_PHASE_NETWORK       0x04    /**< \brief Established and working. */
00206 #define PPP_PHASE_TERMINATE     0x05    /**< \brief Tearing down the link. */
00207 /** @} */
00208
00209 /** \brief  Set the device used to do PPP communications.
00210
00211     This function sets the device that further communications over a
00212     point-to-point link will take place over. The device need not be ready to
00213     communicate immediately upon calling this function.
00214
00215     Unless you are adding support for a new device, you will probably never have
00216     to call this function. For instance, if you want to use the Dreamcast serial
00217     port to establish a link, the ppp_scif_init() function will call this for
00218     you.
00219
00220     \param  dev         The device to use for communication.
00221     \return             0 on success, <0 on failure.
00222
00223     \note               Calling this function after establishing a PPP link will
```

```
00224                       fail.
00225 */
00226 int ppp_set_device(ppp_device_t *dev);
00227
00228 /** \brief  Set the login credentials used to authenticate to the peer.
00229
00230     This function sets the login credentials that will be used to authenticate
00231     to the peer, if the peer requests authentication. The specifics of how the
00232     authentication takes place depends on what options are configured when
00233     establishing the link.
00234
00235     \param  username    The username to authenticate as.
00236     \param  password    The password to use to authenticate.
00237     \return             0 on success, <0 on failure.
00238
00239     \note               Calling this function after establishing a PPP link will
00240                         fail.
00241 */
00242 int ppp_set_login(const char *username, const char *password);
00243
00244 /** \brief  Send a packet on the PPP link.
00245
00246     This function sends a single packet to the peer on the PPP link. Generally,
00247     you should not use this function directly, but rather use the facilities
00248     provided by KOS' network stack.
00249
00250     \param  data        The packet to send.
00251     \param  len         The length of the packet, in bytes.
00252     \param  proto       The PPP protocol number for the packet.
00253     \return             0 on success, <0 on failure.
00254 */
00255 int ppp_send(const uint8_t *data, size_t len, uint16_t proto);
00256
00257 /** \brief  Register a protocol with the PPP stack.
00258
00259     This function adds a new protocol to the PPP stack, allowing the stack to
00260     communicate packets for the given protocol across the link. Generally, you
00261     should not have any reason to call this function, as the library includes
00262     a set of protocols to do normal communications.
00263
00264     \param  hnd         A protocol handler structure.
00265     \return             0 on success, <0 on failure.
00266 */
00267 int ppp_add_protocol(ppp_protocol_t *hnd);
00268
00269 /** \brief  Unregister a protocol from the PPP stack.
00270
00271     This function removes protocol from the PPP stack. This should be done at
00272     shutdown time of any protocols added with ppp_add_protocol().
00273
00274     \param  hnd         A protocol handler structure.
00275     \return             0 on success, <0 on failure.
00276 */
00277 int ppp_del_protocol(ppp_protocol_t *hnd);
00278
00279 /** \brief  Send a Protocol Reject packet on the link.
00280
00281     This function sends a LCP protocol reject packet on the link for the
00282     specified packet. Generally, you should not have to call this function, as
00283     the library will handle doing so internally.
00284
00285     \param  proto       The PPP protocol number of the invalid packet.
00286     \param  pkt         The packet itself.
00287     \param  len         The length of the packet, in bytes.
00288     \return             0 on success, <0 on failure.
00289 */
00290 int ppp_lcp_send_proto_reject(uint16_t proto, const uint8_t *pkt, size_t len);
00291
00292 /** \defgroup ppp_flags                 PPP link configuration flags
00293
00294     This list defines the flags we can negotiate during link establishment.
00295
00296     @{
00297 */
00298 #define PPP_FLAG_AUTH_PAP       0x00000001  /**< \brief PAP authentication */
00299 #define PPP_FLAG_AUTH_CHAP      0x00000002  /**< \brief CHAP authentication */
00300 #define PPP_FLAG_PCOMP          0x00000004  /**< \brief Protocol compression */
00301 #define PPP_FLAG_ACCOMP         0x00000008  /**< \brief Addr/ctrl compression */
00302 #define PPP_FLAG_MAGIC_NUMBER   0x00000010  /**< \brief Use magic numbers */
00303 #define PPP_FLAG_WANT_MRU       0x00000020  /**< \brief Specify MRU */
00304 #define PPP_FLAG_NO_ACCM        0x00000040  /**< \brief No ctl character map */
```

```
00305 /** @} */
00306
00307 /** \brief  Get the flags set for our side of the link.
00308
00309     This function retrieves the connection flags set for our side of the PPP
00310     link. Before link establishment, this indicates the flags we would like to
00311     establish on the link and after establishment it represents the flags that
00312     were actually negotiated during link establishment.
00313
00314     \return             Bitwise OR of \ref ppp_flags.
00315 */
00316 uint32_t ppp_get_flags(void);
00317
00318 /** \brief  Get the flags set for the peer's side of the link.
00319
00320     This function retrieves the connection flags set for the other side of the
00321     PPP link. This value is only valid after link establishment.
00322
00323     \return             Bitwise OR of \ref ppp_flags.
00324 */
00325 uint32_t ppp_get_peer_flags(void);
00326
00327 /** \brief  Set the flags set for our side of the link.
00328
00329     This function sets the connection flags for our side of the PPP link.
00330 */
00331 void ppp_set_flags(uint32_t flags);
00332
00333 /** \brief  Establish a point-to-point link across a previously set-up device.
00334
00335     This function establishes a point-to-point link to the peer across a device
00336     that was previously set up with ppp_set_device(). Before calling this
00337     function, the device must be ready to communicate with the peer. That is to
00338     say, any handshaking needed to establish the underlying hardware link must
00339     have already completed.
00340
00341     \return             0 on success, <0 on failure.
00342
00343     \note               This function will block until the link is established.
00344 */
00345 int ppp_connect(void);
00346
00347 /** \brief  Initialize the Dreamcast serial port for a PPP link.
00348
00349     This function sets up the Dreamcast serial port to act as a communications
00350     link for a point-to-point connection. This can be used in conjunction with a
00351     coder's cable (or similar) to connect the Dreamcast to a PC and the internet
00352     if the target system is set up properly.
00353
00354     \param  bps         The speed to initialize the serial port at.
00355     \return             0 on success, <0 on failure.
00356 */
00357 int ppp_scif_init(int bps);
00358
00359 /** \brief  Initialize the Dreamcast modem for a PPP link.
00360
00361     This function sets up the Dreamcast modem to act as a communications
00362     link for a point-to-point connection. This includes dialing the specified
00363     phone number and establishing the low-level link.
00364
00365     \param  number      The phone number to dial out.
00366     \param  blind       Non-zero to blind dial (don't wait for a dial tone).
00367     \param  conn_rate   Storage for the connection rate, in bits per second. Set
00368                         to NULL if you do not need this value back from the
00369                         function.
00370     \retval 0           On success.
00371     \retval -1          If modem initialization fails.
00372     \retval -2          If not using blind dial and no dial tone is detected
00373                         within 5 seconds of opening the line.
00374     \retval -3          If dialing the modem fails.
00375     \retval -4          If a connection is not established in 60 seconds after
00376                         dialing.
00377 */
00378 int ppp_modem_init(const char *number, int blind, int *conn_rate);
00379
00380 /** \brief  Initialize the PPP library.
00381
00382     This function initializes the PPP library, preparing internal structures for
00383     use and initializing the PPP protocols needed for normal IP communications.
00384
00385     \return             0 on success, <0 on failure.
```

```
00386 */
00387 int ppp_init(void);
00388
00389 /** \brief  Shut down the PPP library.
00390
00391     This function cleans up the PPP library, shutting down any connections and
00392     deinitializing any protocols that have bene registered previously.
00393
00394     \return             0 on success, <0 on failure.
00395 */
00396 int ppp_shutdown(void);
00397
00398 __END_DECLS
00399 #endif /* !__PPP_PPP_H */
```

9.21 doc/pages/threading.dox File Reference

9.22 include/arpa/inet.h File Reference

Definitions for internet operations.

```
#include <sys/cdefs.h>
#include <netinet/in.h>
#include <inttypes.h>
```

Functions

- uint32_t htonl (uint32_t value)

  Convert a 32-bit value from host byte order to network byte order.
- uint32_t ntohl (uint32_t value)

  Convert a 32-bit value from network byte order to host byte order.
- uint16_t htons (uint16_t value)

  Convert a 16-bit value from host byte order to network byte order.
- uint16_t ntohs (uint16_t value)

  Convert a 16-bit value from network byte order to host byte order.
- in_addr_t inet_addr (const char ∗cp)

  Convert a string representation of an IPv4 address to an in_addr_t.
- int inet_aton (const char ∗cp, struct in_addr ∗pin)

  Convert a string representation of an IPv4 address to a struct in_addr.
- int inet_pton (int af, const char ∗src, void ∗dst)

  Convert a string representation of an IP address to its binary representation.
- const char ∗ inet_ntop (int af, const void ∗src, char ∗dst, socklen_t size)

  Convert a binary representation of an IP address to a string.
- char ∗ inet_ntoa (struct in_addr addr)

  Convert a binary representation of an IPv4 address to a string.

9.22.1 Detailed Description

Definitions for internet operations.

This file contains the standard definitions (as directed by the POSIX 2008 standard) for several internet-related functions.

Author

Lawrence Sebald

9.22.2 Function Documentation

htonl()

```
uint32_t htonl (
            uint32_t value )
```

Convert a 32-bit value from host byte order to network byte order.

Parameters

value	The value to convert.

Returns

value converted to network byte order.

htons()

```
uint16_t htons (
            uint16_t value )
```

Convert a 16-bit value from host byte order to network byte order.

Parameters

value	The value to convert.

Returns

value converted to network byte order.

inet_addr()

```
in_addr_t inet_addr (
```

```
            const char * cp )
```

Convert a string representation of an IPv4 address to an in_addr_t.

This function converts a "dotted-decimal" string representation of an IPv4 address to an in_addr_t for use in a struct in_addr. This function supports all POSIX-required formats for the representation of the address.

Parameters

cp	A string representation of an IPv4 address.

Returns

The binary representation of the requested IPv4 address. (in_addr_t)(-1) is returned on error.

inet_aton()

```
int inet_aton (
            const char * cp,
            struct in_addr * pin )
```

Convert a string representation of an IPv4 address to a struct in_addr.

This function, much like inet_addr, converts a string representation of an IPv4 address to a binary representation. This function, however, is non-standard (but seems to appear a lot of places). This function is a little nicer to work with than inet_addr simply because of the fact that the error return from inet_addr happens to actually correspond to a real IPv4 address (255.255.255.255). This version actually distinguishes between that address and invalid addresses.

Parameters

cp	A string representation of an IPv4 address.
pin	The destination for the conversion.

Return values

0	An invalid IPv4 address was given.
1	Upon successful conversion.

inet_ntoa()

```
char * inet_ntoa (
            struct in_addr addr )
```

Convert a binary representation of an IPv4 address to a string.

This function does the exact opposite of the inet_addr function, converting a binary form of an address to a string. This function, unlike inet_ntop is non-reentrant (not thread-safe), and will always only support IPv4 addresses. It is suggested to use inet_ntop in any new code.

Parameters

addr	The address to convert.

Returns

A string representation of addr (in dotted-decimal form).

inet_ntop()

```
const char * inet_ntop (
             int af,
             const void * src,
             char * dst,
             socklen_t size )
```

Convert a binary representation of an IP address to a string.

This function does the exact oposite of the inet_pton function, converting a binary form of an address to a string. This function, unlike inet_ntoa, is reentrant, and is the function that you should generally use if you need to convert a binary representation of an IP address to a string.

Parameters

af	The address family that src is in. The only supported values are AF_INET and AF_INET6.
src	A binary representation of an IP address.
dst	Storage for the resulting string. This string should be at least 16-bytes long for IPv4, and 46 bytes for IPv6.
size	The length of dst.

Return values

NULL	Upon failed conversion.
dst	Upon successful conversion.

Error Conditions:

EAFNOSUPPORT - the specified address family is unsupported
ENOSPC - the size given is insufficient

inet_pton()

```
int inet_pton (
             int af,
```

```
            const char * src,
            void * dst )
```

Convert a string representation of an IP address to its binary representation.

This function, like inet_addr, converts a string representation of an IP address to its binary representation. This function, unlike inet_aton, is actually standard (in POSIX 2008), and operates very similarly. The only differences between this function and inet_aton are that this function does not support hexadecimal or octal representations and that this function has the ability to support IPv6. This is the function that you should actually use to convert addresses from strings to binary in new code, rather than inet_addr or inet_aton.

Parameters

af	The address family that src is an address in. The only supported values are AF_INET and AF_INET6.
src	A string representation of the address.
dst	Storage for the result. For AF_INET, this must be at least 32-bits in size (the function treats it as a struct in_addr). For AF_INET6, this must be at least 128-bits in size (the function treats it as a struct in6_addr).

Return values

-1	af is unsupported.
0	An invalid address was given.
1	Upon successful conversion.

Error Conditions:

EAFNOSUPPORT - the specified address family is unsupported

ntohl()

```
uint32_t ntohl (
            uint32_t value )
```

Convert a 32-bit value from network byte order to host byte order.

Parameters

value	The value to convert.

Returns

value converted to host byte order.

ntohs()

```
uint16_t ntohs (
            uint16_t value )
```

Convert a 16-bit value from network byte order to host byte order.

Parameters

value	The value to convert.

Returns

value converted to host byte order.

9.23 inet.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arpa/inet.h
00004    Copyright (C) 2006, 2007, 2010 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   arpa/inet.h
00009     \brief  Definitions for internet operations.
00010
00011     This file contains the standard definitions (as directed by the POSIX 2008
00012     standard) for several internet-related functions.
00013
00014     \author Lawrence Sebald
00015 */
00016
00017 #ifndef __ARPA_INET_H
00018 #define __ARPA_INET_H
00019
00020 #include <sys/cdefs.h>
00021
00022 __BEGIN_DECLS
00023
00024 /* Bring in <netinet/in.h> to get the in_port_t, in_addr_t, and struct in_addr
00025    types. Bring in <inttypes.h> for uint32_t and uint16_t. IEEE Std 1003.1-2008
00026    specifically says that <arpa/inet.h> can make all the symbols from these
00027    headers visible. */
00028 #include <netinet/in.h>
00029 #include <inttypes.h>
00030
00031 /** \brief  Convert a 32-bit value from host byte order to network byte order.
00032     \param  value           The value to convert.
00033     \return                 value converted to network byte order.
00034 */
00035 uint32_t htonl(uint32_t value);
00036
00037 /** \brief  Convert a 32-bit value from network byte order to host byte order.
00038     \param  value           The value to convert.
00039     \return                 value converted to host byte order.
00040 */
00041 uint32_t ntohl(uint32_t value);
00042
00043 /** \brief  Convert a 16-bit value from host byte order to network byte order.
00044     \param  value           The value to convert.
00045     \return                 value converted to network byte order.
00046 */
00047 uint16_t htons(uint16_t value);
00048
00049 /** \brief  Convert a 16-bit value from network byte order to host byte order.
00050     \param  value           The value to convert.
00051     \return                 value converted to host byte order.
00052 */
00053 uint16_t ntohs(uint16_t value);
00054
00055 /** \brief  Convert a string representation of an IPv4 address to an in_addr_t.
00056
00057     This function converts a "dotted-decimal" string representation of an IPv4
```

```
00058     address to an in_addr_t for use in a struct in_addr. This function supports
00059     all POSIX-required formats for the representation of the address.
00060
00061     \param  cp              A string representation of an IPv4 address.
00062     \return                 The binary representation of the requested IPv4
00063                             address. (in_addr_t)(-1) is returned on error.
00064 */
00065 in_addr_t inet_addr(const char *cp);
00066
00067 /** \brief  Convert a string representation of an IPv4 address to a struct
00068             in_addr.
00069
00070     This function, much like inet_addr, converts a string representation of an
00071     IPv4 address to a binary representation. This function, however, is
00072     non-standard (but seems to appear a lot of places). This function is a
00073     little nicer to work with than inet_addr simply because of the fact that the
00074     error return from inet_addr happens to actually correspond to a real IPv4
00075     address (255.255.255.255). This version actually distinguishes between that
00076     address and invalid addresses.
00077
00078     \param  cp              A string representation of an IPv4 address.
00079     \param  pin             The destination for the conversion.
00080     \retval 0               An invalid IPv4 address was given.
00081     \retval 1               Upon successful conversion.
00082 */
00083 int inet_aton(const char *cp, struct in_addr *pin);
00084
00085 /** \brief  Convert a string representation of an IP address to its binary
00086             representation.
00087
00088     This function, like inet_addr, converts a string representation of an IP
00089     address to its binary representation. This function, unlike inet_aton, is
00090     actually standard (in POSIX 2008), and operates very similarly. The only
00091     differences between this function and inet_aton are that this function does
00092     not support hexadecimal or octal representations and that this function has
00093     the ability to support IPv6. This is the function that you should actually
00094     use to convert addresses from strings to binary in new code, rather than
00095     inet_addr or inet_aton.
00096
00097     \param  af              The address family that src is an address in. The
00098                             only supported values are AF_INET and AF_INET6.
00099     \param  src             A string representation of the address.
00100     \param  dst             Storage for the result. For AF_INET, this must be at
00101                             least 32-bits in size (the function treats it as a
00102                             struct in_addr). For AF_INET6, this must be at least
00103                             128-bits in size (the function treats it as a struct
00104                             in6_addr).
00105     \retval -1              af is unsupported.
00106     \retval 0               An invalid address was given.
00107     \retval 1               Upon successful conversion.
00108
00109     \par    Error Conditions:
00110     \em     EAFNOSUPPORT - the specified address family is unsupported
00111 */
00112 int inet_pton(int af, const char *src, void *dst);
00113
00114 /** \brief  Convert a binary representation of an IP address to a string.
00115
00116     This function does the exact oposite of the inet_pton function, converting
00117     a binary form of an address to a string. This function, unlike inet_ntoa, is
00118     reentrant, and is the function that you should generally use if you need to
00119     convert a binary representation of an IP address to a string.
00120
00121     \param  af              The address family that src is in. The only
00122                             supported values are AF_INET and AF_INET6.
00123     \param  src             A binary representation of an IP address.
00124     \param  dst             Storage for the resulting string. This string should
00125                             be at least 16-bytes long for IPv4, and 46 bytes for
00126                             IPv6.
00127     \param  size            The length of dst.
00128     \retval NULL            Upon failed conversion.
00129     \retval dst             Upon successful conversion.
00130
00131     \par    Error Conditions:
00132     \em     EAFNOSUPPORT - the specified address family is unsupported \n
00133     \em     ENOSPC - the size given is insufficient
00134 */
00135 const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);
00136
00137 /** \brief  Convert a binary representation of an IPv4 address to a string.
00138
```

```
00139     This function does the exact opposite of the inet_addr function, converting
00140     a binary form of an address to a string. This function, unlike inet_ntop
00141     is non-reentrant (not thread-safe), and will always only support IPv4
00142     addresses. It is suggested to use inet_ntop in any new code.
00143
00144     \param  addr            The address to convert.
00145     \return                 A string representation of addr (in dotted-decimal
00146                             form).
00147 */
00148 char *inet_ntoa(struct in_addr addr);
00149
00150 __END_DECLS
00151
00152 #endif /* __ARPA_INET_H */
```

9.24 include/assert.h File Reference

Standard C Assertions.

```
#include <kos/cdefs.h>
```

Macros

- #define _assert(e) assert(e)
- #define assert(e) ((e) ? (void)0 : __assert(__FILE__, __LINE__, #e, NULL, __ASSERT_FUNC))

  Standard C assertion macro.
- #define assert_msg(e, m) ((e) ? (void)0 : __assert(__FILE__, __LINE__, #e, m, __ASSERT_FUNC))

  assert() with a custom message.

Typedefs

- typedef void(∗ assert_handler_t) (const char ∗file, int line, const char ∗expr, const char ∗msg, const char ∗func)

  Assertion handler type.

Functions

- assert_handler_t assert_set_handler (assert_handler_t hnd)

  Set an assertion handler to call on a failed assertion.

9.24.1 Detailed Description

Standard C Assertions.

This file contains the standard C assertions to raise an assertion or to change the assertion handler.

Author

Megan Potter

9.24.2 Macro Definition Documentation

_assert

```
#define _assert(
              e ) assert(e)
```

assert

```
#define assert(
              e ) ((e) ?  (void)0 :  __assert(__FILE__, __LINE__, #e, NULL, __ASSERT_FUNC))
```

Standard C assertion macro.

This macro does a standard C assertion, wherein the expression is evaluated, and if false, the program is ultimately aborted using abort(). If the expression evaluates to true, the macro does nothing (other than any side effects of evaluating the expression).

Parameters

e	A value or expression to be evaluated as true or false.

assert_msg

```
#define assert_msg(
              e,
              m ) ((e) ?  (void)0 :  __assert(__FILE__, __LINE__, #e, m, __ASSERT_FUNC))
```

assert() with a custom message.

This macro acts the same as the assert() macro, but allows you to specify a custom message to be printed out if the assertion fails.

Parameters

e	A value or expression to be evaluated as true or false.
m	A message (const char *).

9.24.3 Typedef Documentation

assert_handler_t

```
typedef void(* assert_handler_t) (const char *file, int line, const char *expr, const char *msg,
const char *func)
```

Assertion handler type.

The user can provide their own assertion handler with this type. If none is provided, a default is used which ultimately prints out the location of the failed assertion and calls abort().

Parameters

file	The filename where the assertion happened.
line	The line number where the assertion happened.
expr	The expression that raised the assertion.
msg	A custom message for why the assertion happened.
func	The function name from which the assertion happened.

See also

assert_set_handler

9.24.4 Function Documentation

assert_set_handler()

```
assert_handler_t assert_set_handler (
            assert_handler_t hnd )
```

Set an assertion handler to call on a failed assertion.

The default assertion handler simply will print a message and call abort(). NULL is a valid value and will cause nothing to happen on an assert.

Returns

The old assertion handler so it may be restored later if appropriate.

See also

assert_handler_t

9.25 assert.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    assert.h
00004    Copyright (C)2002,2004 Megan Potter
00005
00006 */
00007
00008 #ifndef __ASSERT_H
00009 #define __ASSERT_H
00010
00011 #include <kos/cdefs.h>
```

```
00012 __BEGIN_DECLS
00013
00014 /**
00015     \file   assert.h
00016     \brief  Standard C Assertions
00017
00018     This file contains the standard C assertions to raise an assertion or to
00019     change the assertion handler.
00020
00021     \author Megan Potter
00022 */
00023
00024 /* This is nice and simple, modeled after the BSD one like most of KOS;
00025    the addition here is assert_msg(), which allows you to provide an
00026    error message. */
00027 #define _assert(e) assert(e)
00028
00029 #ifdef NDEBUG
00030 #   define assert(e) ((void)0)
00031 #   define assert_msg(e, m) ((void)0)
00032 #else
00033
00034 /* This bit of magic borrowed from Newlib's assert.h... */
00035 /* \cond */
00036 #ifndef __ASSERT_FUNC
00037 #if defined(__cplusplus)
00038 #   define __ASSERT_FUNC __PRETTY_FUNCTION__
00039 #elif __STDC_VERSION__ >= 199901L
00040 #   define __ASSERT_FUNC __func__
00041 #elif __GNUC__ >= 2
00042 #   define __ASSERT_FUNC __FUNCTION__
00043 #else
00044 #   define __ASSERT_FUNC ((char *)0)
00045 #endif
00046 #endif
00047 /* \endcond */
00048
00049 /** \brief  Standard C assertion macro.
00050
00051     This macro does a standard C assertion, wherein the expression is evaluated,
00052     and if false, the program is ultimately aborted using abort(). If the
00053     expression evaluates to true, the macro does nothing (other than any side
00054     effects of evaluating the expression).
00055
00056     \param  e               A value or expression to be evaluated as true or
00057                             false.
00058 */
00059 #   define assert(e)        ((e) ? (void)0 : __assert(__FILE__, __LINE__, #e, NULL, __ASSERT_FUNC))
00060
00061 /** \brief  assert() with a custom message.
00062
00063     This macro acts the same as the assert() macro, but allows you to specify a
00064     custom message to be printed out if the assertion fails.
00065
00066     \param  e               A value or expression to be evaluated as true or
00067                             false.
00068     \param  m               A message (const char *).
00069 */
00070 #   define assert_msg(e, m) ((e) ? (void)0 : __assert(__FILE__, __LINE__, #e, m, __ASSERT_FUNC))
00071 #endif
00072
00073 /* \cond */
00074 /* Defined in assert.c */
00075 void __assert(const char *file, int line, const char *expr,
00076               const char *msg, const char *func);
00077 /* \endcond */
00078
00079 /** \brief  Assertion handler type.
00080
00081     The user can provide their own assertion handler with this type. If none is
00082     provided, a default is used which ultimately prints out the location of the
00083     failed assertion and calls abort().
00084
00085     \param  file            The filename where the assertion happened.
00086     \param  line            The line number where the assertion happened.
00087     \param  expr            The expression that raised the assertion.
00088     \param  msg             A custom message for why the assertion happened.
00089     \param  func            The function name from which the assertion happened.
00090
00091     \see    assert_set_handler
00092 */
```

```
00093 typedef void (*assert_handler_t)(const char * file, int line, const char * expr,
00094                                   const char * msg, const char * func);
00095
00096 /** \brief  Set an assertion handler to call on a failed assertion.
00097
00098     The default assertion handler simply will print a message and call abort().
00099     NULL is a valid value and will cause nothing to happen on an assert.
00100
00101     \return                 The old assertion handler so it may be restored
00102                             later if appropriate.
00103
00104     \see    assert_handler_t
00105 */
00106 assert_handler_t assert_set_handler(assert_handler_t hnd);
00107
00108 __END_DECLS
00109
00110 #endif  /* __ASSERT_H */
```

9.26 include/kos.h File Reference

Include everything KOS has to offer!

```
#include <kos/cdefs.h>
#include <ctype.h>
#include <malloc.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <kos/fs.h>
#include <kos/fs_romdisk.h>
#include <kos/fs_ramdisk.h>
#include <kos/fs_dev.h>
#include <kos/fs_pty.h>
#include <kos/limits.h>
#include <kos/thread.h>
#include <kos/sem.h>
#include <kos/rwsem.h>
#include <kos/once.h>
#include <kos/tls.h>
#include <kos/mutex.h>
#include <kos/cond.h>
#include <kos/genwait.h>
#include <kos/library.h>
#include <kos/net.h>
#include <kos/nmmgr.h>
#include <kos/exports.h>
#include <kos/dbgio.h>
#include <kos/blockdev.h>
#include <kos/dbglog.h>
#include <kos/elf.h>
#include <kos/fs_socket.h>
#include <kos/init.h>
#include <arch/arch.h>
#include <arch/cache.h>
#include <arch/irq.h>
#include <arch/spinlock.h>
#include <arch/timer.h>
```

```
#include <arch/wdt.h>
#include <arch/types.h>
#include <arch/exec.h>
#include <arch/stack.h>
#include <arch/byteorder.h>
#include <arch/rtc.h>
#include <arch/gdb.h>
#include <arch/mmu.h>
#include <arch/memory.h>
#include <dc/asic.h>
#include <dc/biosfont.h>
#include <dc/cdrom.h>
#include <dc/fb_console.h>
#include <dc/flashrom.h>
#include <dc/fmath.h>
#include <dc/fs_dcload.h>
#include <dc/fs_dclsocket.h>
#include <dc/fs_iso9660.h>
#include <dc/fs_vmu.h>
#include <dc/g1ata.h>
#include <dc/g2bus.h>
#include <dc/maple.h>
#include <dc/maple/controller.h>
#include <dc/maple/dreameye.h>
#include <dc/maple/keyboard.h>
#include <dc/maple/mouse.h>
#include <dc/maple/purupuru.h>
#include <dc/maple/sip.h>
#include <dc/maple/vmu.h>
#include <dc/matrix3d.h>
#include <dc/matrix.h>
#include <dc/modem/modem.h>
#include <dc/net/broadband_adapter.h>
#include <dc/net/lan_adapter.h>
#include <dc/pvr.h>
#include <dc/scif.h>
#include <dc/sd.h>
#include <dc/sound/stream.h>
#include <dc/sound/sfxmgr.h>
#include <dc/spu.h>
#include <dc/sq.h>
#include <dc/ubc.h>
#include <dc/vblank.h>
#include <dc/vec3f.h>
#include <dc/video.h>
#include <dc/vmu_pkg.h>
#include <dc/vmufs.h>
```

9.26.1 Detailed Description

Include everything KOS has to offer!

This file includes pretty much every KOS-related header file, so you don't have to figure out what you actually need. The ultimate for the truly lazy!

You may want to include individual header files yourself if you need more fine-grained control, as may be more appropriate for some projects.

Author

Megan Potter

9.27 kos.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos.h
00004    Copyright (C) 2001 Megan Potter
00005
00006 */
00007
00008 /** \file   kos.h
00009     \brief  Include everything KOS has to offer!
00010
00011     This file includes pretty much every KOS-related header file, so you don't
00012     have to figure out what you actually need. The ultimate for the truly lazy!
00013
00014     You may want to include individual header files yourself if you need more
00015     fine-grained control, as may be more appropriate for some projects.
00016
00017     \author Megan Potter
00018 */
00019
00020 #ifndef __KOS_H
00021 #define __KOS_H
00022
00023 /* The ultimate for the truly lazy: include and go! No more figuring out
00024    which headers to include for your project. */
00025
00026 #include <kos/cdefs.h>
00027 __BEGIN_DECLS
00028
00029 #include <ctype.h>
00030 #include <malloc.h>
00031 #include <stdio.h>
00032 #include <string.h>
00033 #include <unistd.h>
00034
00035 #include <kos/fs.h>
00036 #include <kos/fs_romdisk.h>
00037 #include <kos/fs_ramdisk.h>
00038 #include <kos/fs_dev.h>
00039 #include <kos/fs_pty.h>
00040 #include <kos/limits.h>
00041 #include <kos/thread.h>
00042 #include <kos/sem.h>
00043 #include <kos/rwsem.h>
00044 #include <kos/once.h>
00045 #include <kos/tls.h>
00046 #include <kos/mutex.h>
00047 #include <kos/cond.h>
00048 #include <kos/genwait.h>
00049 #include <kos/library.h>
00050 #include <kos/net.h>
00051 #include <kos/nmmgr.h>
00052 #include <kos/exports.h>
00053 #include <kos/dbgio.h>
00054 #include <kos/blockdev.h>
00055 #include <kos/dbglog.h>
00056 #include <kos/elf.h>
00057 #include <kos/fs_socket.h>
00058 #include <kos/string.h>
00059 #include <kos/init.h>
```

```
00060
00061 #include <arch/arch.h>
00062 #include <arch/cache.h>
00063 #include <arch/irq.h>
00064 #include <arch/spinlock.h>
00065 #include <arch/timer.h>
00066 #include <arch/wdt.h>
00067 #include <arch/types.h>
00068 #include <arch/exec.h>
00069 #include <arch/stack.h>
00070 #include <arch/byteorder.h>
00071 #include <arch/rtc.h>
00072
00073 #ifdef _arch_dreamcast
00074 #   include <arch/gdb.h>
00075 #   include <arch/mmu.h>
00076 #   include <arch/memory.h>
00077
00078 #   include <dc/asic.h>
00079 #   include <dc/biosfont.h>
00080 #   include <dc/cdrom.h>
00081 #   include <dc/fb_console.h>
00082 #   include <dc/flashrom.h>
00083 #   include <dc/fmath.h>
00084 #   include <dc/fs_dcload.h>
00085 #   include <dc/fs_dclsocket.h>
00086 #   include <dc/fs_iso9660.h>
00087 #   include <dc/fs_vmu.h>
00088 #   include <dc/glata.h>
00089 #   include <dc/g2bus.h>
00090 #   include <dc/maple.h>
00091 #   include <dc/maple/controller.h>
00092 #   include <dc/maple/dreameye.h>
00093 #   include <dc/maple/keyboard.h>
00094 #   include <dc/maple/mouse.h>
00095 #   include <dc/maple/purupuru.h>
00096 #   include <dc/maple/sip.h>
00097 #   include <dc/maple/vmu.h>
00098 #   include <dc/matrix3d.h>
00099 #   include <dc/matrix.h>
00100 #   include <dc/modem/modem.h>
00101 #   include <dc/net/broadband_adapter.h>
00102 #   include <dc/net/lan_adapter.h>
00103 #   include <dc/pvr.h>
00104 #   include <dc/scif.h>
00105 #   include <dc/sd.h>
00106 #   include <dc/sound/stream.h>
00107 #   include <dc/sound/sfxmgr.h>
00108 #   include <dc/spu.h>
00109 #   include <dc/sq.h>
00110 #   include <dc/ubc.h>
00111 #   include <dc/vblank.h>
00112 #   include <dc/vec3f.h>
00113 #   include <dc/video.h>
00114 #   include <dc/vmu_pkg.h>
00115 #   include <dc/vmufs.h>
00116 #else   /* _arch_dreamcast */
00117 #   error Invalid architecture or no architecture specified
00118 #endif
00119
00120 __END_DECLS
00121
00122 #endif
```

9.28 include/kos/blockdev.h File Reference

Definitions for a simple block device interface.

```
#include <sys/cdefs.h>
#include <stdint.h>
#include <sys/types.h>
```

Data Structures

- struct kos_blockdev_t

  A simple block device.

9.28.1 Detailed Description

Definitions for a simple block device interface.

This file contains the definition of a very simple block device that is to be used with filesystems in the kernel. This device interface is designed to abstract away direct hardware access and make it easier to interface the various filesystems that we may add support for to multiple potential devices.

The most common of these devices that people are probably interested in directly would be the Dreamcast SD card reader, and that was indeed the primary impetus to this device structure. However, it could also be used to support a file-based disk image or any number of other devices.

Author

Lawrence Sebald

9.29 blockdev.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/blockdev.h
00004    Copyright (C) 2012, 2013 Lawrence Sebald
00005 */
00006
00007 #ifndef __KOS_BLOCKDEV_H
00008 #define __KOS_BLOCKDEV_H
00009
00010 #include <sys/cdefs.h>
00011 __BEGIN_DECLS
00012
00013 #include <stdint.h>
00014 #include <sys/types.h>
00015
00016 /** \file   kos/blockdev.h
00017     \brief  Definitions for a simple block device interface.
00018
00019     This file contains the definition of a very simple block device that is to
00020     be used with filesystems in the kernel. This device interface is designed to
00021     abstract away direct hardware access and make it easier to interface the
00022     various filesystems that we may add support for to multiple potential
00023     devices.
00024
00025     The most common of these devices that people are probably interested in
00026     directly would be the Dreamcast SD card reader, and that was indeed the
00027     primary impetus to this device structure. However, it could also be used
00028     to support a file-based disk image or any number of other devices.
00029
00030     \author Lawrence Sebald
00031 */
00032
00033 /** \brief  A simple block device.
00034
00035     This structure represents a single block device. Each block device should be
00036     associated with exactly one filesystem and is used to actually read the data
00037     from the disk (or other device) where it is stored.
00038
00039     By using a block device with any new filesystems, we can abstract away a few
00040     things so that filesystems can be used with a variety of different
```

```
00041     "devices", such as the SD card reader for the Dreamcast or a disk image file
00042     of some sort.
00043
00044     \headerfile kos/blockdev.h
00045 */
00046 typedef struct kos_blockdev {
00047     void *dev_data;         /**< \brief Internal device data. */
00048     uint32_t l_block_size;  /**< \brief Log base 2 of the bytes per block. */
00049
00050     /** \brief  Initialize the block device.
00051
00052         This function should do any necessary initialization to use the block
00053         device passed in.
00054
00055         \param  d           The device to initialize.
00056         \retval 0           On success.
00057         \retval -1          On failure. Set errno as appropriate.
00058     */
00059     int (*init)(struct kos_blockdev *d);
00060
00061     /** \brief  Shut down the block device.
00062
00063         This function should do any teardown work that is needed to clean up
00064         the block device.
00065
00066         \param  d           The device to shut down.
00067         \retval 0           On success.
00068         \retval -1          On failure. Set errno as appropriate.
00069     */
00070     int (*shutdown)(struct kos_blockdev *d);
00071
00072     /** \brief  Read a number of blocks from the device.
00073
00074         This function should read the specified number of device blocks into
00075         the given buffer. The buffer will already be allocated by the caller.
00076
00077         \param  d           The device to read from.
00078         \param  block       The first block to read.
00079         \param  count       The number of blocks to read.
00080         \param  buf         The buffer to read into.
00081         \retval 0           On success.
00082         \retval -1          On failure. Set errno as appropriate.
00083     */
00084     int (*read_blocks)(struct kos_blockdev *d, uint64_t block, size_t count,
00085                        void *buf);
00086
00087     /** \brief  Write a number of blocks to the device.
00088
00089         This function should write the specified number of device blocks onto
00090         the device from the given buffer.
00091
00092         \param  d           The device to write to.
00093         \param  block       The first block to write.
00094         \param  count       The number of blocks to write.
00095         \param  buf         The buffer to write from.
00096         \retval 0           On success.
00097         \retval -1          On failure. Set errno as appropriate.
00098     */
00099     int (*write_blocks)(struct kos_blockdev *d, uint64_t block, size_t count,
00100                         const void *buf);
00101
00102     /** \brief  Count the number of blocks on the device.
00103
00104         This function should return the total number of blocks on the device.
00105         There is no expectation of the device to keep track of which blocks are
00106         in use or anything else of the sort.
00107
00108         \param  d           The device to read the block count from.
00109         \return             The number of blocks that the device has.
00110     */
00111     uint64_t (*count_blocks)(struct kos_blockdev *d);
00112
00113     /** \brief  Flush the write cache (if any) of the device.
00114
00115         This function shall signal to the device that any write caches that are
00116         present on the device shall be flushed so that all data written to this
00117         point shall persist to the underlying storage.
00118
00119         \param  d           The device to flush caches on.
00120         \retval 0           On success.
00121         \retval -1          On failure. Set errno as appropriate.
```

```
00122     */
00123     int (*flush)(struct kos_blockdev *d);
00124 } kos_blockdev_t;
00125
00126 __END_DECLS
00127
00128 #endif /* !__KOS_BLOCKDEV_H */
```

9.30 include/kos/cdefs.h File Reference

Definitions for builtin attributes and compiler directives.

`#include <sys/cdefs.h>`

Macros

- #define __noreturn __attribute__((__noreturn__))

  Identify a function that will never return.
- #define __pure __attribute__((__const__))

  Identify a function that has no side effects other than its return, and only uses its arguments for any work.
- #define __unused __attribute__((__unused__))

  Identify a function or variable that may be unused.
- #define __used __attribute__((used))

  Prevent a symbol from being removed from the binary.
- #define __weak __attribute__((weak))

  Identify a function or variable that may be overriden by another symbol.
- #define __dead2 __noreturn /* BSD compat */

  Alias for __noreturn. For BSD compatibility.
- #define __pure2 __pure /* ditto */

  Alias for __pure. For BSD compatibility.
- #define likely(exp) __builtin_expect(!!(exp), 1)

  Directive to inform the compiler the condition is in the likely path.
- #define unlikely(exp) __builtin_expect(!!(exp), 0)

  Directive to inform the compiler the condition is in the unlikely path.
- #define __deprecated __attribute__((deprecated))

  Mark something as deprecated. This should be used to warn users that a function/type/etc will be removed in a future version of KOS.
- #define __depr(m) __attribute__((deprecated(m)))

  Mark something as deprecated, with an informative message. This should be used to warn users that a function/type/etc will be removed in a future version of KOS and to suggest an alternative that they can use instead.
- #define __printflike(fmtarg, firstvararg) __attribute__((__format__ (__printf__, fmtarg, firstvararg)))

  Identify a function as accepting formatting like printf().
- #define __scanflike(fmtarg, firstvararg) __attribute__((__format__ (__scanf__, fmtarg, firstvararg)))

  Identify a function as accepting formatting like scanf().
- #define __fallthrough /* Fall through */
- #define __always_inline inline __attribute__((__always_inline__))

  Ask the compiler to always inline a given function.
- #define __RESTRICT
- #define __extension__

9.30.1 Detailed Description

Definitions for builtin attributes and compiler directives.

This file contains definitions of various **attribute** directives in shorter forms for use in programs. These typically aid in optimizations or provide the compiler with extra information about a symbol.

Author

Megan Potter

Lawrence Sebald

Falco Girgis

9.30.2 Macro Definition Documentation

__always_inline

```
#define __always_inline inline __attribute__((__always_inline__))
```

Ask the compiler to always inline a given function.

__dead2

```
#define __dead2 __noreturn /* BSD compat */
```

Alias for __noreturn. For BSD compatibility.

__depr

```
#define __depr(
              m ) __attribute__((deprecated(m)))
```

Mark something as deprecated, with an informative message. This should be used to warn users that a function/type/etc will be removed in a future version of KOS and to suggest an alternative that they can use instead.

Parameters

m	A string literal that is included with the warning message at compile time.

__deprecated

```
#define __deprecated __attribute__((deprecated))
```

Mark something as deprecated. This should be used to warn users that a function/type/etc will be removed in a future version of KOS.

__extension__

```
#define __extension__
```

__fallthrough

```
#define __fallthrough /* Fall through */
```

__noreturn

```
#define __noreturn __attribute__((__noreturn__))
```

Identify a function that will never return.

__printflike

```
#define __printflike(
             fmtarg,
             firstvararg )  __attribute__((__format__ (__printf__, fmtarg, firstvararg)))
```

Identify a function as accepting formatting like printf().

Using this macro allows GCC to typecheck calls to printf-like functions, which can aid in finding mistakes.

Parameters

fmtarg	The argument number (1-based) of the format string.
firstvararg	The argument number of the first vararg (the ...).

__pure

```
#define __pure __attribute__((__const__))
```

Identify a function that has no side effects other than its return, and only uses its arguments for any work.

__pure2

```
#define __pure2 __pure /* ditto */
```

Alias for __pure. For BSD compatibility.

__RESTRICT

```
#define __RESTRICT
```

__scanflike

```
#define __scanflike(
              fmtarg,
              firstvararg )  __attribute__((__format__ (__scanf__, fmtarg, firstvararg)))
```

Identify a function as accepting formatting like scanf().

Using this macro allows GCC to typecheck calls to scanf-like functions, which can aid in finding mistakes.

Parameters

fmtarg	The argument number (1-based) of the format string.
firstvararg	The argument number of the first vararg (the ...).

__unused

```
#define __unused __attribute__((__unused__))
```

Identify a function or variable that may be unused.

__used

```
#define __used __attribute__((used))
```

Prevent a symbol from being removed from the binary.

__weak

```
#define __weak __attribute__((weak))
```

Identify a function or variable that may be overriden by another symbol.

likely

```
#define likely(
              exp ) __builtin_expect(!!(exp), 1)
```

Directive to inform the compiler the condition is in the likely path.

This can be used around conditionals or loops to help inform the compiler which path to optimize for as the common-case.

Parameters

exp	Boolean expression which expected to be true.

See also

unlikely()

unlikely

```
#define unlikely(
              exp ) __builtin_expect(!!(exp), 0)
```

Directive to inform the compiler the condition is in the unlikely path.

This can be used around conditionals or loops to help inform the compiler which path to optimize against as the infrequent-case.

Parameters

exp	Boolean expression which is expected to be false.

See also

likely()

9.31 cdefs.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/cdefs.h
00004    Copyright (C) 2002, 2004 Megan Potter
00005    Copyright (C) 2020, 2023 Lawrence Sebald
00006    Copyright (C) 2023 Falco Girgis
00007
00008    Based loosely around some stuff in BSD's sys/cdefs.h
00009 */
00010
00011 /** \file   kos/cdefs.h
00012     \brief  Definitions for builtin attributes and compiler directives
00013
00014     This file contains definitions of various __attribute__ directives in
00015     shorter forms for use in programs. These typically aid  in optimizations
00016     or provide the compiler with extra information about a symbol.
00017
00018     \author Megan Potter
00019     \author Lawrence Sebald
00020     \author Falco Girgis
00021 */
00022
00023 #ifndef __KOS_CDEFS_H
00024 #define __KOS_CDEFS_H
00025
00026 #include <sys/cdefs.h>
00027
```

```
00028 /* Check GCC version */
00029 #if __GNUC__ <= 3
00030 #   warning Your GCC is too old. This will probably not work right.
00031 #endif
00032
00033 /* Special function/variable attributes */
00034
00035 #ifndef __noreturn
00036 /** \brief  Identify a function that will never return. */
00037 #define __noreturn  __attribute__((__noreturn__))
00038 #endif
00039
00040 #ifndef __pure
00041 /** \brief  Identify a function that has no side effects other than its return,
00042             and only uses its arguments for any work. */
00043 #define __pure      __attribute__((__const__))
00044 #endif
00045
00046 #ifndef __unused
00047 /** \brief  Identify a function or variable that may be unused. */
00048 #define __unused    __attribute__((__unused__))
00049 #endif
00050
00051 #ifndef __used
00052 /** \brief  Prevent a symbol from being removed from the binary. */
00053 #define __used      __attribute__((used))
00054 #endif
00055
00056 #ifndef __weak
00057 /** \brief  Identify a function or variable that may be overriden by another symbol. */
00058 #define __weak      __attribute__((weak))
00059 #endif
00060
00061 #ifndef __dead2
00062 /** \brief  Alias for \ref __noreturn. For BSD compatibility. */
00063 #define __dead2     __noreturn  /* BSD compat */
00064 #endif
00065
00066 #ifndef __pure2
00067 /** \brief  Alias for \ref __pure. For BSD compatibility. */
00068 #define __pure2     __pure      /* ditto */
00069 #endif
00070
00071 #ifndef likely
00072 /** \brief  Directive to inform the compiler the condition is in the likely path.
00073
00074     This can be used around conditionals or loops to help inform the
00075     compiler which path to optimize for as the common-case.
00076
00077     \param  exp     Boolean expression which expected to be true.
00078
00079     \sa unlikely()
00080 */
00081 #define likely(exp)   __builtin_expect(!!(exp), 1)
00082 #endif
00083
00084 #ifndef unlikely
00085 /** \brief  Directive to inform the compiler the condition is in the unlikely path.
00086
00087     This can be used around conditionals or loops to help inform the
00088     compiler which path to optimize against as the infrequent-case.
00089
00090     \param  exp     Boolean expression which is expected to be false.
00091
00092     \sa likely()
00093 */
00094 #define unlikely(exp) __builtin_expect(!!(exp), 0)
00095 #endif
00096
00097 #ifndef __deprecated
00098 /** \brief  Mark something as deprecated.
00099     This should be used to warn users that a function/type/etc will be removed
00100     in a future version of KOS. */
00101 #define __deprecated    __attribute__((deprecated))
00102 #endif
00103
00104 #ifndef __depr
00105 /** \brief  Mark something as deprecated, with an informative message.
00106     This should be used to warn users that a function/type/etc will be removed
00107     in a future version of KOS and to suggest an alternative that they can use
00108     instead.
```

```
00109     \param  m       A string literal that is included with the warning message
00110                     at compile time. */
00111 #define __depr(m) __attribute__((deprecated(m)))
00112 #endif
00113
00114 /* Printf/Scanf-like declaration */
00115 #ifndef __printflike
00116 /** \brief  Identify a function as accepting formatting like printf().
00117
00118     Using this macro allows GCC to typecheck calls to printf-like functions,
00119     which can aid in finding mistakes.
00120
00121     \param  fmtarg          The argument number (1-based) of the format string.
00122     \param  firstvararg     The argument number of the first vararg (the ...).
00123 */
00124 #define __printflike(fmtarg, firstvararg) \
00125     __attribute__((__format__ (__printf__, fmtarg, firstvararg)))
00126 #endif
00127
00128 #ifndef __scanflike
00129 /** \brief  Identify a function as accepting formatting like scanf().
00130
00131     Using this macro allows GCC to typecheck calls to scanf-like functions,
00132     which can aid in finding mistakes.
00133
00134     \param  fmtarg          The argument number (1-based) of the format string.
00135     \param  firstvararg     The argument number of the first vararg (the ...).
00136 */
00137 #define __scanflike(fmtarg, firstvararg) \
00138     __attribute__((__format__ (__scanf__, fmtarg, firstvararg)))
00139 #endif
00140
00141 #if __GNUC__ >= 7
00142 /** \brief  Identify a case statement that is expected to fall through to the
00143             statement underneath it. */
00144 #define __fallthrough __attribute__((__fallthrough__))
00145 #else
00146 #define __fallthrough /* Fall through */
00147 #endif
00148
00149 #ifndef __always_inline
00150 /** \brief  Ask the compiler to always inline a given function. */
00151 #define __always_inline inline __attribute__((__always_inline__))
00152 #endif
00153
00154 /* GCC macros for special cases */
00155 /* #if __GNUC__ ==  */
00156
00157 #ifndef __RESTRICT
00158 #if (__STDC_VERSION__ >= 199901L)
00159 #define __RESTRICT restrict
00160 #elif defined(__GNUC__) || defined(__GNUG__)
00161 #define __RESTRICT __restrict__
00162 #else /* < C99 and not GCC */
00163 #define __RESTRICT
00164 #endif
00165 #endif /* !__RESTRICT */
00166
00167 #ifndef __GNUC__
00168 #define __extension__
00169 #endif
00170
00171 #endif  /* __KOS_CDEFS_H */
```

9.32 include/kos/cond.h File Reference

Condition variables.

```
#include <kos/cdefs.h>
#include <arch/types.h>
#include <kos/thread.h>
#include <kos/mutex.h>
```

Data Structures

- struct condvar_t

  Condition variable.

Macros

- #define COND_INITIALIZER { 0, 0 }

  Initializer for a transient condvar.

Functions

- condvar_t * cond_create () __depr("Use cond_init or COND_INITIALIZER.")

  Allocate a new condition variable.
- int cond_init (condvar_t *cv)

  Initialize a condition variable.
- int cond_destroy (condvar_t *cv)

  Free a condition variable.
- int cond_wait (condvar_t *cv, mutex_t *m)

  Wait on a condition variable.
- int cond_wait_timed (condvar_t *cv, mutex_t *m, int timeout)

  Wait on a condition variable with a timeout.
- int cond_signal (condvar_t *cv)

  Signal a single thread waiting on the condition variable.
- int cond_broadcast (condvar_t *cv)

  Signal all threads waiting on the condition variable.

9.32.1 Detailed Description

Condition variables.

This file contains the definition of a Condition Variable. Condition Variables (or condvars for short) are used with a mutex to act as a lock and checkpoint pair for threads.

Basically, things work as follows (for the thread doing work):

- The associated mutex is locked.
- A predicate is checked to see if it is safe to do something.
- If it is not safe, you call cond_wait(), which releases the mutex.
- When cond_wait() returns, the mutex is reaquired, and work can go on.
- Update any predicates so that we konw that the work is done, and unlock the mutex.

Meanwhile, the thread updating the condition works as follows:

- Lock the mutex associated with the condvar.
- Produce work to be done.
- Call cond_signal() (with the associated mutex still locked), so that any threads waiting on the condvar will know they can continue on when the mutex is released, also update any predicates that say whether work can be done.
- Unlock the mutex so that worker threads can acquire the mutex and do whatever work needs to be done.

Condition variables can be quite useful when used properly, and provide a fairly easy way to wait for work to be ready to be done.

Condition variables should never be used with mutexes that are of the type MUTEX_TYPE_RECURSIVE. The lock will only be released once by the wait function, and thus you will end up deadlocking if you use a recursive mutex that has been locked more than once.

Author

Megan Potter

9.32.2 Macro Definition Documentation

COND_INITIALIZER

```
#define COND_INITIALIZER { 0, 0 }
```

Initializer for a transient condvar.

9.32.3 Function Documentation

cond_broadcast()

```
int cond_broadcast (
            condvar_t * cv )
```

Signal all threads waiting on the condition variable.

This function will wake up all threads that are waiting on the condition. The calling thread should be holding the associated mutex or recursive lock before calling this to guarantee sane behavior.

Parameters

cv	The condition to signal

Return values

0	On success
-1	On error, errno will be set as appropriate

Error Conditions:

EINVAL - the condvar was not initialized

cond_create()

```
condvar_t * cond_create ( )
```

Allocate a new condition variable.

This function allocates and initializes a new condition variable for use.

Deprecated This function is formally deprecated and should not be used in new code. Instead you should use either the static initializer or the cond_init() function.

Returns

The created condvar on success. NULL is returned on failure and errno is set as appropriate.

Error Conditions:

ENOMEM - out of memory

cond_destroy()

```
int cond_destroy (
            condvar_t * cv )
```

Free a condition variable.

This function frees a condition variable, releasing all memory associated with it (but not with the mutex that is associated with it). This will also wake all threads waiting on the condition.

Return values

0	On success (no error conditions currently defined)

cond_init()

```
int cond_init (
            condvar_t * cv )
```

Initialize a condition variable.

This function initializes a new condition variable for use.

Parameters

cv	The condition variable to initialize

Return values

0	On success
-1	On error, sets errno as appropriate

cond_signal()

```
int cond_signal (
            condvar_t * cv )
```

Signal a single thread waiting on the condition variable.

This function will wake up a single thread that is waiting on the condition. The calling thread should be holding the associated mutex or recursive lock before calling this to guarantee sane behavior.

Parameters

cv	The condition to signal

Return values

0	On success
-1	On error, errno will be set as appropriate

Error Conditions:

EINVAL - the condvar was not initialized

cond_wait()

```
int cond_wait (
            condvar_t * cv,
            mutex_t * m )
```

Wait on a condition variable.

This function will wait on the condition variable, unlocking the mutex and putting the calling thread to sleep as one atomic operation. The wait in this function has no timeout, and will sleep forever if the condition is not signalled.

The mutex will be locked and owned by the calling thread on return, regardless of whether it is a successful or error return.

Parameters

cv	The condition to wait on
m	The associated mutex

Return values

0	On success
-1	On error, sets errno as appropriate

Error Conditions:

EPERM - called inside an interrupt
EINVAL - the condvar was not initialized
EINVAL - the mutex is not initialized or not locked
ENOTRECOVERABLE - the condvar was destroyed while waiting

cond_wait_timed()

```
int cond_wait_timed (
              condvar_t * cv,
              mutex_t * m,
              int timeout )
```

Wait on a condition variable with a timeout.

This function will wait on the condition variable, unlocking the mutex and putting the calling thread to sleep as one atomic operation. If the timeout elapses before the condition is signalled, this function will return error. If a timeout of 0 is given, the call is equivalent to cond_wait() (there is no timeout).

The mutex will be locked and owned by the calling thread on return, regardless of whether it is a successful or error return.

Parameters

cv	The condition to wait on
m	The associated mutex
timeout	The number of milliseconds before timeout

Return values

0	On success
-1	On error, sets errno as appropriate

Error Conditions:

EPERM - called inside an interrupt
ETIMEDOUT - timed out
EINVAL - the condvar was not initialized
EINVAL - the mutex is not initialized or not locked
ENOTRECOVERABLE - the condvar was destroyed while waiting

9.33 cond.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    include/kos/cond.h
00004    Copyright (C) 2001, 2003 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/cond.h
00009     \brief  Condition variables.
00010     \ingroup kthreads
00011
00012     This file contains the definition of a Condition Variable. Condition
00013     Variables (or condvars for short) are used with a mutex to act as a lock and
00014     checkpoint pair for threads.
00015
00016     Basically, things work as follows (for the thread doing work):
00017     \li The associated mutex is locked.
00018     \li A predicate is checked to see if it is safe to do something.
00019     \li If it is not safe, you call cond_wait(), which releases the mutex.
00020     \li When cond_wait() returns, the mutex is reaquired, and work can go on.
00021     \li Update any predicates so that we konw that the work is done, and unlock
00022         the mutex.
00023
00024     Meanwhile, the thread updating the condition works as follows:
00025     \li Lock the mutex associated with the condvar.
00026     \li Produce work to be done.
00027     \li Call cond_signal() (with the associated mutex still locked), so that any
00028         threads waiting on the condvar will know they can continue on when the
00029         mutex is released, also update any predicates that say whether work can
00030         be done.
00031     \li Unlock the mutex so that worker threads can acquire the mutex and do
00032         whatever work needs to be done.
00033
00034     Condition variables can be quite useful when used properly, and provide a
00035     fairly easy way to wait for work to be ready to be done.
00036
00037     Condition variables should never be used with mutexes that are of the type
00038     MUTEX_TYPE_RECURSIVE. The lock will only be released once by the wait
00039     function, and thus you will end up deadlocking if you use a recursive mutex
00040     that has been locked more than once.
00041
00042     \author Megan Potter
00043 */
00044
00045 #ifndef __KOS_COND_H
00046 #define __KOS_COND_H
00047
00048 #include <kos/cdefs.h>
00049 __BEGIN_DECLS
00050
00051 #include <arch/types.h>
00052 #include <kos/thread.h>
00053 #include <kos/mutex.h>
00054
00055 /** \brief  Condition variable.
00056
00057     There are no public members of this structure for you to actually do
00058     anything with in your code, so don't try.
00059
00060     \headerfile kos/cond.h
00061 */
00062 typedef struct condvar {
00063     int dummy;
```

```
00064     int dynamic;
00065 } condvar_t;
00066
00067 /** \brief  Initializer for a transient condvar. */
00068 #define COND_INITIALIZER    { 0, 0 }
00069
00070 /** \brief  Allocate a new condition variable.
00071
00072     This function allocates and initializes a new condition variable for use.
00073
00074     \deprecated
00075     This function is formally deprecated and should not be used in new code.
00076     Instead you should use either the static initializer or the cond_init()
00077     function.
00078
00079     \return                 The created condvar on success. NULL is returned on
00080                             failure and errno is set as appropriate.
00081
00082     \par    Error Conditions:
00083     \em     ENOMEM - out of memory
00084 */
00085 condvar_t *cond_create() __depr("Use cond_init or COND_INITIALIZER.");
00086
00087 /** \brief  Initialize a condition variable.
00088
00089     This function initializes a new condition variable for use.
00090
00091     \param  cv              The condition variable to initialize
00092     \retval 0               On success
00093     \retval -1              On error, sets errno as appropriate
00094 */
00095 int cond_init(condvar_t *cv);
00096
00097 /** \brief  Free a condition variable.
00098
00099     This function frees a condition variable, releasing all memory associated
00100     with it (but not with the mutex that is associated with it). This will also
00101     wake all threads waiting on the condition.
00102
00103     \retval 0               On success (no error conditions currently defined)
00104 */
00105 int cond_destroy(condvar_t *cv);
00106
00107 /** \brief  Wait on a condition variable.
00108
00109     This function will wait on the condition variable, unlocking the mutex and
00110     putting the calling thread to sleep as one atomic operation. The wait in
00111     this function has no timeout, and will sleep forever if the condition is not
00112     signalled.
00113
00114     The mutex will be locked and owned by the calling thread on return,
00115     regardless of whether it is a successful or error return.
00116
00117     \param  cv              The condition to wait on
00118     \param  m               The associated mutex
00119     \retval 0               On success
00120     \retval -1              On error, sets errno as appropriate
00121
00122     \par    Error Conditions:
00123     \em     EPERM - called inside an interrupt \n
00124     \em     EINVAL - the condvar was not initialized \n
00125     \em     EINVAL - the mutex is not initialized or not locked \n
00126     \em     ENOTRECOVERABLE - the condvar was destroyed while waiting
00127 */
00128 int cond_wait(condvar_t *cv, mutex_t * m);
00129
00130 /** \brief  Wait on a condition variable with a timeout.
00131
00132     This function will wait on the condition variable, unlocking the mutex and
00133     putting the calling thread to sleep as one atomic operation. If the timeout
00134     elapses before the condition is signalled, this function will return error.
00135     If a timeout of 0 is given, the call is equivalent to cond_wait() (there is
00136     no timeout).
00137
00138     The mutex will be locked and owned by the calling thread on return,
00139     regardless of whether it is a successful or error return.
00140
00141     \param  cv              The condition to wait on
00142     \param  m               The associated mutex
00143     \param  timeout         The number of milliseconds before timeout
00144     \retval 0               On success
```

```
00145     \retval -1              On error, sets errno as appropriate
00146
00147     \par    Error Conditions:
00148     \em     EPERM - called inside an interrupt \n
00149     \em     ETIMEDOUT - timed out \n
00150     \em     EINVAL - the condvar was not initialized \n
00151     \em     EINVAL - the mutex is not initialized or not locked \n
00152     \em     ENOTRECOVERABLE - the condvar was destroyed while waiting
00153 */
00154 int cond_wait_timed(condvar_t *cv, mutex_t * m, int timeout);
00155
00156 /** \brief  Signal a single thread waiting on the condition variable.
00157
00158     This function will wake up a single thread that is waiting on the condition.
00159     The calling thread should be holding the associated mutex or recursive lock
00160     before calling this to guarantee sane behavior.
00161
00162     \param  cv              The condition to signal
00163     \retval 0               On success
00164     \retval -1              On error, errno will be set as appropriate
00165
00166     \par    Error Conditions:
00167     \em     EINVAL - the condvar was not initialized
00168 */
00169 int cond_signal(condvar_t *cv);
00170
00171 /** \brief  Signal all threads waiting on the condition variable.
00172
00173     This function will wake up all threads that are waiting on the condition.
00174     The calling thread should be holding the associated mutex or recursive lock
00175     before calling this to guarantee sane behavior.
00176
00177     \param  cv              The condition to signal
00178     \retval 0               On success
00179     \retval -1              On error, errno will be set as appropriate
00180
00181     \par    Error Conditions:
00182     \em     EINVAL - the condvar was not initialized
00183 */
00184 int cond_broadcast(condvar_t *cv);
00185
00186 __END_DECLS
00187
00188 #endif  /* __KOS_COND_H */
```

9.34 include/kos/dbgio.h File Reference

Debug I/O.

```
#include <kos/cdefs.h>
#include <arch/types.h>
```

Data Structures

- struct dbgio_handler_t

  Debug I/O Interface.

Macros

- #define DBGIO_MODE_POLLED 0

  Polled I/O mode.
- #define DBGIO_MODE_IRQ 1

  IRQ-based I/O mode.

Functions

- int dbgio_dev_select (const char *name)

  Select a new dbgio interface by name.
- const char * dbgio_dev_get (void)

  Fetch the name of the currently selected dbgio interface.
- int dbgio_init (void)

  Initialize the dbgio console.
- int dbgio_set_irq_usage (int mode)

  Set IRQ usage.
- int dbgio_read (void)

  Read one character from the console.
- int dbgio_write (int c)

  Write one character to the console.
- int dbgio_flush (void)

  Flush any queued output.
- int dbgio_write_buffer (const uint8 *data, int len)

  Write an entire buffer of data to the console.
- int dbgio_read_buffer (uint8 *data, int len)

  Read an entire buffer of data from the console.
- int dbgio_write_buffer_xlat (const uint8 *data, int len)

  Write an entire buffer of data to the console (potentially with newline transformations).
- int dbgio_write_str (const char *str)

  Write a NUL-terminated string to the console.
- void dbgio_disable (void)

  Disable debug I/O globally.
- void dbgio_enable (void)

  Enable debug I/O globally.
- int dbgio_printf (const char *fmt,...) __printflike(1

  Built-in debug I/O printf function.

9.34.1 Detailed Description

Debug I/O.

This file contains the Debug I/O system, which abstracts things so that various types of debugging tools can be used by programs in KOS. Included among these tools is the dcload console (dcload-serial, dcload-ip, and fs_dclsocket), a raw serial console, and a framebuffer based console.

Author

Megan Potter

9.34.2 Macro Definition Documentation

DBGIO_MODE_IRQ

```
#define DBGIO_MODE_IRQ 1
```

IRQ-based I/O mode.

See also

dbgio_set_irq_usage()

DBGIO_MODE_POLLED

```
#define DBGIO_MODE_POLLED 0
```

Polled I/O mode.

See also

dbgio_set_irq_usage()

9.34.3 Function Documentation

dbgio_dev_get()

```
const char * dbgio_dev_get (
            void  )
```

Fetch the name of the currently selected dbgio interface.

Returns

The name of the current dbgio interface (or NULL if no device is selected)

dbgio_dev_select()

```
int dbgio_dev_select (
            const char * name )
```

Select a new dbgio interface by name.

This function manually selects a new dbgio interface by name. This function will allow you to select a device, even if it is not detected.

Parameters

name	The dbgio interface to select

Return values

0	On success
-1	On error

Error Conditions:

ENODEV - The specified device could not be initialized

dbgio_disable()

```
void dbgio_disable (
             void  )
```

Disable debug I/O globally.

dbgio_enable()

```
void dbgio_enable (
             void  )
```

Enable debug I/O globally.

dbgio_flush()

```
int dbgio_flush (
             void  )
```

Flush any queued output.

Return values

0	On success
-1	On error (errno should be set as appropriate)

dbgio_init()

```
int dbgio_init (
             void  )
```

Initialize the dbgio console.

This function is called internally, and shouldn't need to be called by any user programs.

Return values

0	On success
-1	On error

Error Conditions:

ENODEV - No devices could be detected/initialized

dbgio_printf()

```
int dbgio_printf (
             const char * fmt,
              ...  )
```

Built-in debug I/O printf function.

Parameters

fmt	A printf() style format string
...	Format arguments

Returns

The number of bytes written, or $<$0 on error (errno should be set as appropriate)

dbgio_read()

```
int dbgio_read (
             void  )
```

Read one character from the console.

Return values

0	On success
-1	On error (errno should be set as appropriate)

dbgio_read_buffer()

```
int dbgio_read_buffer (
            uint8 * data,
            int len )
```

Read an entire buffer of data from the console.

Parameters

data	The buffer to read into
len	The length of the buffer

Returns

Number of characters read on success, or -1 on failure (errno should be set as appropriate)

dbgio_set_irq_usage()

```
int dbgio_set_irq_usage (
            int mode )
```

Set IRQ usage.

The dbgio system defaults to polled usage. Some devices may not support IRQ mode at all.

Parameters

mode	The mode to use

Return values

0	On success
-1	On error (errno should be set as appropriate)

dbgio_write()

```
int dbgio_write (
            int c )
```

Write one character to the console.

Parameters

c	The character to write

Return values

1	On success (number of characters written)
-1	On error (errno should be set as appropriate)

Note

Interfaces may require a call to flush() before the output is actually flushed to the console.

dbgio_write_buffer()

```
int dbgio_write_buffer (
            const uint8 * data,
            int len )
```

Write an entire buffer of data to the console.

Parameters

data	The buffer to write
len	The length of the buffer

Returns

Number of characters written on success, or -1 on failure (errno should be set as appropriate)

dbgio_write_buffer_xlat()

```
int dbgio_write_buffer_xlat (
            const uint8 * data,
            int len )
```

Write an entire buffer of data to the console (potentially with newline transformations).

Parameters

data	The buffer to write
len	The length of the buffer

Returns

Number of characters written on success, or -1 on failure (errno should be set as appropriate)

dbgio_write_str()

```
int dbgio_write_str (
            const char * str )
```

Write a NUL-terminated string to the console.

Parameters

str	The string to write

Returns

Number of characters written on success, or -1 on failure (errno should be set as appropriate)

9.35 dbgio.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/include/dbgio.h
00004    Copyright (C)2000,2004 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/dbgio.h
00009     \brief  Debug I/O.
00010
00011     This file contains the Debug I/O system, which abstracts things so that
00012     various types of debugging tools can be used by programs in KOS. Included
00013     among these tools is the dcload console (dcload-serial, dcload-ip, and
00014     fs_dclsocket), a raw serial console, and a framebuffer based console.
00015
00016     \author Megan Potter
00017 */
00018
00019 #ifndef __KOS_DBGIO_H
00020 #define __KOS_DBGIO_H
00021
00022 #include <kos/cdefs.h>
00023 __BEGIN_DECLS
00024
00025 #include <arch/types.h>
00026
00027 /** \brief  Debug I/O Interface.
00028
00029     This struct represents a single dbgio interface. This should represent
00030     a generic pollable console interface. We will store an ordered list of
00031     these statically linked into the program and fall back from one to the
00032     next until one returns true for detected(). Note that the last device in
00033     this chain is the null console, which will always return true.
00034
00035     \headerfile kos/dbgio.h
00036 */
00037 typedef struct dbgio_handler {
00038     /** \brief  Name of the dbgio handler */
00039     const char  * name;
00040
00041     /** \brief  Detect this debug interface.
00042         \retval 1           If the device is available and useable
00043         \retval 0           If the device is unavailable
00044     */
00045     int (*detected)(void);
00046
00047     /** \brief  Initialize this debug interface with default parameters.
00048         \retval 0           On success
00049         \retval -1          On failure
```

```
00050     */
00051     int (*init)(void);
00052
00053     /** \brief  Shutdown this debug interface.
00054         \retval 0           On success
00055         \retval -1          On failure
00056     */
00057     int (*shutdown)(void);
00058
00059     /** \brief  Set either polled or IRQ usage for this interface.
00060         \param  mode        1 for IRQ-based usage, 0 for polled I/O
00061         \retval 0           On success
00062         \retval -1          On failure
00063     */
00064     int (*set_irq_usage)(int mode);
00065
00066     /** \brief  Read one character from the console.
00067         \retval 0           On success
00068         \retval -1          On failure (set errno as appropriate)
00069     */
00070     int (*read)(void);
00071
00072     /** \brief  Write one character to the console.
00073         \param  c           The character to write
00074         \retval 1           On success
00075         \retval -1          On error (set errno as appropriate)
00076         \note               Interfaces may require a call to flush() before the
00077                             output is actually flushed to the console.
00078     */
00079     int (*write)(int c);
00080
00081     /** \brief  Flush any queued output.
00082         \retval 0           On success
00083         \retval -1          On error (set errno as appropriate)
00084     */
00085     int (*flush)(void);
00086
00087     /** \brief  Write an entire buffer of data to the console.
00088         \param  data        The buffer to write
00089         \param  len         The length of the buffer
00090         \param  xlat        If non-zero, newline transformations may occur
00091         \return             Number of characters written on success, or -1 on
00092                             failure (set errno as appropriate)
00093     */
00094     int (*write_buffer)(const uint8 *data, int len, int xlat);
00095
00096     /** \brief  Read an entire buffer of data from the console.
00097         \param  data        The buffer to read into
00098         \param  len         The length of the buffer
00099         \return             Number of characters read on success, or -1 on
00100                             failure (set errno as appropriate)
00101     */
00102     int (*read_buffer)(uint8 *data, int len);
00103 } dbgio_handler_t;
00104
00105 /** \cond */
00106 /* These two should be initialized in arch. */
00107 extern dbgio_handler_t * dbgio_handlers[];
00108 extern int dbgio_handler_cnt;
00109
00110 /* This is defined by the shared code, in case there's no valid handler. */
00111 extern dbgio_handler_t dbgio_null;
00112 /** \endcond */
00113
00114 /** \brief  Select a new dbgio interface by name.
00115
00116     This function manually selects a new dbgio interface by name. This function
00117     will allow you to select a device, even if it is not detected.
00118
00119     \param  name            The dbgio interface to select
00120     \retval 0               On success
00121     \retval -1              On error
00122
00123     \par    Error Conditions:
00124     \em     ENODEV - The specified device could not be initialized
00125 */
00126 int dbgio_dev_select(const char * name);
00127
00128 /** \brief  Fetch the name of the currently selected dbgio interface.
00129     \return                 The name of the current dbgio interface (or NULL if
00130                             no device is selected)
```

```
00131 */
00132 const char * dbgio_dev_get(void);
00133
00134 /** \brief  Initialize the dbgio console.
00135
00136     This function is called internally, and shouldn't need to be called by any
00137     user programs.
00138
00139     \retval 0               On success
00140     \retval -1              On error
00141     \par    Error Conditions:
00142     \em     ENODEV - No devices could be detected/initialized
00143 */
00144 int dbgio_init(void);
00145
00146 /** \brief  Set IRQ usage.
00147
00148     The dbgio system defaults to polled usage. Some devices may not support IRQ
00149     mode at all.
00150
00151     \param  mode            The mode to use
00152     \retval 0               On success
00153     \retval -1              On error (errno should be set as appropriate)
00154 */
00155 int dbgio_set_irq_usage(int mode);
00156
00157 /** \brief  Polled I/O mode.
00158     \see    dbgio_set_irq_usage()
00159 */
00160 #define DBGIO_MODE_POLLED 0
00161
00162 /** \brief  IRQ-based I/O mode.
00163     \see    dbgio_set_irq_usage()
00164 */
00165 #define DBGIO_MODE_IRQ 1
00166
00167 /** \brief  Read one character from the console.
00168     \retval 0               On success
00169     \retval -1              On error (errno should be set as appropriate)
00170 */
00171 int dbgio_read(void);
00172
00173 /** \brief  Write one character to the console.
00174     \param  c               The character to write
00175     \retval 1               On success (number of characters written)
00176     \retval -1              On error (errno should be set as appropriate)
00177     \note                   Interfaces may require a call to flush() before the
00178                             output is actually flushed to the console.
00179 */
00180 int dbgio_write(int c);
00181
00182 /** \brief  Flush any queued output.
00183     \retval 0               On success
00184     \retval -1              On error (errno should be set as appropriate)
00185 */
00186 int dbgio_flush(void);
00187
00188 /** \brief  Write an entire buffer of data to the console.
00189     \param  data            The buffer to write
00190     \param  len             The length of the buffer
00191     \return                 Number of characters written on success, or -1 on
00192                             failure (errno should be set as appropriate)
00193 */
00194 int dbgio_write_buffer(const uint8 *data, int len);
00195
00196 /** \brief  Read an entire buffer of data from the console.
00197     \param  data            The buffer to read into
00198     \param  len             The length of the buffer
00199     \return                 Number of characters read on success, or -1 on
00200                             failure (errno should be set as appropriate)
00201 */
00202 int dbgio_read_buffer(uint8 *data, int len);
00203
00204 /** \brief  Write an entire buffer of data to the console (potentially with
00205             newline transformations).
00206     \param  data            The buffer to write
00207     \param  len             The length of the buffer
00208     \return                 Number of characters written on success, or -1 on
00209                             failure (errno should be set as appropriate)
00210 */
00211 int dbgio_write_buffer_xlat(const uint8 *data, int len);
```

```
00212
00213 /** \brief  Write a NUL-terminated string to the console.
00214     \param  str             The string to write
00215     \return                 Number of characters written on success, or -1 on
00216                             failure (errno should be set as appropriate)
00217 */
00218 int dbgio_write_str(const char *str);
00219
00220 /** \brief  Disable debug I/O globally. */
00221 void dbgio_disable(void);
00222
00223 /** \brief  Enable debug I/O globally. */
00224 void dbgio_enable(void);
00225
00226 /** \brief  Built-in debug I/O printf function.
00227     \param  fmt             A printf() style format string
00228     \param  ...             Format arguments
00229     \return                 The number of bytes written, or <0 on error (errno
00230                             should be set as appropriate)
00231 */
00232 int dbgio_printf(const char *fmt, ...) __printflike(1, 2);
00233
00234 __END_DECLS
00235
00236 #endif  /* __KOS_DBGIO_H */
00237
```

9.36 include/kos/dbglog.h File Reference

A debugging log.

```
#include <kos/cdefs.h>
#include <unistd.h>
#include <stdarg.h>
#include <kos/fs.h>
```

Macros

- #define DBG_DEAD 0

  The system is dead.
- #define DBG_CRITICAL 1

  A critical error message.
- #define DBG_ERROR 2

  A normal error message.
- #define DBG_WARNING 3

  Potential problem.
- #define DBG_NOTICE 4

  Normal but significant.
- #define DBG_INFO 5

  Informational messages.
- #define DBG_DEBUG 6

  User debug messages.
- #define DBG_KDEBUG 7

  Kernel debug messages.

Functions

- void dbglog (int level, const char *fmt,...) __printflike(2

  Kernel debugging printf.
- void dbglog_set_level (int level)

  Set the debugging log level.

9.36.1 Detailed Description

A debugging log.

This file contains declarations related a debugging log. This log can be used to restrict log messages, for instance to make it so that only the most urgent of messages get printed for a release version of a program.

Author

Megan Potter

9.36.2 Function Documentation

dbglog()

```
void dbglog (
            int level,
            const char * fmt,
             ...  )
```

Kernel debugging printf.

This function is similar to printf(), but filters its output through a log level check before being printed. This way, you can set the level of debug info you want to see (or want your users to see).

Parameters

level	The level of importance of this message.
fmt	Message format string.
...	Format arguments

See also

Log levels for dbglog

dbglog_set_level()

```
void dbglog_set_level (
            int level )
```

Set the debugging log level.

This function sets the level for which dbglog() will ignore messages for if the message has a higher level.

Parameters

level	The level to stop paying attention after.

See also

Log levels for dbglog

9.37 dbglog.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/dbglog.h
00004    Copyright (C)2004 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/dbglog.h
00009     \brief  A debugging log.
00010
00011     This file contains declarations related a debugging log. This log can be
00012     used to restrict log messages, for instance to make it so that only the most
00013     urgent of messages get printed for a release version of a program.
00014
00015     \author Megan Potter
00016 */
00017
00018 #ifndef __KOS_DBGLOG_H
00019 #define __KOS_DBGLOG_H
00020
00021 #include <kos/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 #include <unistd.h>
00025 #include <stdarg.h>
00026 #include <kos/fs.h>
00027
00028 /** \brief  Kernel debugging printf.
00029
00030     This function is similar to printf(), but filters its output through a log
00031     level check before being printed. This way, you can set the level of debug
00032     info you want to see (or want your users to see).
00033
00034     \param  level           The level of importance of this message.
00035     \param  fmt             Message format string.
00036     \param  ...             Format arguments
00037     \see    dbglog_levels
00038 */
00039 void dbglog(int level, const char *fmt, ...) __printflike(2, 3);
00040
00041 /** \defgroup   dbglog_levels   Log levels for dbglog
00042
00043     This is the list of levels that are allowed to be passed into the dbglog()
00044     function, representing different levels of importance.
00045
00046     @{
00047 */
00048 #define DBG_DEAD        0       /**< \brief The system is dead */
00049 #define DBG_CRITICAL    1       /**< \brief A critical error message */
00050 #define DBG_ERROR       2       /**< \brief A normal error message */
00051 #define DBG_WARNING     3       /**< \brief Potential problem */
00052 #define DBG_NOTICE      4       /**< \brief Normal but significant */
00053 #define DBG_INFO        5       /**< \brief Informational messages */
00054 #define DBG_DEBUG       6       /**< \brief User debug messages */
```

```
00055 #define DBG_KDEBUG      7       /**< \brief Kernel debug messages */
00056 /** @} */
00057
00058 /** \brief  Set the debugging log level.
00059
00060     This function sets the level for which dbglog() will ignore messages for if
00061     the message has a higher level.
00062
00063     \param  level           The level to stop paying attention after.
00064     \see    dbglog_levels
00065 */
00066 void dbglog_set_level(int level);
00067
00068 __END_DECLS
00069
00070 #endif  /* __KOS_DBGLOG_H */
00071
```

9.38 include/kos/elf.h File Reference

ELF binary loading support.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <sys/queue.h>
```

Data Structures

- struct elf_hdr_t

  ELF file header.
- struct elf_shdr_t

  ELF Section header.
- struct elf_sym_t

  Symbol table entry.
- struct elf_rela_t

  ELF Relocation entry (with explicit addend).
- struct elf_rel_t

  ELF Relocation entry (without explicit addend).
- struct elf_prog_t

  Kernel-specific definition of a loaded ELF binary.

Macros

- #define EM_386 3

  x86 (IA32)
- #define EM_ARM 40

  ARM.
- #define EM_SH 42

  SuperH.
- #define SHT_NULL 0

  Inactive section.

- #define SHT_PROGBITS 1

  Program code/data.
- #define SHT_SYMTAB 2

  Full symbol table.
- #define SHT_STRTAB 3

  String table.
- #define SHT_RELA 4

  Relocation table, with addends.
- #define SHT_HASH 5

  Symbol hash table.
- #define SHT_DYNAMIC 6

  Dynamic linking info.
- #define SHT_NOTE 7

  Notes section.
- #define SHT_NOBITS 8

  A section that occupies no space in the file.
- #define SHT_REL 9

  Relocation table, no addends.
- #define SHT_SHLIB 10

  Reserved.
- #define SHT_DYNSYM 11

  Dynamic-only sym tab.
- #define SHT_LOPROC 0x70000000

  Start of processor specific types.
- #define SHT_HIPROC 0x7fffffff

  End of processor specific types.
- #define SHT_LOUSER 0x80000000

  Start of program specific types.
- #define SHT_HIUSER 0xffffffff

  End of program specific types.
- #define SHF_WRITE 1

  Writable data.
- #define SHF_ALLOC 2

  Resident.
- #define SHF_EXECINSTR 4

  Executable instructions.
- #define SHF_MASKPROC 0xf0000000

  Processor specific mask.
- #define SHN_UNDEF 0

  Undefined, missing, irrelevant.
- #define SHN_ABS 0xfff1

  Absolute values.
- #define STB_LOCAL 0

  Local (non-exported) symbol.
- #define STB_GLOBAL 1

  Global (exported) symbol.
- #define STB_WEAK 2

Weak-linked symbol.

- #define STT_NOTYPE 0

  Symbol has no type.
- #define STT_OBJECT 1

  Symbol is an object.
- #define STT_FUNC 2

  Symbol is a function.
- #define STT_SECTION 3

  Symbol is a section.
- #define STT_FILE 4

  Symbol is a file name.
- #define ELF32_ST_BIND(info) ((info) >> 4)

  Retrieve the binding type for a symbol.
- #define ELF32_ST_TYPE(info) ((info) & 0xf)

  Retrieve the symbol type for a symbol.
- #define R_SH_DIR32 1

  SuperH: Rel = Symbol + Addend.
- #define R_386_32 1

  x86: Rel = Symbol + Addend
- #define R_386_PC32 2

  x86: Rel = Symbol + Addend - Value
- #define ELF32_R_SYM(i) ((i) >> 8)

  Retrieve the symbol index from a relocation entry.
- #define ELF32_R_TYPE(i) ((uint8)(i))

  Retrieve the relocation type from a relocation entry.

Functions

- int elf_load (const char *fn, struct klibrary *shell, elf_prog_t *out)

  Load an ELF binary.
- void elf_free (elf_prog_t *prog)

  Free a loaded ELF program.

9.38.1 Detailed Description

ELF binary loading support.

This file contains the support functionality for loading ELF binaries in KOS. This includes the various header structures and whatnot that are used in ELF files to store code/data/relocations/etc. This isn't necessarily meant for running multiple processes, but more for loadable library support within KOS.

Author

Megan Potter

9.38.2 Macro Definition Documentation

ELF32_R_SYM

```
#define ELF32_R_SYM(
              i ) ((i) >> 8)
```

Retrieve the symbol index from a relocation entry.

Parameters

i	The info field of an elf_rel_t or elf_rela_t.

Returns

The symbol table index from that relocation entry.

ELF32_R_TYPE

```
#define ELF32_R_TYPE(
              i ) ((uint8)(i))
```

Retrieve the relocation type from a relocation entry.

Parameters

i	The info field of an elf_rel_t or an elf_rela_t.

Returns

The relocation type of that relocation.

See also

ELF relocation types

ELF32_ST_BIND

```
#define ELF32_ST_BIND(
              info ) ((info) >> 4)
```

Retrieve the binding type for a symbol.

Parameters

info	The info field of an elf_sym_t.

Returns

The binding type of the symbol.

See also

Symbol binding types.

ELF32_ST_TYPE

```
#define ELF32_ST_TYPE(
              info ) ((info) & 0xf)
```

Retrieve the symbol type for a symbol.

Parameters

info	The info field of an elf_sym_t.

Returns

The symbol type of the symbol.

See also

Symbol types.

9.38.3 Function Documentation

elf_free()

```
void elf_free (
             elf_prog_t * prog )
```

Free a loaded ELF program.

This function cleans up an ELF binary that was loaded with elf_load().

Parameters

prog	The loaded binary to clean up.

elf_load()

```
int elf_load (
             const char * fn,
             struct klibrary * shell,
             elf_prog_t * out )
```

Load an ELF binary.

This function loads an ELF binary from the VFS and fills in an elf_prog_t for it.

Parameters

fn	The filename of the binary on the VFS.
shell	Unused?
out	Storage for the binary that will be loaded.

Returns

0 on success, <0 on failure.

9.39 elf.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/elf.h
00004    Copyright (C)2000,2001,2003 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/elf.h
00009     \brief  ELF binary loading support.
00010
00011     This file contains the support functionality for loading ELF binaries in
00012     KOS. This includes the various header structures and whatnot that are used
00013     in ELF files to store code/data/relocations/etc. This isn't necessarily
00014     meant for running multiple processes, but more for loadable library support
00015     within KOS.
00016
00017     \author Megan Potter
00018 */
00019
00020 #ifndef __KOS_ELF_H
00021 #define __KOS_ELF_H
00022
00023 #include <sys/cdefs.h>
00024 __BEGIN_DECLS
00025
00026 #include <arch/types.h>
00027 #include <sys/queue.h>
00028
00029 /** \brief  ELF file header.
00030
00031     This header is at the beginning of any valid ELF binary and serves to
00032     identify the architecture of the binary and various data about it.
00033
00034     \headerfile kos/elf.h
00035 */
00036 struct elf_hdr_t {
00037     uint8 ident[16];    /**< \brief ELF identifier */
00038     uint16 type;        /**< \brief ELF file type */
00039     uint16 machine;     /**< \brief ELF file architecture */
00040     uint32 version;     /**< \brief Object file version */
00041     uint32 entry;       /**< \brief Entry point */
00042     uint32 phoff;       /**< \brief Program header offset */
00043     uint32 shoff;       /**< \brief Section header offset */
00044     uint32 flags;       /**< \brief Processor flags */
00045     uint16 ehsize;      /**< \brief ELF header size in bytes */
00046     uint16 phentsize;   /**< \brief Program header entry size */
00047     uint16 phnum;       /**< \brief Program header entry count */
00048     uint16 shentsize;   /**< \brief Section header entry size */
00049     uint16 shnum;       /**< \brief Section header entry count */
00050     uint16 shstrndx;    /**< \brief String table section index */
00051 };
```

```
00052
00053 /** \defgroup elf_archs                ELF architecture types
00054
00055     These are the various architectures that we might care about for ELF files.
00056
00057     @{
00058 */
00059 #define EM_386  3   /**< \brief x86 (IA32) */
00060 #define EM_ARM  40  /**< \brief ARM */
00061 #define EM_SH   42  /**< \brief SuperH */
00062 /** @} */
00063
00064 /** \defgroup elf_sections             Section header types
00065
00066     These are the various types of section headers that can exist in an ELF
00067     file.
00068
00069     @{
00070 */
00071 #define SHT_NULL        0       /**< \brief Inactive section */
00072 #define SHT_PROGBITS    1       /**< \brief Program code/data */
00073 #define SHT_SYMTAB      2       /**< \brief Full symbol table */
00074 #define SHT_STRTAB      3       /**< \brief String table */
00075 #define SHT_RELA        4       /**< \brief Relocation table, with addends */
00076 #define SHT_HASH        5       /**< \brief Symbol hash table */
00077 #define SHT_DYNAMIC     6       /**< \brief Dynamic linking info */
00078 #define SHT_NOTE        7       /**< \brief Notes section */
00079 #define SHT_NOBITS      8       /**< \brief A section that occupies no space in
00080 the file */
00081 #define SHT_REL         9       /**< \brief Relocation table, no addends */
00082 #define SHT_SHLIB       10      /**< \brief Reserved */
00083 #define SHT_DYNSYM      11      /**< \brief Dynamic-only sym tab */
00084 #define SHT_LOPROC  0x70000000  /**< \brief Start of processor specific types */
00085 #define SHT_HIPROC  0x7fffffff  /**< \brief End of processor specific types */
00086 #define SHT_LOUSER  0x80000000  /**< \brief Start of program specific types */
00087 #define SHT_HIUSER  0xffffffff  /**< \brief End of program specific types */
00088 /** @} */
00089
00090 /** \defgroup elf_hdrflags             Section header flags
00091
00092     These are the flags that can be set on a section header. These are related
00093     to whether the section should reside in memory and permissions on it.
00094
00095     @{
00096 */
00097 #define SHF_WRITE       1           /**< \brief Writable data */
00098 #define SHF_ALLOC       2           /**< \brief Resident */
00099 #define SHF_EXECINSTR   4           /**< \brief Executable instructions */
00100 #define SHF_MASKPROC    0xf0000000  /**< \brief Processor specific mask */
00101 /** @} */
00102
00103 /** \defgroup elf_specsec              Special section indeces
00104
00105     These are the indices to be used in special situations in the section array.
00106
00107     @{
00108 */
00109 #define SHN_UNDEF   0       /**< \brief Undefined, missing, irrelevant */
00110 #define SHN_ABS     0xfff1  /**< \brief Absolute values */
00111 /** @} */
00112
00113 /** \brief  ELF Section header.
00114
00115     This structure represents the header on each ELF section.
00116
00117     \headerfile kos/elf.h
00118 */
00119 struct elf_shdr_t {
00120     uint32 name;        /**< \brief Index into string table */
00121     uint32 type;        /**< \brief Section type \see elf_sections */
00122     uint32 flags;       /**< \brief Section flags \see elf_hdrflags */
00123     uint32 addr;        /**< \brief In-memory offset */
00124     uint32 offset;      /**< \brief On-disk offset */
00125     uint32 size;        /**< \brief Size (if SHT_NOBITS, amount of 0s needed) */
00126     uint32 link;        /**< \brief Section header table index link */
00127     uint32 info;        /**< \brief Section header extra info */
00128     uint32 addralign;   /**< \brief Alignment constraints */
00129     uint32 entsize;     /**< \brief Fixed-size table entry sizes */
00130 };
00131 /* Link and info fields:
00132
```

```
00133 switch (sh_type) {
00134     case SHT_DYNAMIC:
00135         link = section header index of the string table used by
00136             the entries in this section
00137         info = 0
00138     case SHT_HASH:
00139         ilnk = section header index of the string table to which
00140             this info applies
00141         info = 0
00142     case SHT_REL, SHT_RELA:
00143         link = section header index of associated symbol table
00144         info = section header index of section to which reloc applies
00145     case SHT_SYMTAB, SHT_DYNSYM:
00146         link = section header index of associated string table
00147         info = one greater than the symbol table index of the last
00148             local symbol (binding STB_LOCAL)
00149 }
00150
00151 */
00152
00153 /** \defgroup elf_binding               Symbol binding types.
00154
00155     These are the values that can be set to say how a symbol is bound in an ELF
00156     binary. This is stored in the upper 4 bits of the info field in elf_sym_t.
00157
00158     @{
00159 */
00160 #define STB_LOCAL   0       /**< \brief Local (non-exported) symbol */
00161 #define STB_GLOBAL  1       /**< \brief Global (exported) symbol */
00162 #define STB_WEAK    2       /**< \brief Weak-linked symbol */
00163 /** @} */
00164
00165 /** \defgroup elf_symtype               Symbol types.
00166
00167     These are the values that can be set to say what kind of symbol a given
00168     symbol in an ELF file is. This is stored in the lower 4 bits of the info
00169     field in elf_sym_t.
00170
00171     @{
00172 */
00173 #define STT_NOTYPE  0       /**< \brief Symbol has no type */
00174 #define STT_OBJECT  1       /**< \brief Symbol is an object */
00175 #define STT_FUNC    2       /**< \brief Symbol is a function */
00176 #define STT_SECTION 3       /**< \brief Symbol is a section */
00177 #define STT_FILE    4       /**< \brief Symbol is a file name */
00178 /** @} */
00179
00180 /** \brief  Symbol table entry
00181
00182     This structure represents a single entry in a symbol table in an ELF file.
00183
00184     \headerfile kos/elf.h
00185 */
00186 struct elf_sym_t {
00187     uint32 name;        /**< \brief Index into file's string table */
00188     uint32 value;       /**< \brief Value of the symbol */
00189     uint32 size;        /**< \brief Size of the symbol */
00190     uint8 info;         /**< \brief Symbol type and binding */
00191     uint8 other;        /**< \brief 0. Holds no meaning. */
00192     uint16 shndx;       /**< \brief Section index */
00193 };
00194
00195 /** \brief  Retrieve the binding type for a symbol.
00196     \param  info            The info field of an elf_sym_t.
00197     \return                 The binding type of the symbol.
00198     \see                    elf_binding
00199 */
00200 #define ELF32_ST_BIND(info) ((info) » 4)
00201
00202 /** \brief  Retrieve the symbol type for a symbol.
00203     \param  info            The info field of an elf_sym_t.
00204     \return                 The symbol type of the symbol.
00205     \see                    elf_symtype
00206 */
00207 #define ELF32_ST_TYPE(info) ((info) & 0xf)
00208
00209 /** \brief  ELF Relocation entry (with explicit addend).
00210
00211     This structure represents an ELF relocation entry with an explicit addend.
00212     This structure is used on some architectures, whereas others use the
00213     elf_rel_t structure instead.
```

```
00214
00215     \headerfile kos/elf.h
00216 */
00217 struct elf_rela_t {
00218     uint32 offset;      /**< \brief Offset within section */
00219     uint32 info;        /**< \brief Symbol and type */
00220     int32 addend;       /**< \brief Constant addend for the symbol */
00221 };
00222
00223 /** \brief  ELF Relocation entry (without explicit addend).
00224
00225     This structure represents an ELF relocation entry without an explicit
00226     addend. This structure is used on some architectures, whereas others use the
00227     elf_rela_t structure instead.
00228
00229     \headerfile kos/elf.h
00230 */
00231 struct elf_rel_t {
00232     uint32      offset;     /**< \brief Offset within section */
00233     uint32      info;       /**< \brief Symbol and type */
00234 };
00235
00236 /** \defgroup elf_reltypes              ELF relocation types
00237
00238     These define the types of operations that can be done to calculate
00239     relocations within ELF files.
00240
00241     @{
00242 */
00243 #define R_SH_DIR32  1       /**< \brief SuperH: Rel = Symbol + Addend */
00244 #define R_386_32    1       /**< \brief x86: Rel = Symbol + Addend */
00245 #define R_386_PC32  2       /**< \brief x86: Rel = Symbol + Addend - Value */
00246 /** @} */
00247
00248 /** \brief  Retrieve the symbol index from a relocation entry.
00249     \param  i               The info field of an elf_rel_t or elf_rela_t.
00250     \return                 The symbol table index from that relocation entry.
00251 */
00252 #define ELF32_R_SYM(i) ((i) » 8)
00253
00254 /** \brief  Retrieve the relocation type from a relocation entry.
00255     \param  i               The info field of an elf_rel_t or an elf_rela_t.
00256     \return                 The relocation type of that relocation.
00257     \see                    elf_reltypes
00258 */
00259 #define ELF32_R_TYPE(i) ((uint8)(i))
00260
00261 struct klibrary;
00262
00263 /** \brief  Kernel-specific definition of a loaded ELF binary.
00264
00265     This structure represents the internal representation of a loaded ELF binary
00266     in KallistiOS (specifically as a dynamically loaded library).
00267
00268     \headerfile kos/elf.h
00269 */
00270 typedef struct elf_prog {
00271     void *data;             /**< \brief Pointer to program in memory */
00272     uint32 size;            /**< \brief Memory image size (rounded up to page size) */
00273
00274     /* Library exports */
00275     ptr_t lib_get_name;     /**< \brief Pointer to get_name() function */
00276     ptr_t lib_get_version;  /**< \brief Pointer to get_version() function */
00277     ptr_t lib_open;         /**< \brief Pointer to library's open function */
00278     ptr_t lib_close;        /**< \brief Pointer to library's close function */
00279
00280     char fn[256];           /**< \brief Filename of library */
00281 } elf_prog_t;
00282
00283 /** \brief  Load an ELF binary.
00284
00285     This function loads an ELF binary from the VFS and fills in an elf_prog_t
00286     for it.
00287
00288     \param  fn              The filename of the binary on the VFS.
00289     \param  shell           Unused?
00290     \param  out             Storage for the binary that will be loaded.
00291     \return                 0 on success, <0 on failure.
00292 */
00293 int elf_load(const char *fn, struct klibrary * shell, elf_prog_t * out);
00294
```

```
00295 /** \brief  Free a loaded ELF program.
00296
00297     This function cleans up an ELF binary that was loaded with elf_load().
00298
00299     \param  prog            The loaded binary to clean up.
00300 */
00301 void elf_free(elf_prog_t *prog);
00302
00303 __END_DECLS
00304
00305 #endif  /* __OS_ELF_H */
00306
```

9.40 include/kos/exports.h File Reference

Kernel exported symbols support.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <kos/nmmgr.h>
```

Data Structures

- struct export_sym_t

  A single export symbol.
- struct symtab_handler_t

  A symbol table "handler" for nmmgr.

Functions

- void export_init (void)

  Setup initial kernel exports.
- export_sym_t * export_lookup (const char *name)

  Look up a symbol by name.

9.40.1 Detailed Description

Kernel exported symbols support.

This file contains support related to dynamic linking of the kernel of KOS. The kernel (at compile time) produces a list of exported symbols, which can be looked through using the funtionality in this file.

Author

Megan Potter

9.40.2 Function Documentation

export_init()

```
void export_init (
            void  )
```

Setup initial kernel exports.

export_lookup()

```
export_sym_t * export_lookup (
            const char * name )
```

Look up a symbol by name.

Parameters

name	The symbol to look up

Returns

The export structure, or NULL on failure

9.41 exports.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/exports.h
00004    Copyright (C)2003 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/exports.h
00009     \brief  Kernel exported symbols support.
00010
00011     This file contains support related to dynamic linking of the kernel of KOS.
00012     The kernel (at compile time) produces a list of exported symbols, which can
00013     be looked through using the funtionality in this file.
00014
00015     \author Megan Potter
00016 */
00017
00018 #ifndef __KOS_EXPORTS_H
00019 #define __KOS_EXPORTS_H
00020
00021 #include <sys/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 #include <arch/types.h>
00025
00026 /** \brief  A single export symbol.
00027
00028     This structure holds a single symbol that has been exported from the kernel.
00029     These will be patched into loaded ELF binaries at load time.
00030
00031     \headerfile kos/exports.h
```

```
00032 */
00033 typedef struct export_sym {
00034     const char  * name;     /**< \brief The name of the symbol. */
00035     ptr_t       ptr;        /**< \brief A pointer to the symbol. */
00036 } export_sym_t;
00037
00038 /** \cond */
00039 /* These are the platform-independent exports */
00040 extern export_sym_t kernel_symtab[];
00041
00042 /* And these are the arch-specific exports */
00043 extern export_sym_t arch_symtab[];
00044 /** \endcond */
00045
00046 #ifndef __EXPORTS_FILE
00047 #include <kos/nmmgr.h>
00048
00049 /** \brief  A symbol table "handler" for nmmgr.
00050     \headerfile kos/exports.h
00051 */
00052 typedef struct symtab_handler {
00053     struct nmmgr_handler nmmgr;     /**< \brief Name manager handler header */
00054     export_sym_t         * table;   /**< \brief Location of the first entry */
00055 } symtab_handler_t;
00056 #endif
00057
00058 /** \brief  Setup initial kernel exports. */
00059 void export_init(void);
00060
00061 /** \brief  Look up a symbol by name.
00062     \param  name            The symbol to look up
00063     \return                 The export structure, or NULL on failure
00064 */
00065 export_sym_t * export_lookup(const char * name);
00066
00067 __END_DECLS
00068
00069 #endif  /* __KOS_EXPORTS_H */
00070
```

9.42 include/kos/fs.h File Reference

Virtual filesystem support.

```
#include <sys/cdefs.h>
#include <sys/types.h>
#include <kos/limits.h>
#include <time.h>
#include <sys/queue.h>
#include <stdarg.h>
#include <sys/stat.h>
#include <kos/nmmgr.h>
#include <sys/fcntl.h>
```

Data Structures

- struct dirent_t

  Directory entry.
- struct vfs_handler_t

  VFS handler interface.

Macros

- #define STAT_UNIQUE_NONE 0

  stat_t.unique: Constant to use denoting the file has no unique ID
- #define STAT_TYPE_NONE 0

  stat_t.type: Unknown / undefined / not relevant
- #define STAT_TYPE_FILE 1

  stat_t.type: Standard file
- #define STAT_TYPE_DIR 2

  stat_t.type: Standard directory
- #define STAT_TYPE_PIPE 3

  stat_t.type: A virtual device of some sort (pipe, socket, etc)
- #define STAT_TYPE_META 4

  stat_t.type: Meta data
- #define STAT_TYPE_SYMLINK 5

  stat_t.type: Symbolic link
- #define STAT_ATTR_NONE 0x00

  stat_t.attr: No attributes
- #define STAT_ATTR_R 0x01

  stat_t.attr: Read-capable
- #define STAT_ATTR_W 0x02

  stat_t.attr: Write-capable
- #define STAT_ATTR_RW (STAT_ATTR_R | STAT_ATTR_W)

  stat_t.attr: Read/Write capable
- #define FILEHND_INVALID ((file_t)-1)

  Invalid file handle constant (for open failure, etc)
- #define FD_SETSIZE 1024

  The number of distinct file descriptors that can be in use at a time.
- #define O_MODE_MASK 0x0f

  Mask for mode numbers.
- #define O_ASYNC 0x0200

  Open for asynchronous I/O.
- #define O_DIR 0x1000

  Open as directory.
- #define O_META 0x2000

  Open as metadata.
- #define SEEK_SET 0

  Set position to offset.
- #define SEEK_CUR 1

  Seek from current position.
- #define SEEK_END 2

  Seek from end of file.

Typedefs

- typedef int file_t

  File descriptor type.

Functions

- file_t fs_open (const char *fn, int mode)

  Open a file on the VFS.
- int fs_close (file_t hnd)

  Close an opened file.
- ssize_t fs_read (file_t hnd, void *buffer, size_t cnt)

  Read from an opened file.
- ssize_t fs_write (file_t hnd, const void *buffer, size_t cnt)

  Write to an opened file.
- off_t fs_seek (file_t hnd, off_t offset, int whence)

  Seek to a new position within a file.
- _off64_t fs_seek64 (file_t hnd, _off64_t offset, int whence)

  Seek to a new position within a file (64-bit offsets).
- off_t fs_tell (file_t hnd)

  Retrieve the position of the pointer within a file.
- _off64_t fs_tell64 (file_t hnd)

  Retrieve the position of the 64-bit pointer within a file.
- size_t fs_total (file_t hnd)

  Retrieve the length of an opened file.
- uint64 fs_total64 (file_t hnd)

  Retrieve the length of an opened file as a 64-bit integer.
- dirent_t * fs_readdir (file_t hnd)

  Read an entry from an opened directory.
- int fs_ioctl (file_t hnd, int cmd,...)

  Execute a device-specific command on a file descriptor.
- int fs_rename (const char *fn1, const char *fn2)

  Rename the specified file to the given filename.
- int fs_unlink (const char *fn)

  Delete the specified file.
- int fs_chdir (const char *fn)

  Change the current working directory of the current thread.
- void * fs_mmap (file_t hnd)

  Memory-map a previously opened file.
- int fs_complete (file_t fd, ssize_t *rv)

  Perform an I/O completion on the given file descriptor.
- int fs_mkdir (const char *fn)

  Create a directory.
- int fs_rmdir (const char *fn)

  Remove a directory by name.
- int fs_fcntl (file_t fd, int cmd,...)

  Manipulate file control flags.
- int fs_link (const char *path1, const char *path2)

  Create a hard link.
- int fs_symlink (const char *path1, const char *path2)

  Create a symbolic link.
- ssize_t fs_readlink (const char *path, char *buf, size_t bufsize)

Read the value of a symbolic link.

- int fs_stat (const char *path, struct stat *buf, int flag)

  Retrieve information about the specified path.
- int fs_rewinddir (file_t hnd)

  Rewind a directory to the start.
- int fs_fstat (file_t hnd, struct stat *buf)

  Retrieve information about an opened file.
- file_t fs_dup (file_t oldfd)

  Duplicate a file descriptor.
- file_t fs_dup2 (file_t oldfd, file_t newfd)

  Duplicate a file descriptor onto the specified descriptor.
- file_t fs_open_handle (vfs_handler_t *vfs, void *hnd)

  Create a "transient" file descriptor.
- vfs_handler_t * fs_get_handler (file_t fd)

  Retrieve the VFS Handler for a file descriptor.
- void * fs_get_handle (file_t fd)

  Retrieve the internal handle for a file descriptor.
- const char * fs_getwd (void)

  Get the current working directory of the running thread.
- ssize_t fs_copy (const char *src, const char *dst)

  Copy a file.
- ssize_t fs_load (const char *src, void **out_ptr)

  Open and read a whole file into RAM.
- ssize_t fs_path_append (char *dst, const char *src, size_t len)

  Append a path component to a string.
- int fs_init (void)

  Initialize the virtual filesystem.
- void fs_shutdown (void)

  Shut down the virtual filesystem.

9.42.1 Detailed Description

Virtual filesystem support.

This file contains the interface to the virtual filesystem (VFS) of KOS. The functions defined in this file make up the base of the filesystem operations that can be performed by programs. The functions in here are abstracted by various other layers in libc, and shouldn't be necessarily used (for portability reasons). However, if you want only to interact with KOS in your programs, feel free to use them to your heart's content!

Author

Megan Potter

Lawrence Sebald

9.42.2 Macro Definition Documentation

FD_SETSIZE

```
#define FD_SETSIZE 1024
```

The number of distinct file descriptors that can be in use at a time.

FILEHND_INVALID

```
#define FILEHND_INVALID ((file_t)-1)
```

Invalid file handle constant (for open failure, etc)

STAT_ATTR_NONE

```
#define STAT_ATTR_NONE 0x00
```

stat_t.attr: No attributes

STAT_ATTR_R

```
#define STAT_ATTR_R 0x01
```

stat_t.attr: Read-capable

STAT_ATTR_RW

```
#define STAT_ATTR_RW (STAT_ATTR_R | STAT_ATTR_W)
```

stat_t.attr: Read/Write capable

STAT_ATTR_W

```
#define STAT_ATTR_W 0x02
```

stat_t.attr: Write-capable

STAT_TYPE_DIR

```
#define STAT_TYPE_DIR 2
```

stat_t.type: Standard directory

STAT_TYPE_FILE

```
#define STAT_TYPE_FILE 1
```

stat_t.type: Standard file

STAT_TYPE_META

```
#define STAT_TYPE_META 4
```

stat_t.type: Meta data

STAT_TYPE_NONE

```
#define STAT_TYPE_NONE 0
```

stat_t.type: Unknown / undefined / not relevant

STAT_TYPE_PIPE

```
#define STAT_TYPE_PIPE 3
```

stat_t.type: A virtual device of some sort (pipe, socket, etc)

STAT_TYPE_SYMLINK

```
#define STAT_TYPE_SYMLINK 5
```

stat_t.type: Symbolic link

STAT_UNIQUE_NONE

```
#define STAT_UNIQUE_NONE 0
```

stat_t.unique: Constant to use denoting the file has no unique ID

<

9.42.3 Typedef Documentation

file_t

```
typedef int file_t
```

File descriptor type.

9.42.4 Function Documentation

fs_chdir()

```
int fs_chdir (
            const char * fn )
```

Change the current working directory of the current thread.

This function changes the current working directory for the current thread. Any relative paths passed into file-related functions will be relative to the path that is changed to.

Parameters

fn	The path to set as the current working directory.

Returns

0 on success, -1 on failure.

fs_close()

```
int fs_close (
              file_t hnd )
```

Close an opened file.

This function closes the specified file descriptor, releasing all resources associated with the descriptor.

Parameters

hnd	The file descriptor to close.

Returns

0 for success, -1 for error

fs_complete()

```
int fs_complete (
              file_t fd,
              ssize_t * rv )
```

Perform an I/O completion on the given file descriptor.

This function is used with asynchronous I/O to perform an I/O completion on the given file descriptor.

Parameters

fd	The descriptor to complete I/O on.
rv	A buffer to store the size of the I/O in.

Returns

0 on success, -1 on failure.

Note

Most of the filesystems in KallistiOS do not support this operation. If you attempt to use this function on a filesystem that does not support it, the function will return -1 and set errno to EINVAL.

fs_copy()

```
ssize_t fs_copy (
            const char * src,
            const char * dst )
```

Copy a file.

This function copies the file at src to dst on the filesystem.

Parameters

src	The filename to copy from.
dst	The filename to copy to.

Returns

The number of bytes copied successfully.

fs_dup()

```
file_t fs_dup (
            file_t oldfd )
```

Duplicate a file descriptor.

This function duplicates the specified file descriptor, returning a new file descriptor that can be used to access the file. This is equivalent to the standard POSIX function dup().

Parameters

oldfd	The old file descriptor to duplicate.

Returns

The new file descriptor on success, -1 on failure.

fs_dup2()

```
file_t fs_dup2 (
```

```
            file_t oldfd,
            file_t newfd )
```

Duplicate a file descriptor onto the specified descriptor.

This function duplicates the specified file descriptor onto the other file descriptor provided. If the newfd parameter represents an open file, that file will be closed before the old descriptor is duplicated onto it. This is equivalent to the standard POSIX function dup2().

Parameters

oldfd	The old file descriptor to duplicate.
newfd	The descriptor to copy into.

Returns

The new file descriptor on success, -1 on failure.

fs_fcntl()

```
int fs_fcntl (
            file_t fd,
            int cmd,
             ...  )
```

Manipulate file control flags.

This function implements the standard C fcntl function.

Parameters

fd	The file descriptor to use.
cmd	The command to run.
...	Arguments for the command specified.

Returns

-1 on error (generally).

fs_fstat()

```
int fs_fstat (
            file_t hnd,
            struct stat * buf )
```

Retrieve information about an opened file.

This function retrieves status information on the given file descriptor, which must correspond to an already opened file.

Parameters

hnd	The file descriptor to retrieve information about.
buf	The buffer to store stat information in.

Returns

0 on success, -1 on failure.

Note

Some filesystems may not support this function. If a filesystem doesn't support it, errno will be set to ENOSYS and -1 will be returned.

fs_get_handle()

```
void * fs_get_handle (
            file_t fd )
```

Retrieve the internal handle for a file descriptor.

This function retrieves the internal file handle data of the specified file descriptor. There is generally no reason to call this function in user code, as it is meant for use internally.

Parameters

fd	The file descriptor to retrieve the handler for.

Returns

The internal handle for the file descriptor.

fs_get_handler()

```
vfs_handler_t * fs_get_handler (
            file_t fd )
```

Retrieve the VFS Handler for a file descriptor.

This function retrieves the Handler structure for the VFS of the specified file descriptor. There is generally no reason to call this function in user code, as it is meant for use internally.

Parameters

fd	The file descriptor to retrieve the handler for.

Returns

The VFS' handler structure.

fs_getwd()

```
const char * fs_getwd (
             void  )
```

Get the current working directory of the running thread.

Returns

The current working directory.

fs_init()

```
int fs_init (
             void  )
```

Initialize the virtual filesystem.

This is normally done for you by default when KOS starts. In general, there should be no reason for you to call this function.

Return values

0	On success.

fs_ioctl()

```
int fs_ioctl (
             file_t hnd,
             int cmd,
              ...  )
```

Execute a device-specific command on a file descriptor.

The types and formats of the commands are device/filesystem specific, and are not documented here. Each filesystem may define any commands that are specific to it with its implementation of this function.

Parameters

hnd	The file descriptor to use.
cmd	The command to run.
...	Arguments for the command specified.

Returns

-1 on error.

fs_link()

```
int fs_link (
            const char * path1,
            const char * path2 )
```

Create a hard link.

This function implements the POSIX function link(), which creates a hard link for an existing file.

Parameters

path1	An existing file to create a new link to.
path2	The pathname of the new link to be created.

Returns

0 on success, -1 on failure.

Note

Most filesystems in KallistiOS do not support hard links. If you call this function on a filesystem that does not support hard links, the function will return -1 and set errno to EMLINK.

fs_load()

```
ssize_t fs_load (
            const char * src,
            void ** out_ptr )
```

Open and read a whole file into RAM.

This function opens the specified file, reads it into memory (allocating the necessary space with malloc), and closes the file. The caller is responsible for freeing the memory when they are done with it.

Parameters

src	The filename to open and read.
out_ptr	A pointer to the buffer on success, NULL otherwise.

Returns

The size of the file on success, -1 otherwise.

fs_mkdir()

```
int fs_mkdir (
              const char * fn )
```

Create a directory.

This function creates the specified directory, if possible.

Parameters

fn	The path of the directory to create.

Returns

0 on success, -1 on failure.

fs_mmap()

```
void * fs_mmap (
              file_t hnd )
```

Memory-map a previously opened file.

This file "maps" the opened file into memory, reading the whole file into a buffer, and returning that buffer. The returned buffer should not be freed, as it will be freed when the file is closed. Bytes written into the buffer, up to the original length of the file, will be written back to the file when it is closed, assuming that the file is opened for writing.

Parameters

hnd	The descriptor to memory map.

Returns

The memory mapped buffer, or NULL on failure.

Note

Some of the filesystems in KallistiOS do not support this operation. If you attempt to use this function on a filesystem that does not support it, the function will return NULL and set errno to EINVAL.

fs_open()

```
file_t fs_open (
            const char * fn,
            int mode )
```

Open a file on the VFS.

This function opens the specified file, returning a new file descriptor to access the file.

Parameters

fn	The path to open.
mode	The mode to use with opening the file. This may include the standard open modes (O_RDONLY, O_WRONLY, etc), as well as values from the File open modes list. Multiple values can be ORed together.

Returns

The new file descriptor on success, -1 on error.

fs_open_handle()

```
file_t fs_open_handle (
            vfs_handler_t * vfs,
            void * hnd )
```

Create a "transient" file descriptor.

This function creates and opens a new file descriptor that isn't associated directly with a file on the filesystem. This is used internally to actually open files, and should (in general) not be called by user code. Effectively, if you're trying to implement your own filesystem handler in your code, you may need this function, otherwise you should just ignore it.

Parameters

vfs	The VFS handler structure to use for the file.
hnd	Internal handle data for the file.

Returns

The opened descriptor on success, -1 on failure.

fs_path_append()

```
ssize_t fs_path_append (
            char * dst,
```

```
            const char * src,
            size_t len )
```

Append a path component to a string.

This function acts mostly like the function strncat(), with a few slight differences. First, if the destination string doesn't end in a '/' character, this function will add it. Second, it returns the length of the resulting string, including the NUL terminator. Finally, no modification of the destination string will occur if there isn't enough space left in the string to do so.

Parameters

dst	The string to modify.
src	The path component to append.
len	The length allocated for dst.

Returns

The length of the new string (including the NUL terminator) on success, -1 otherwise.

Error Conditions:

EFAULT - src or dst is a NULL pointer
EINVAL - len is zero
ENAMETOOLONG - the resulting path would be longer than len bytes

fs_read()

```
ssize_t fs_read (
            file_t hnd,
            void * buffer,
            size_t cnt )
```

Read from an opened file.

This function reads into the specified buffer from the file at its current file pointer.

Parameters

hnd	The file descriptor to read from.
buffer	The buffer to read into.
cnt	The size of the buffer (or the number of bytes requested).

Returns

The number of bytes read, or -1 on error. Note that this may not be the full number of bytes requested.

fs_readdir()

```
dirent_t * fs_readdir (
            file_t hnd )
```

Read an entry from an opened directory.

This function reads the next entry from the directory specified by the given file descriptor.

Parameters

hnd	The opened directory's file descriptor.

Returns

The next entry, or NULL on failure.

fs_readlink()

```
ssize_t fs_readlink (
            const char * path,
            char * buf,
            size_t bufsize )
```

Read the value of a symbolic link.

This function implements the POSIX function readlink(), which simply reads the value of the symbolic link at the end of a path. This does not resolve any internal links and it does not canonicalize the path either.

Parameters

path	The symbolic link to read.
buf	The buffer to place the link's contents in.
bufsize	The number of bytes allocated to buf.

Returns

-1 on failure, the number of bytes placed into buf on success. If the return value is equal to bufsize, you may not have the whole link – provide a larger buffer and try again.

Note

Most filesystems in KallistiOS do not support symbolic links. Filesystems that do not support symlinks will simply set errno to ENOSYS and return -1.

fs_rename()

```
int fs_rename (
            const char * fn1,
            const char * fn2 )
```

Rename the specified file to the given filename.

This function renames the file specified by the first argument to the second argument. The two paths should be on the same filesystem.

Parameters

fn1	The existing file to rename.
fn2	The new filename to rename to.

Returns

0 on success, -1 on failure.

fs_rewinddir()

```
int fs_rewinddir (
            file_t hnd )
```

Rewind a directory to the start.

This function rewinds the position of a directory stream to the beginning of the directory.

Parameters

hnd	The opened directory's file descriptor.

Returns

0 on success, -1 on failure.

Note

Some filesystems may not support this function. If a filesystem doesn't support it, errno will be set to ENOSYS and -1 will be returned.

fs_rmdir()

```
int fs_rmdir (
            const char * fn )
```

Remove a directory by name.

This function removes the specified directory. The directory shall only be removed if it is empty.

Parameters

fn	The path of the directory to remove.

Returns

0 on success, -1 on failure.

fs_seek()

```
off_t fs_seek (
            file_t hnd,
            off_t offset,
            int whence )
```

Seek to a new position within a file.

This function moves the file pointer to the specified position within the file (the base of this position is determined by the whence parameter).

Parameters

hnd	The file descriptor to move the pointer for.
offset	The offset in bytes from the specified base.
whence	The base of the pointer move. This should be one of the Seek modes values.

Returns

The new position of the file pointer.

fs_seek64()

```
_off64_t fs_seek64 (
            file_t hnd,
            _off64_t offset,
            int whence )
```

Seek to a new position within a file (64-bit offsets).

This function moves the file pointer to the specified position within the file (the base of this position is determined by the whence parameter).

Parameters

hnd	The file descriptor to move the pointer for.
offset	The offset in bytes from the specified base.
whence	The base of the pointer move. This should be one of the Seek modes values.

Returns

The new position of the file pointer.

fs_shutdown()

```
void fs_shutdown (
            void  )
```

Shut down the virtual filesystem.

This is done for you by the normal shutdown procedure of KOS. There should not really be any reason for you to call this function yourself.

fs_stat()

```
int fs_stat (
            const char * path,
            struct stat * buf,
            int flag )
```

Retrieve information about the specified path.

This function retrieves status information on the given path. This function now returns the normal POSIX-style struct stat, rather than the old KOS stat_t structure. In addition, you can specify whether or not this function should resolve symbolic links on filesystems that support symlinks.

Parameters

path	The path to retrieve information about.
buf	The buffer to store stat information in.
flag	Specifies whether or not to resolve a symbolic link. If you don't want to resolve any symbolic links at the end of the path, pass AT_SYMLINK_NOFOLLOW, otherwise pass 0.

Returns

0 on success, -1 on failure.

fs_symlink()

```
int fs_symlink (
            const char * path1,
            const char * path2 )
```

Create a symbolic link.

This function implements the POSIX function symlink(), which creates a symbolic link on the filesystem. Symbolic links are not required to point to an existing file (per POSIX) and may result in circular links if care is not taken. For now, symbolic links cannot cross filesystem boundaries in KOS.

Parameters

path1	The content of the link (i.e, what to point at).
path2	The pathname of the new link to be created.

Returns

0 on success, -1 on failure.

Note

Most filesystems in KallistiOS do not support symbolic links. Filesystems that do not support symlinks will simply set errno to ENOSYS and return -1.

fs_tell()

```
off_t fs_tell (
             file_t hnd )
```

Retrieve the position of the pointer within a file.

This function retrieves the current location of the file pointer within an opened file. This is an offset in bytes from the start of the file.

Parameters

hnd	The file descriptor to retrieve the pointer from.

Returns

The offset within the file for the pointer.

fs_tell64()

```
_off64_t fs_tell64 (
             file_t hnd )
```

Retrieve the position of the 64-bit pointer within a file.

This function retrieves the current location of the file pointer within an opened file. This is an offset in bytes from the start of the file.

Parameters

hnd	The file descriptor to retrieve the pointer from.

Returns

The offset within the file for the pointer.

fs_total()

```
size_t fs_total (
            file_t hnd )
```

Retrieve the length of an opened file.

This file retrieves the length of the file associated with the given file descriptor.

Parameters

hnd	The file descriptor to retrieve the size from.

Returns

The length of the file on success, -1 on failure.

Note

size_t is unsigned, so the error return value is not less than 0.

fs_total64()

```
uint64 fs_total64 (
            file_t hnd )
```

Retrieve the length of an opened file as a 64-bit integer.

This file retrieves the length of the file associated with the given file descriptor.

Parameters

hnd	The file descriptor to retrieve the size from.

Returns

The length of the file on success, -1 on failure.

Note

uint64 is unsigned, so the error return value is not less than 0.

fs_unlink()

```
int fs_unlink (
            const char * fn )
```

Delete the specified file.

This function deletes the specified file from the filesystem. This should only be used for files, not for directories. For directories, use fs_rmdir() instead of this function.

Parameters

fn	The path to remove.

Returns

0 on success, -1 on failure.

fs_write()

```
ssize_t fs_write (
            file_t hnd,
            const void * buffer,
            size_t cnt )
```

Write to an opened file.

This function writes the specfied buffer into the file at the current file pointer.

Parameters

hnd	The file descriptor to write into.
buffer	The data to write into the file.
cnt	The size of the buffer, in bytes.

Returns

The number of bytes written, or -1 on failure. Note that the number of bytes written may be less than what was requested.

9.43 fs.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/fs.h
00004    Copyright (C) 2000, 2001, 2002, 2003 Megan Potter
```

```
00005    Copyright (C) 2012, 2013, 2014, 2016 Lawrence Sebald
00006
00007 */
00008
00009 #ifndef __KOS_FS_H
00010 #define __KOS_FS_H
00011
00012 #include <sys/cdefs.h>
00013 __BEGIN_DECLS
00014
00015 #include <sys/types.h>
00016 #include <kos/limits.h>
00017 #include <time.h>
00018 #include <sys/queue.h>
00019 #include <stdarg.h>
00020 #include <sys/stat.h>
00021
00022 #include <kos/nmmgr.h>
00023
00024 /** \file   kos/fs.h
00025     \brief  Virtual filesystem support.
00026
00027     This file contains the interface to the virtual filesystem (VFS) of KOS. The
00028     functions defined in this file make up the base of the filesystem operations
00029     that can be performed by programs. The functions in here are abstracted by
00030     various other layers in libc, and shouldn't be necessarily used (for
00031     portability reasons). However, if you want only to interact with KOS in your
00032     programs, feel free to use them to your heart's content!
00033
00034     \author Megan Potter
00035     \author Lawrence Sebald
00036 */
00037
00038 /** \brief  Directory entry.
00039
00040     All VFS handlers must conform to this interface in their directory entries.
00041
00042     \headerfile kos/fs.h
00043 */
00044 typedef struct kos_dirent {
00045     int size;               /**< \brief Size of the file in bytes. */
00046     char name[NAME_MAX];  /**< \brief Name of the file. */
00047     time_t time;            /**< \brief Last access/mod/change time (depends on VFS) */
00048     uint32 attr;            /**< \brief Attributes of the file. */
00049 } dirent_t;
00050
00051 /* Forward declaration */
00052 struct vfs_handler;
00053
00054 /* stat_t.unique */
00055 /**< \brief stat_t.unique: Constant to use denoting the file has no unique ID */
00056 #define STAT_UNIQUE_NONE    0
00057
00058 /* stat_t.type */
00059 /** \brief stat_t.type: Unknown / undefined / not relevant */
00060 #define STAT_TYPE_NONE      0
00061
00062 /** \brief stat_t.type: Standard file */
00063 #define STAT_TYPE_FILE      1
00064
00065 /** \brief stat_t.type: Standard directory */
00066 #define STAT_TYPE_DIR       2
00067
00068 /** \brief stat_t.type: A virtual device of some sort (pipe, socket, etc) */
00069 #define STAT_TYPE_PIPE      3
00070
00071 /** \brief stat_t.type: Meta data */
00072 #define STAT_TYPE_META      4
00073
00074 /** \brief stat_t.type: Symbolic link */
00075 #define STAT_TYPE_SYMLINK   5
00076
00077 /* stat_t.attr */
00078 #define STAT_ATTR_NONE  0x00    /**< \brief stat_t.attr: No attributes */
00079 #define STAT_ATTR_R     0x01    /**< \brief stat_t.attr: Read-capable */
00080 #define STAT_ATTR_W     0x02    /**< \brief stat_t.attr: Write-capable */
00081
00082 /** \brief stat_t.attr: Read/Write capable */
00083 #define STAT_ATTR_RW    (STAT_ATTR_R | STAT_ATTR_W)
00084
00085 /** \brief  File descriptor type */
```

```
00086 typedef int file_t;
00087
00088 /** \brief  Invalid file handle constant (for open failure, etc) */
00089 #define FILEHND_INVALID ((file_t)-1)
00090
00091 /** \brief  VFS handler interface.
00092
00093     All VFS handlers must implement this interface.
00094
00095     \headerfile kos/fs.h
00096 */
00097 typedef struct vfs_handler {
00098     /** \brief Name manager handler header */
00099     nmmgr_handler_t nmmgr;
00100
00101     /* Some VFS-specific pieces */
00102     /** \brief Allow VFS cacheing; 0=no, 1=yes */
00103     int cache;
00104     /** \brief Pointer to private data for the handler */
00105     void *privdata;
00106
00107     /** \brief Open a file on the given VFS; return a unique identifier */
00108     void *(*open)(struct vfs_handler *vfs, const char *fn, int mode);
00109
00110     /** \brief Close a previously opened file */
00111     int (*close)(void *hnd);
00112
00113     /** \brief Read from a previously opened file */
00114     ssize_t (*read)(void *hnd, void *buffer, size_t cnt);
00115
00116     /** \brief Write to a previously opened file */
00117     ssize_t (*write)(void *hnd, const void *buffer, size_t cnt);
00118
00119     /** \brief Seek in a previously opened file */
00120     off_t (*seek)(void *hnd, off_t offset, int whence);
00121
00122     /** \brief Return the current position in a previously opened file */
00123     off_t (*tell)(void *hnd);
00124
00125     /** \brief Return the total size of a previously opened file */
00126     size_t (*total)(void *hnd);
00127
00128     /** \brief Read the next directory entry in a directory opened with O_DIR */
00129     dirent_t *(*readdir)(void *hnd);
00130
00131     /** \brief Execute a device-specific call on a previously opened file */
00132     int (*ioctl)(void *hnd, int cmd, va_list ap);
00133
00134     /** \brief Rename/move a file on the given VFS */
00135     int (*rename)(struct vfs_handler *vfs, const char *fn1, const char *fn2);
00136
00137     /** \brief Delete a file from the given VFS */
00138     int (*unlink)(struct vfs_handler *vfs, const char *fn);
00139
00140     /** \brief "Memory map" a previously opened file */
00141     void *(*mmap)(void *fd);
00142
00143     /** \brief Perform an I/O completion (async I/O) for a previously opened
00144                file */
00145     int (*complete)(void *fd, ssize_t *rv);
00146
00147     /** \brief Get status information on a file on the given VFS
00148         \note  path will not be passed through realpath() before calling the
00149                filesystem-level function. It is also important to not call
00150                realpath() in any implementation of this function as it is
00151                possible that realpath() will call this function. */
00152     int (*stat)(struct vfs_handler *vfs, const char *path, struct stat *buf,
00153                 int flag);
00154
00155     /** \brief Make a directory on the given VFS */
00156     int (*mkdir)(struct vfs_handler *vfs, const char *fn);
00157
00158     /** \brief Remove a directory from the given VFS */
00159     int (*rmdir)(struct vfs_handler *vfs, const char *fn);
00160
00161     /** \brief Manipulate file control flags on the given file */
00162     int (*fcntl)(void *fd, int cmd, va_list ap);
00163
00164     /** \brief Check if an event is pending on the given file */
00165     short (*poll)(void *fd, short events);
00166
```

```
00167     /** \brief Create a hard link */
00168     int (*link)(struct vfs_handler *vfs, const char *path1, const char *path2);
00169
00170     /** \brief Create a symbolic link */
00171     int (*symlink)(struct vfs_handler *vfs, const char *path1,
00172                    const char *path2);
00173
00174     /* 64-bit file access functions. Generally, you should only define one of
00175        the 64-bit or 32-bit versions of these functions. */
00176
00177     /** \brief Seek in a previously opened file (64-bit offsets) */
00178     _off64_t (*seek64)(void *hnd, _off64_t offset, int whence);
00179
00180     /** \brief Return the current position in an opened file (64-bit offset) */
00181     _off64_t (*tell64)(void *hnd);
00182
00183     /** \brief Return the size of an opened file as a 64-bit integer */
00184     uint64 (*total64)(void *hnd);
00185
00186     /** \brief Read the value of a symbolic link
00187         \note  path will not be passed through realpath() before calling the
00188                filesystem-level function. It is also important to not call
00189                realpath() in any implementation of this function as it is
00190                possible that realpath() will call this function. */
00191     ssize_t (*readlink)(struct vfs_handler *vfs, const char *path, char *buf,
00192                         size_t bufsize);
00193
00194     /** \brief Rewind a directory stream to the start */
00195     int (*rewinddir)(void *hnd);
00196
00197     /** \brief Get status information on an already opened file. */
00198     int (*fstat)(void *hnd, struct stat *st);
00199 } vfs_handler_t;
00200
00201 /** \brief  The number of distinct file descriptors that can be in use at a
00202             time.
00203 */
00204 #define FD_SETSIZE  1024
00205
00206 /** \cond */
00207 /* This is the private struct that will be used as raw file handles
00208    underlying descriptors. */
00209 struct fs_hnd;
00210
00211 /* The kernel-wide file descriptor table. These will reference to open files. */
00212 extern struct fs_hnd *fd_table[FD_SETSIZE];
00213 /** \endcond */
00214
00215 /* Open modes */
00216 #include <sys/fcntl.h>
00217 /** \defgroup open_modes            File open modes
00218
00219     @{
00220 */
00221 #define O_MODE_MASK 0x0f        /**< \brief Mask for mode numbers */
00222 //#define O_TRUNC       0x0100      /* Truncate */
00223 #define O_ASYNC     0x0200      /**< \brief Open for asynchronous I/O */
00224 //#define O_NONBLOCK    0x0400      /* Open for non-blocking I/O */
00225 #define O_DIR       0x1000      /**< \brief Open as directory */
00226 #define O_META      0x2000      /**< \brief Open as metadata */
00227 /** @} */
00228
00229 /** \defgroup seek_modes            Seek modes
00230
00231     These are the values you can pass for the whence parameter to fs_seek().
00232
00233     @{
00234 */
00235 #define SEEK_SET    0           /**< \brief Set position to offset. */
00236 #define SEEK_CUR    1           /**< \brief Seek from current position. */
00237 #define SEEK_END    2           /**< \brief Seek from end of file. */
00238 /** @} */
00239
00240 /* Standard file descriptor functions */
00241 /** \brief  Open a file on the VFS.
00242
00243     This function opens the specified file, returning a new file descriptor to
00244     access the file.
00245
00246     \param  fn              The path to open.
00247     \param  mode            The mode to use with opening the file. This may
```

```
00248                           include the standard open modes (O_RDONLY, O_WRONLY,
00249                           etc), as well as values from the \ref open_modes
00250                           list. Multiple values can be ORed together.
00251     \return               The new file descriptor on success, -1 on error.
00252 */
00253 file_t fs_open(const char *fn, int mode);
00254
00255 /** \brief  Close an opened file.
00256
00257     This function closes the specified file descriptor, releasing all resources
00258     associated with the descriptor.
00259
00260     \param  hnd             The file descriptor to close.
00261     \return                 0 for success, -1 for error
00262 */
00263 int fs_close(file_t hnd);
00264
00265 /** \brief  Read from an opened file.
00266
00267     This function reads into the specified buffer from the file at its current
00268     file pointer.
00269
00270     \param  hnd             The file descriptor to read from.
00271     \param  buffer          The buffer to read into.
00272     \param  cnt             The size of the buffer (or the number of bytes
00273                             requested).
00274     \return                 The number of bytes read, or -1 on error. Note that
00275                             this may not be the full number of bytes requested.
00276 */
00277 ssize_t fs_read(file_t hnd, void *buffer, size_t cnt);
00278
00279 /** \brief  Write to an opened file.
00280
00281     This function writes the specfied buffer into the file at the current file
00282     pointer.
00283
00284     \param  hnd             The file descriptor to write into.
00285     \param  buffer          The data to write into the file.
00286     \param  cnt             The size of the buffer, in bytes.
00287     \return                 The number of bytes written, or -1 on failure. Note
00288                             that the number of bytes written may be less than
00289                             what was requested.
00290 */
00291 ssize_t fs_write(file_t hnd, const void *buffer, size_t cnt);
00292
00293 /** \brief  Seek to a new position within a file.
00294
00295     This function moves the file pointer to the specified position within the
00296     file (the base of this position is determined by the whence parameter).
00297
00298     \param  hnd             The file descriptor to move the pointer for.
00299     \param  offset          The offset in bytes from the specified base.
00300     \param  whence          The base of the pointer move. This should be one of
00301                             the \ref seek_modes values.
00302     \return                 The new position of the file pointer.
00303 */
00304 off_t fs_seek(file_t hnd, off_t offset, int whence);
00305
00306 /** \brief  Seek to a new position within a file (64-bit offsets).
00307
00308     This function moves the file pointer to the specified position within the
00309     file (the base of this position is determined by the whence parameter).
00310
00311     \param  hnd             The file descriptor to move the pointer for.
00312     \param  offset          The offset in bytes from the specified base.
00313     \param  whence          The base of the pointer move. This should be one of
00314                             the \ref seek_modes values.
00315     \return                 The new position of the file pointer.
00316 */
00317 _off64_t fs_seek64(file_t hnd, _off64_t offset, int whence);
00318
00319 /** \brief  Retrieve the position of the pointer within a file.
00320
00321     This function retrieves the current location of the file pointer within an
00322     opened file. This is an offset in bytes from the start of the file.
00323
00324     \param  hnd             The file descriptor to retrieve the pointer from.
00325     \return                 The offset within the file for the pointer.
00326 */
00327 off_t fs_tell(file_t hnd);
00328
```

```
00329 /** \brief  Retrieve the position of the 64-bit pointer within a file.
00330
00331     This function retrieves the current location of the file pointer within an
00332     opened file. This is an offset in bytes from the start of the file.
00333
00334     \param  hnd             The file descriptor to retrieve the pointer from.
00335     \return                 The offset within the file for the pointer.
00336 */
00337 _off64_t fs_tell64(file_t hnd);
00338
00339 /** \brief  Retrieve the length of an opened file.
00340
00341     This file retrieves the length of the file associated with the given file
00342     descriptor.
00343
00344     \param  hnd             The file descriptor to retrieve the size from.
00345     \return                 The length of the file on success, -1 on failure.
00346     \note                   size_t is unsigned, so the error return value is not
00347                             less than 0.
00348 */
00349 size_t fs_total(file_t hnd);
00350
00351 /** \brief  Retrieve the length of an opened file as a 64-bit integer.
00352
00353     This file retrieves the length of the file associated with the given file
00354     descriptor.
00355
00356     \param  hnd             The file descriptor to retrieve the size from.
00357     \return                 The length of the file on success, -1 on failure.
00358     \note                   uint64 is unsigned, so the error return value is not
00359                             less than 0.
00360 */
00361 uint64 fs_total64(file_t hnd);
00362
00363
00364 /** \brief  Read an entry from an opened directory.
00365
00366     This function reads the next entry from the directory specified by the given
00367     file descriptor.
00368
00369     \param  hnd             The opened directory's file descriptor.
00370     \return                 The next entry, or NULL on failure.
00371 */
00372 dirent_t *fs_readdir(file_t hnd);
00373
00374 /** \brief  Execute a device-specific command on a file descriptor.
00375
00376     The types and formats of the commands are device/filesystem specific, and
00377     are not documented here. Each filesystem may define any commands that are
00378     specific to it with its implementation of this function.
00379
00380     \param  hnd             The file descriptor to use.
00381     \param  cmd             The command to run.
00382     \param  ...             Arguments for the command specified.
00383     \return                 -1 on error.
00384 */
00385 int fs_ioctl(file_t hnd, int cmd, ...);
00386
00387 /** \brief  Rename the specified file to the given filename.
00388
00389     This function renames the file specified by the first argument to the second
00390     argument. The two paths should be on the same filesystem.
00391
00392     \param  fn1             The existing file to rename.
00393     \param  fn2             The new filename to rename to.
00394     \return                 0 on success, -1 on failure.
00395 */
00396 int fs_rename(const char *fn1, const char *fn2);
00397
00398 /** \brief  Delete the specified file.
00399
00400     This function deletes the specified file from the filesystem. This should
00401     only be used for files, not for directories. For directories, use fs_rmdir()
00402     instead of this function.
00403
00404     \param  fn              The path to remove.
00405     \return                 0 on success, -1 on failure.
00406 */
00407 int fs_unlink(const char *fn);
00408
00409 /** \brief  Change the current working directory of the current thread.
```

```
00410
00411     This function changes the current working directory for the current thread.
00412     Any relative paths passed into file-related functions will be relative to
00413     the path that is changed to.
00414
00415     \param  fn              The path to set as the current working directory.
00416     \return                 0 on success, -1 on failure.
00417 */
00418 int fs_chdir(const char *fn);
00419
00420 /** \brief  Memory-map a previously opened file.
00421
00422     This file "maps" the opened file into memory, reading the whole file into a
00423     buffer, and returning that buffer. The returned buffer should not be freed,
00424     as it will be freed when the file is closed. Bytes written into the buffer,
00425     up to the original length of the file, will be written back to the file when
00426     it is closed, assuming that the file is opened for writing.
00427
00428     \param  hnd             The descriptor to memory map.
00429     \return                 The memory mapped buffer, or NULL on failure.
00430
00431     \note                   Some of the filesystems in KallistiOS do not support
00432                             this operation. If you attempt to use this function
00433                             on a filesystem that does not support it, the
00434                             function will return NULL and set errno to EINVAL.
00435 */
00436 void *fs_mmap(file_t hnd);
00437
00438 /** \brief  Perform an I/O completion on the given file descriptor.
00439
00440     This function is used with asynchronous I/O to perform an I/O completion on
00441     the given file descriptor.
00442
00443     \param  fd              The descriptor to complete I/O on.
00444     \param  rv              A buffer to store the size of the I/O in.
00445     \return                 0 on success, -1 on failure.
00446
00447     \note                   Most of the filesystems in KallistiOS do not support
00448                             this operation. If you attempt to use this function
00449                             on a filesystem that does not support it, the
00450                             function will return -1 and set errno to EINVAL.
00451 */
00452 int fs_complete(file_t fd, ssize_t *rv);
00453
00454 /** \brief  Create a directory.
00455
00456     This function creates the specified directory, if possible.
00457
00458     \param  fn              The path of the directory to create.
00459     \return                 0 on success, -1 on failure.
00460 */
00461 int fs_mkdir(const char *fn);
00462
00463 /** \brief  Remove a directory by name.
00464
00465     This function removes the specified directory. The directory shall only be
00466     removed if it is empty.
00467
00468     \param  fn              The path of the directory to remove.
00469     \return                 0 on success, -1 on failure.
00470 */
00471 int fs_rmdir(const char *fn);
00472
00473 /** \brief  Manipulate file control flags.
00474
00475     This function implements the standard C fcntl function.
00476
00477     \param  fd              The file descriptor to use.
00478     \param  cmd             The command to run.
00479     \param  ...             Arguments for the command specified.
00480     \return                 -1 on error (generally).
00481 */
00482 int fs_fcntl(file_t fd, int cmd, ...);
00483
00484 /** \brief  Create a hard link.
00485
00486     This function implements the POSIX function link(), which creates a hard
00487     link for an existing file.
00488
00489     \param  path1           An existing file to create a new link to.
00490     \param  path2           The pathname of the new link to be created.
```

```
00491     \return                 0 on success, -1 on failure.
00492
00493     \note                   Most filesystems in KallistiOS do not support hard
00494                             links. If you call this function on a filesystem
00495                             that does not support hard links, the function will
00496                             return -1 and set errno to EMLINK.
00497 */
00498 int fs_link(const char *path1, const char *path2);
00499
00500 /** \brief  Create a symbolic link.
00501
00502     This function implements the POSIX function symlink(), which creates a
00503     symbolic link on the filesystem. Symbolic links are not required to point to
00504     an existing file (per POSIX) and may result in circular links if care is not
00505     taken. For now, symbolic links cannot cross filesystem boundaries in KOS.
00506
00507     \param  path1           The content of the link (i.e, what to point at).
00508     \param  path2           The pathname of the new link to be created.
00509     \return                 0 on success, -1 on failure.
00510
00511     \note                   Most filesystems in KallistiOS do not support
00512                             symbolic links. Filesystems that do not support
00513                             symlinks will simply set errno to ENOSYS and return
00514                             -1.
00515 */
00516 int fs_symlink(const char *path1, const char *path2);
00517
00518 /** \brief  Read the value of a symbolic link.
00519
00520     This function implements the POSIX function readlink(), which simply reads
00521     the value of the symbolic link at the end of a path. This does not resolve
00522     any internal links and it does not canonicalize the path either.
00523
00524     \param  path            The symbolic link to read.
00525     \param  buf             The buffer to place the link's contents in.
00526     \param  bufsize         The number of bytes allocated to buf.
00527     \return                 -1 on failure, the number of bytes placed into buf
00528                             on success. If the return value is equal to bufsize,
00529                             you may not have the whole link -- provide a larger
00530                             buffer and try again.
00531
00532     \note                   Most filesystems in KallistiOS do not support
00533                             symbolic links. Filesystems that do not support
00534                             symlinks will simply set errno to ENOSYS and return
00535                             -1.
00536 */
00537 ssize_t fs_readlink(const char *path, char *buf, size_t bufsize);
00538
00539 /** \brief  Retrieve information about the specified path.
00540
00541     This function retrieves status information on the given path. This function
00542     now returns the normal POSIX-style struct stat, rather than the old KOS
00543     stat_t structure. In addition, you can specify whether or not this function
00544     should resolve symbolic links on filesystems that support symlinks.
00545
00546     \param  path            The path to retrieve information about.
00547     \param  buf             The buffer to store stat information in.
00548     \param  flag            Specifies whether or not to resolve a symbolic link.
00549                             If you don't want to resolve any symbolic links at
00550                             the end of the path, pass AT_SYMLINK_NOFOLLOW,
00551                             otherwise pass 0.
00552     \return                 0 on success, -1 on failure.
00553 */
00554 int fs_stat(const char *path, struct stat *buf, int flag);
00555
00556 /** \brief  Rewind a directory to the start.
00557
00558     This function rewinds the position of a directory stream to the beginning of
00559     the directory.
00560
00561     \param  hnd             The opened directory's file descriptor.
00562     \return                 0 on success, -1 on failure.
00563
00564     \note                   Some filesystems may not support this function. If a
00565                             filesystem doesn't support it, errno will be set to
00566                             ENOSYS and -1 will be returned.
00567 */
00568 int fs_rewinddir(file_t hnd);
00569
00570 /** \brief  Retrieve information about an opened file.
00571
```

```
00572     This function retrieves status information on the given file descriptor,
00573     which must correspond to an already opened file.
00574
00575     \param  hnd             The file descriptor to retrieve information about.
00576     \param  buf             The buffer to store stat information in.
00577     \return                 0 on success, -1 on failure.
00578
00579     \note                   Some filesystems may not support this function. If a
00580                             filesystem doesn't support it, errno will be set to
00581                             ENOSYS and -1 will be returned.
00582 */
00583 int fs_fstat(file_t hnd, struct stat *buf);
00584
00585 /** \brief  Duplicate a file descriptor.
00586
00587     This function duplicates the specified file descriptor, returning a new file
00588     descriptor that can be used to access the file. This is equivalent to the
00589     standard POSIX function dup().
00590
00591     \param  oldfd           The old file descriptor to duplicate.
00592     \return                 The new file descriptor on success, -1 on failure.
00593 */
00594 file_t fs_dup(file_t oldfd);
00595
00596 /** \brief  Duplicate a file descriptor onto the specified descriptor.
00597
00598     This function duplicates the specified file descriptor onto the other file
00599     descriptor provided. If the newfd parameter represents an open file, that
00600     file will be closed before the old descriptor is duplicated onto it. This is
00601     equivalent to the standard POSIX function dup2().
00602
00603     \param  oldfd           The old file descriptor to duplicate.
00604     \param  newfd           The descriptor to copy into.
00605     \return                 The new file descriptor on success, -1 on failure.
00606 */
00607 file_t fs_dup2(file_t oldfd, file_t newfd);
00608
00609 /** \brief  Create a "transient" file descriptor.
00610
00611     This function creates and opens a new file descriptor that isn't associated
00612     directly with a file on the filesystem. This is used internally to actually
00613     open files, and should (in general) not be called by user code. Effectively,
00614     if you're trying to implement your own filesystem handler in your code, you
00615     may need this function, otherwise you should just ignore it.
00616
00617     \param  vfs             The VFS handler structure to use for the file.
00618     \param  hnd             Internal handle data for the file.
00619     \return                 The opened descriptor on success, -1 on failure.
00620 */
00621 file_t fs_open_handle(vfs_handler_t *vfs, void *hnd);
00622
00623 /** \brief  Retrieve the VFS Handler for a file descriptor.
00624
00625     This function retrieves the Handler structure for the VFS of the specified
00626     file descriptor. There is generally no reason to call this function in user
00627     code, as it is meant for use internally.
00628
00629     \param  fd              The file descriptor to retrieve the handler for.
00630     \return                 The VFS' handler structure.
00631 */
00632 vfs_handler_t *fs_get_handler(file_t fd);
00633
00634 /** \brief  Retrieve the internal handle for a file descriptor.
00635
00636     This function retrieves the internal file handle data of the specified file
00637     descriptor. There is generally no reason to call this function in user code,
00638     as it is meant for use internally.
00639
00640     \param  fd              The file descriptor to retrieve the handler for.
00641     \return                 The internal handle for the file descriptor.
00642 */
00643 void *fs_get_handle(file_t fd);
00644
00645 /** \brief  Get the current working directory of the running thread.
00646     \return                 The current working directory.
00647 */
00648 const char *fs_getwd(void);
00649
00650 /* Couple of util functions */
00651
00652 /** \brief  Copy a file.
```

```
00653
00654     This function copies the file at src to dst on the filesystem.
00655
00656     \param  src             The filename to copy from.
00657     \param  dst             The filename to copy to.
00658     \return                 The number of bytes copied successfully.
00659 */
00660 ssize_t fs_copy(const char *src, const char *dst);
00661
00662 /** \brief  Open and read a whole file into RAM.
00663
00664     This function opens the specified file, reads it into memory (allocating the
00665     necessary space with malloc), and closes the file. The caller is responsible
00666     for freeing the memory when they are done with it.
00667
00668     \param  src             The filename to open and read.
00669     \param  out_ptr         A pointer to the buffer on success, NULL otherwise.
00670     \return                 The size of the file on success, -1 otherwise.
00671 */
00672 ssize_t fs_load(const char *src, void **out_ptr);
00673
00674 /** \brief  Append a path component to a string.
00675
00676     This function acts mostly like the function strncat(), with a few slight
00677     differences. First, if the destination string doesn't end in a '/'
00678     character, this function will add it. Second, it returns the length of the
00679     resulting string, including the NUL terminator. Finally, no modification of
00680     the destination string will occur if there isn't enough space left in the
00681     string to do so.
00682
00683     \param  dst             The string to modify.
00684     \param  src             The path component to append.
00685     \param  len             The length allocated for dst.
00686     \return                 The length of the new string (including the NUL
00687                             terminator) on success, -1 otherwise.
00688
00689     \par    Error Conditions:
00690     \em     EFAULT - src or dst is a NULL pointer \n
00691     \em     EINVAL - len is zero \n
00692     \em     ENAMETOOLONG - the resulting path would be longer than len bytes \n
00693 */
00694 ssize_t fs_path_append(char *dst, const char *src, size_t len);
00695
00696 /** \brief  Initialize the virtual filesystem.
00697
00698     This is normally done for you by default when KOS starts. In general, there
00699     should be no reason for you to call this function.
00700
00701     \retval 0               On success.
00702 */
00703 int fs_init(void);
00704
00705 /** \brief  Shut down the virtual filesystem.
00706
00707     This is done for you by the normal shutdown procedure of KOS. There should
00708     not really be any reason for you to call this function yourself.
00709 */
00710 void fs_shutdown(void);
00711
00712 __END_DECLS
00713
00714 #endif  /* __KOS_FS_H */
```

9.44 include/kos/fs_dev.h File Reference

Driver for /dev/random and /dev/urandom.

```
#include <sys/cdefs.h>
#include <kos/fs.h>
```

9.44.1 Detailed Description

Driver for /dev/random and /dev/urandom.

This filesystem driver provides implementations of /dev/random and /dev/urandom for portability. It seeds randomness from uninitialized memory, and the clock. Obviously we are limited in how we can provide sufficient entropy on an embedded platform like the Dreamcast so the randomness from this driver will not win any awards but it should be sufficiently good for most purposes.

/dev/random is an alias to /dev/urandom for now.

Author

Luke Benstead

9.45 fs_dev.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/fs_dev.h
00004    (c)2023 - Luke Benstead
00005
00006 */
00007
00008 /** \file   kos/fs_dev.h
00009     \brief  Driver for /dev/random and /dev/urandom.
00010
00011     This filesystem driver provides implementations of /dev/random
00012     and /dev/urandom for portability. It seeds randomness from
00013     uninitialized memory, and the clock. Obviously we
00014     are limited in how we can provide sufficient entropy on an
00015     embedded platform like the Dreamcast so the randomness from
00016     this driver will not win any awards but it should be sufficiently
00017     good for most purposes.
00018
00019     /dev/random is an alias to /dev/urandom for now.
00020
00021     \author Luke Benstead
00022 */
00023
00024 #ifndef __DC_FS_DEV_H
00025 #define __DC_FS_DEV_H
00026
00027 #include <sys/cdefs.h>
00028 __BEGIN_DECLS
00029
00030 #include <kos/fs.h>
00031
00032 /* \cond */
00033 /* Initialization */
00034 int fs_dev_init(void);
00035 int fs_dev_shutdown(void);
00036 /* \endcond */
00037
00038 __END_DECLS
00039
00040 #endif  /* __DC_FS_DEV_H */
00041
```

9.46 include/kos/fs_pty.h File Reference

Pseudo-terminal virtual file system.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <kos/fs.h>
```

Functions

- int fs_pty_create (char *buffer, int maxbuflen, file_t *master_out, file_t *slave_out)

  Create a new pseudo-terminal.

9.46.1 Detailed Description

Pseudo-terminal virtual file system.

This file system implements a pseudo-terminal like concept (similar to /dev/pty in Linux). A call to fs_pty_create() will crate two file entries in the VFS, /pty/maXX and /pty/slXX (XX being some hexadecimal number). From there, anybody can open up either end and send data to the other side. Think of it as a simple message passing interface.

This file system mounts on /pty.

Author

Megan Potter

9.46.2 Function Documentation

fs_pty_create()

```
int fs_pty_create (
            char * buffer,
            int maxbuflen,
            file_t * master_out,
            file_t * slave_out )
```

Create a new pseudo-terminal.

This function creates a new pseudo-terminal, opening up two files in the /pty portion of the VFS.

Parameters

buffer	Storage for the name of the PTY, apparently not actually used (but potentially will be fixed at some point). If it was implemented, the name of the PTY would be here on successful return (if not NULL)
maxbuflen	The length of buffer
master_out	A pointer to store the file descriptor for the master end in (must not be NULL)
slave_out	A pointer to store the file descriptor for the slave end in (must not be NULL)

Return values

0	On success
-1	On error

Error Conditions:

ENOMEM - out of memory

9.47 fs_pty.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/fs_pty.h
00004    Copyright (C)2003 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/fs_pty.h
00009     \brief  Pseudo-terminal virtual file system.
00010
00011     This file system implements a pseudo-terminal like concept (similar to
00012     /dev/pty in Linux). A call to fs_pty_create() will crate two file entries in
00013     the VFS, /pty/maXX and /pty/slXX (XX being some hexadecimal number). From
00014     there, anybody can open up either end and send data to the other side. Think
00015     of it as a simple message passing interface.
00016
00017     This file system mounts on /pty.
00018
00019     \author Megan Potter
00020 */
00021
00022 #ifndef __KOS_FS_PTY_H
00023 #define __KOS_FS_PTY_H
00024
00025 #include <sys/cdefs.h>
00026 __BEGIN_DECLS
00027
00028 #include <arch/types.h>
00029 #include <kos/fs.h>
00030
00031 /** \brief  Create a new pseudo-terminal.
00032
00033     This function creates a new pseudo-terminal, opening up two files in the
00034     /pty portion of the VFS.
00035
00036     \param  buffer          Storage for the name of the PTY, apparently not
00037                             actually used (but potentially will be fixed at some
00038                             point). If it was implemented, the name of the PTY
00039                             would be here on successful return (if not NULL)
00040     \param  maxbuflen       The length of buffer
00041     \param  master_out      A pointer to store the file descriptor for the
00042                             master end in (must not be NULL)
00043     \param  slave_out       A pointer to store the file descriptor for the slave
00044                             end in (must not be NULL)
00045     \retval 0               On success
00046     \retval -1              On error
00047
00048     \par    Error Conditions:
00049     \em     ENOMEM - out of memory
00050 */
00051 int fs_pty_create(char * buffer, int maxbuflen, file_t * master_out, file_t * slave_out);
00052
00053 /** \cond */
00054 int fs_pty_init(void);
00055 int fs_pty_shutdown(void);
00056 /** \endcond */
00057
00058 __END_DECLS
00059
00060 #endif  /* __KOS_FS_PTY_H */
00061
```

9.48 include/kos/fs_ramdisk.h File Reference

RAM-based virtual file system.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <kos/fs.h>
```

Functions

- int fs_ramdisk_attach (const char *fn, void *obj, size_t size)

  Attach a block of memory as a file in the ramdisk.
- int fs_ramdisk_detach (const char *fn, void **obj, size_t *size)

  Detach a file from the ramdisk.

9.48.1 Detailed Description

RAM-based virtual file system.

This file contains support for a ramdisk VFS. This VFS allows you to map memory into files that will appear in /ram. Files in this VFS can grow as large as memory allows, and there is full read/write support here. This is useful, for (for instance) cacheing files read from the CD-ROM or for making temporary files.

You only have one ramdisk available, and its mounted on /ram.

Author

Megan Potter

9.48.2 Function Documentation

fs_ramdisk_attach()

```
int fs_ramdisk_attach (
            const char * fn,
            void * obj,
            size_t size )
```

Attach a block of memory as a file in the ramdisk.

This function takes a block of memory and associates it with a file on the ramdisk. This memory should be allocated with malloc(), as an unlink() of the file will call free on the block of memory. The ramdisk then effectively takes control of the block, and is responsible for it at that point.

Parameters

fn	The name to give the new file
obj	The block of memory to associate
size	The size of the block of memory

Return values

0	On success
-1	On failure

fs_ramdisk_detach()

```
int fs_ramdisk_detach (
            const char * fn,
            void ** obj,
            size_t * size )
```

Detach a file from the ramdisk.

This function retrieves the block of memory associated with the file, removing it from the ramdisk. You are responsible for freeing obj when you are done with it.

Parameters

fn	The name of the file to look for.
obj	A pointer to return the address of the object in.
size	A pointer to return the size of the object in.

Return values

0	On success
-1	On failure

9.49 fs_ramdisk.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/fs_ramdisk.h
00004    (c)2002 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/fs_ramdisk.h
00009     \brief  RAM-based virtual file system.
00010
00011     This file contains support for a ramdisk VFS. This VFS allows you to map
00012     memory into files that will appear in /ram. Files in this VFS can grow as
00013     large as memory allows, and there is full read/write support here. This is
00014     useful, for (for instance) cacheing files read from the CD-ROM or for making
00015     temporary files.
00016
00017     You only have one ramdisk available, and its mounted on /ram.
00018
00019     \author Megan Potter
00020 */
00021
00022 #ifndef __KOS_FS_RAMDISK_H
```

```
00023 #define __KOS_FS_RAMDISK_H
00024
00025 #include <sys/cdefs.h>
00026 __BEGIN_DECLS
00027
00028 #include <arch/types.h>
00029 #include <kos/fs.h>
00030
00031 /** \cond */
00032 int fs_ramdisk_init(void);
00033 int fs_ramdisk_shutdown(void);
00034 /** \endcond */
00035
00036 /** \brief  Attach a block of memory as a file in the ramdisk.
00037
00038     This function takes a block of memory and associates it with a file on the
00039     ramdisk. This memory should be allocated with malloc(), as an unlink() of
00040     the file will call free on the block of memory. The ramdisk then effectively
00041     takes control of the block, and is responsible for it at that point.
00042
00043     \param  fn              The name to give the new file
00044     \param  obj             The block of memory to associate
00045     \param  size            The size of the block of memory
00046     \retval 0               On success
00047     \retval -1              On failure
00048 */
00049 int fs_ramdisk_attach(const char * fn, void * obj, size_t size);
00050
00051 /** \brief  Detach a file from the ramdisk.
00052
00053     This function retrieves the block of memory associated with the file,
00054     removing it from the ramdisk. You are responsible for freeing obj when you
00055     are done with it.
00056
00057     \param  fn              The name of the file to look for.
00058     \param  obj             A pointer to return the address of the object in.
00059     \param  size            A pointer to return the size of the object in.
00060     \retval 0               On success
00061     \retval -1              On failure
00062 */
00063 int fs_ramdisk_detach(const char * fn, void ** obj, size_t * size);
00064
00065 __END_DECLS
00066
00067 #endif  /* __KOS_FS_RAMDISK_H */
00068
```

9.50 include/kos/fs_romdisk.h File Reference

ROMFS virtual file system.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <kos/fs.h>
```

Functions

- int fs_romdisk_mount (const char *mountpoint, const uint8 *img, int own_buffer)

  Mount a ROMFS image as a new filesystem.
- int fs_romdisk_unmount (const char *mountpoint)

  Unmount a ROMFS image.

9.50.1 Detailed Description

ROMFS virtual file system.

This file contains support for the romdisk VFS. This VFS allows you to make Linux-style ROMFS images and either embed them into your binary or load them at runtime from some other source (such as a CD-ROM). These images are made with the genromfs program that is included in the utils portion of the tree.

You can choose to automount one ROMFS image by embedding it into your binary and using the appropriate KOS_INIT_FLAGS() setting. The embedded ROMFS will mount itself on /rd. You can also mount additional images that you load from some other source on whatever mountpoint you want.

Author

Megan Potter

9.50.2 Function Documentation

fs_romdisk_mount()

```
int fs_romdisk_mount (
            const char * mountpoint,
            const uint8 * img,
            int own_buffer )
```

Mount a ROMFS image as a new filesystem.

This function will mount a ROMFS image that has been loaded into memory to the specified mountpoint.

Parameters

mountpoint	The directory to mount this romdisk on
img	The ROMFS image
own_buffer	If 0, you are still responsible for img, and must free it if appropriate. If non-zero, img will be freed when it is unmounted

Return values

0	On success
-1	If fs_romdisk_init not called
-2	If img is invalid
-3	If a malloc fails

fs_romdisk_unmount()

```
int fs_romdisk_unmount (
            const char * mountpoint )
```

Unmount a ROMFS image.

This function unmounts a ROMFS image that has been previously mounted with fs_romdisk_mount(). This function does not check for open files on the fs, so make sure that all files have been closed before calling it. If the VFS owns the buffer (own_buffer was non-zero when you called the mount function) then this function will also free the buffer.

Parameters

mountpoint	The ROMFS to unmount

Return values

0	On success
-1	On error

Error Conditions:

ENOENT - no such ROMFS was mounted

9.51 fs_romdisk.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/fs_romdisk.h
00004    (c)2001 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/fs_romdisk.h
00009     \brief  ROMFS virtual file system.
00010
00011     This file contains support for the romdisk VFS. This VFS allows you to make
00012     Linux-style ROMFS images and either embed them into your binary or load them
00013     at runtime from some other source (such as a CD-ROM). These images are made
00014     with the genromfs program that is included in the utils portion of the tree.
00015
00016     You can choose to automount one ROMFS image by embedding it into your binary
00017     and using the appropriate KOS_INIT_FLAGS() setting. The embedded ROMFS will
00018     mount itself on /rd. You can also mount additional images that you load
00019     from some other source on whatever mountpoint you want.
00020
00021     \author Megan Potter
00022 */
00023
00024 #ifndef __KOS_FS_ROMDISK_H
00025 #define __KOS_FS_ROMDISK_H
00026
00027 #include <sys/cdefs.h>
00028 __BEGIN_DECLS
00029
00030 #include <arch/types.h>
00031 #include <kos/fs.h>
00032
00033 /** \cond */
00034 /* Initialize the file system */
00035 void fs_romdisk_init(void);
00036
00037 /* De-init the file system; also unmounts any mounted images. */
00038 void fs_romdisk_shutdown(void);
00039 /** \endcond */
00040
00041 /* NOTE: the mount/unmount are _not_ thread safe as regards doing multiple
00042    mounts/unmounts in different threads at the same time, and they don't
```

```
00043    check for open files currently either. Caveat emptor! */
00044
00045 /** \brief  Mount a ROMFS image as a new filesystem.
00046
00047     This function will mount a ROMFS image that has been loaded into memory to
00048     the specified mountpoint.
00049
00050     \param  mountpoint      The directory to mount this romdisk on
00051     \param  img             The ROMFS image
00052     \param  own_buffer      If 0, you are still responsible for img, and must
00053                             free it if appropriate. If non-zero, img will be
00054                             freed when it is unmounted
00055     \retval 0               On success
00056     \retval -1              If fs_romdisk_init not called
00057     \retval -2              If img is invalid
00058     \retval -3              If a malloc fails
00059 */
00060 int fs_romdisk_mount(const char * mountpoint, const uint8 *img, int own_buffer);
00061
00062 /** \brief  Unmount a ROMFS image.
00063
00064     This function unmounts a ROMFS image that has been previously mounted with
00065     fs_romdisk_mount(). This function does not check for open files on the fs,
00066     so make sure that all files have been closed before calling it. If the VFS
00067     owns the buffer (own_buffer was non-zero when you called the mount function)
00068     then this function will also free the buffer.
00069
00070     \param  mountpoint      The ROMFS to unmount
00071     \retval 0               On success
00072     \retval -1              On error
00073
00074     \par    Error Conditions:
00075     \em     ENOENT - no such ROMFS was mounted
00076 */
00077 int fs_romdisk_unmount(const char * mountpoint);
00078
00079 __END_DECLS
00080
00081 #endif  /* __KOS_FS_ROMDISK_H */
00082
```

9.52 include/kos/fs_socket.h File Reference

Definitions for a sockets "filesystem".

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <kos/fs.h>
#include <kos/net.h>
#include <sys/queue.h>
#include <sys/socket.h>
#include <stdint.h>
```

Data Structures

- struct net_socket_t

  Internal representation of a socket for fs_socket.
- struct fs_socket_proto_t

  Internal sockets protocol handler.

Macros

- #define FS_SOCKET_PROTO_ENTRY { NULL, NULL }

  Initializer for the entry field in the fs_socket_proto_t struct.
- #define FS_SOCKET_NONBLOCK 0x00000001 /∗∗ \brief Non-blocking operations ∗/
- #define FS_SOCKET_V6ONLY 0x00000002 /∗∗ \brief IPv6 Only ∗/
- #define FS_SOCKET_GEN_MAX 0x00008000 /∗∗ \brief Maximum generic flag ∗/
- #define FS_SOCKET_FAM_MAX 0x00800000 /∗∗ \brief Maximum family flag ∗/

Functions

- net_socket_t ∗ fs_socket_open_sock (fs_socket_proto_t ∗proto)

  Open a socket without calling the protocol initializer.
- int fs_socket_input (netif_t ∗src, int domain, int protocol, const void ∗hdr, const uint8 ∗data, size_t size)

  Input a packet into some socket family handler.
- int fs_socket_proto_add (fs_socket_proto_t ∗proto)

  Add a new protocol for use with fs_socket.
- int fs_socket_proto_remove (fs_socket_proto_t ∗proto)

  Unregister a protocol from fs_socket.

9.52.1 Detailed Description

Definitions for a sockets "filesystem".

This file provides definitions to support the BSD-sockets-like filesystem in KallistiOS. Technically, this filesystem mounts itself on /sock, but it doesn't export any files there, so that point is largely irrelevant. The filesystem is designed to be extensible, making it possible to add additional socket family handlers at runtime. Currently, the kernel only implements UDP sockets over IPv4 and IPv6, but as mentioned, this can be extended in a fairly straightforward manner. In general, as a user of KallistiOS (someone not interested in adding additional socket family drivers), there's very little in this file that will be of interest.

Author

Lawrence Sebald

9.52.2 Macro Definition Documentation

FS_SOCKET_PROTO_ENTRY

```
#define FS_SOCKET_PROTO_ENTRY { NULL, NULL }
```

Initializer for the entry field in the fs_socket_proto_t struct.

9.52.3 Function Documentation

fs_socket_input()

```
int fs_socket_input (
            netif_t * src,
            int domain,
            int protocol,
            const void * hdr,
            const uint8 * data,
            size_t size )
```

Input a packet into some socket family handler.

This function is used by the lower-level network protocol handlers to input packets for further processing by upper-level protocols. This will call the input function on the family handler, if one is found.

Parameters

src	The network interface the packet came in on
domain	The low-level protocol used (AF_INET or AF_INET6)
protocol	The upper-level protocol that we're looking for
hdr	The low-level protocol header
data	The upper-level packet, without any lower-level protocol headers, but with the upper-level ones intact
size	The size of the packet (the data parameter)

Return values

-2	The protocol is not known
-1	Protocol-level error processing packet
0	On success

fs_socket_open_sock()

```
net_socket_t * fs_socket_open_sock (
            fs_socket_proto_t * proto )
```

Open a socket without calling the protocol initializer.

This function creates a new socket, but does not call the protocol's socket() function. This is meant to be used for things like accepting an incoming connection, where calling the regular socket initializer could cause issues. You shouldn't really have any need to call this function unless you are implementing a new protocol handler.

Parameters

proto	The protocol to use for the socket.

Returns

The newly created socket on success, NULL on failure.

Error Conditions:

EWOULDBLOCK - if the function would block in an IRQ
ENOMEM - out of memory
EMFILE - too many files open

fs_socket_proto_add()

```
int fs_socket_proto_add (
            fs_socket_proto_t * proto )
```

Add a new protocol for use with fs_socket.

This function registers a protocol handler with fs_socket for use when creating and using sockets. This protocol handler must implement all of the functions in the fs_socket_proto_t structure. See the code in kos/kernel/net/net_udp.c for an example of how to do this.

This function is NOT safe to call inside an interrupt.

Parameters

proto	The new protocol handler to register

Return values

0	On success (no error conditions are currently defined)

fs_socket_proto_remove()

```
int fs_socket_proto_remove (
            fs_socket_proto_t * proto )
```

Unregister a protocol from fs_socket.

This function does the exact opposite of fs_socket_proto_add, and removes a protocol from use with fs_socket. It is the programmer's responsibility to make sure that no sockets are still around that are registered with the protocol to be removed (as they will not work properly once the handler has been removed).

Parameters

proto	The protocol handler to remove

Return values

-1	On error (This function does not directly change errno)
0	On success

9.53 fs_socket.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/fs_socket.h
00004    Copyright (C) 2006, 2009, 2010, 2012, 2013 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   kos/fs_socket.h
00009     \brief  Definitions for a sockets "filesystem".
00010
00011     This file provides definitions to support the BSD-sockets-like filesystem
00012     in KallistiOS. Technically, this filesystem mounts itself on /sock, but it
00013     doesn't export any files there, so that point is largely irrelevant. The
00014     filesystem is designed to be extensible, making it possible to add
00015     additional socket family handlers at runtime. Currently, the kernel only
00016     implements UDP sockets over IPv4 and IPv6, but as mentioned, this can be
00017     extended in a fairly straightforward manner. In general, as a user of
00018     KallistiOS (someone not interested in adding additional socket family
00019     drivers), there's very little in this file that will be of interest.
00020
00021     \author Lawrence Sebald
00022 */
00023
00024 #ifndef __KOS_FS_SOCKET_H
00025 #define __KOS_FS_SOCKET_H
00026
00027 #include <sys/cdefs.h>
00028
00029 __BEGIN_DECLS
00030
00031 #include <arch/types.h>
00032 #include <kos/fs.h>
00033 #include <kos/net.h>
00034 #include <sys/queue.h>
00035 #include <sys/socket.h>
00036 #include <stdint.h>
00037
00038 struct fs_socket_proto;
00039
00040 /** \brief  Internal representation of a socket for fs_socket.
00041
00042     This structure is the internal representation of a socket "file" that is
00043     used within fs_socket. A normal user will never deal with this structure
00044     directly (only protocol handlers and fs_socket itself ever sees this
00045     structure directly).
00046
00047     \headerfile kos/fs_socket.h
00048 */
00049 typedef struct net_socket {
00050     /** \cond */
00051     /* List handle */
00052     LIST_ENTRY(net_socket) sock_list;
00053     /** \endcond */
00054
00055     /** \brief   File handle from the VFS layer. */
00056     file_t fd;
00057
00058     /** \brief  The protocol handler for this socket. */
00059     struct fs_socket_proto *protocol;
00060
00061     /** \brief  Protocol-specific data. */
00062     void *data;
00063 } net_socket_t;
00064
```

```
00065 /** \brief  Internal sockets protocol handler.
00066
00067     This structure is a protocol handler used within fs_socket. Each protocol
00068     that is supported has one of these registered for it within the kernel.
00069     Generally, users will not come in contact with this structure (unless you're
00070     planning on writing a protocol handler), and it can generally be ignored.
00071
00072     For a complete list of appropriate errno values to return from any functions
00073     that are in here, take a look at the Single Unix Specification (aka, the
00074     POSIX spec), specifically the page about sys/socket.h (and all the functions
00075     that it defines, which is available at
00076     http://www.opengroup.org/onlinepubs/9699919799/basedefs/sys_socket.h.html .
00077
00078     \headerfile kos/fs_socket.h
00079 */
00080 typedef struct fs_socket_proto {
00081     /** \brief  Entry into the global list of protocols.
00082
00083         Contrary to what Doxygen might think, this is <b>NOT</b> a function.
00084         This should be initialized with the FS_SOCKET_PROTO_ENTRY macro before
00085         adding the protocol to the kernel with fs_socket_proto_add().
00086     */
00087     TAILQ_ENTRY(fs_socket_proto) entry;
00088
00089     /** \brief  Domain of support for this protocol handler.
00090
00091         This field determines which sockets domain this protocol handler
00092         actually supports. This corresponds with the domain argument of the
00093         ::socket() function.
00094     */
00095     int domain;
00096
00097     /** \brief  Type of support for this protocol handler.
00098
00099         This field determines which types of sockets that this protocol handler
00100         pays attention to. This corresponds with the type argument of the
00101         ::socket() function.
00102     */
00103     int type;
00104
00105     /** \brief  Protocol of support for this protocol handler.
00106
00107         This field determines the protocol that this protocol handler actually
00108         pays attention to. This corresponds with the protocol argument of the
00109         ::socket() function.
00110     */
00111     int protocol;
00112
00113     /** \brief  Create a new socket for the protocol.
00114
00115         This function must create a new socket, initializing any data that the
00116         protocol might need for the socket, based on the parameters passed in.
00117         The socket passed in is already initialized prior to the handler being
00118         called, and will be cleaned up by fs_socket if an error is returned from
00119         the handler (a return value of -1).
00120
00121         \param  s           The socket structure to initialize
00122         \param  domain      Domain of the socket
00123         \param  type        Type of the socket
00124         \param protocol     Protocol of the socket
00125         \retval -1          On error (errno should be set appropriately)
00126         \retval 0           On success
00127     */
00128     int (*socket)(net_socket_t *s, int domain, int type, int protocol);
00129
00130     /** \brief  Close a socket that was created with the protocol.
00131
00132         This function must do any work required to close a socket and destroy
00133         it. This function will be called when a socket requests to be closed
00134         with the close system call. There are no errors defined for this
00135         function.
00136
00137         \param  s           The socket to close
00138     */
00139     void (*close)(net_socket_t *hnd);
00140
00141     /** \brief  Accept a connection on a socket created with the protocol.
00142
00143         This function should implement the ::accept() system call for the
00144         protocol. The semantics are exactly as expected for that function.
00145
```

```
00146         \param  s           The socket to accept a connection on
00147         \param  addr        The address of the incoming connection
00148         \param  alen        The length of the address
00149         \return             A newly created socket for the incoming connection
00150                             or -1 on error (with errno set appropriately)
00151     */
00152     int (*accept)(net_socket_t *s, struct sockaddr *addr, socklen_t *alen);
00153
00154     /** \brief  Bind a socket created with the protocol to an address.
00155
00156         This function should implement the ::bind() system call for the
00157         protocol. The semantics are exactly as expected for that function.
00158
00159         \param  s           The socket to bind to the address
00160         \param  addr        The address to bind to
00161         \param  alen        The length of the address
00162         \retval -1          On error (set errno appropriately)
00163         \retval 0           On success
00164     */
00165     int (*bind)(net_socket_t *s, const struct sockaddr *addr, socklen_t alen);
00166
00167     /** \brief  Connect a socket created with the protocol to a remote system.
00168
00169         This function should implement the ::connect() system call for the
00170         protocol. The semantics are exactly as expected for that function.
00171
00172         \param  s           The socket to connect with
00173         \param  addr        The address to connect to
00174         \param  alen        The length of the address
00175         \retval -1          On error (with errno set appropriately)
00176         \retval 0           On success
00177     */
00178     int (*connect)(net_socket_t *s, const struct sockaddr *addr,
00179                    socklen_t alen);
00180
00181     /** \brief  Listen for incoming connections on a socket created with the
00182                 protocol.
00183
00184         This function should implement the ::listen() system call for the
00185         protocol. The semantics are exactly as expected for that function.
00186
00187         \param  s           The socket to listen on
00188         \param  backlog     The number of connections to queue
00189         \retval -1          On error (with errno set appropriately)
00190         \retval 0           On success
00191     */
00192     int (*listen)(net_socket_t *s, int backlog);
00193
00194     /** \brief  Recieve data on a socket created with the protocol.
00195
00196         This function should implement the ::recvfrom() system call for the
00197         protocol. The semantics are exactly as expected for that function. Also,
00198         this function should implement the ::recv() system call, which will
00199         call this function with NULL for addr and alen.
00200
00201         \param  s           The socket to receive data on
00202         \param  buffer      The buffer to save data in
00203         \param  len         The length of the buffer
00204         \param  flags       Flags to the function
00205         \param  addr        Space to store the address that data came from (NULL
00206                             if this was called by ::recv())
00207         \param  alen        Space to store the length of the address (NULL if
00208                             this was called by ::recv())
00209         \retval -1          On error (set errno appropriately)
00210         \retval 0           No outstanding data and the peer has disconnected
00211                             cleanly
00212         \retval n           The number of bytes received (may be less than len)
00213     */
00214     ssize_t (*recvfrom)(net_socket_t *s, void *buffer, size_t len, int flags,
00215                         struct sockaddr *addr, socklen_t *alen);
00216
00217     /** \brief  Send data on a socket created with the protocol.
00218
00219         This function should implement the ::sendto() system call for the
00220         protocol. The semantics are exactly as expected for that function. Also,
00221         this function should implement the ::send() system call, which will
00222         call this function with NULL for addr and 0 for alen.
00223
00224         \param  s           The socket to send data on
00225         \param  msg         The data to send
00226         \param  len         The length of data to send
```

```
00227         \param  flags       Flags to the function
00228         \param  addr        The address to send data to (NULL if this was called
00229                             by ::send())
00230         \param  alen        The length of the address (0 if this was called by
00231                             ::send())
00232         \retval -1          On error (set errno appropriately)
00233         \retval n           The number of bytes actually sent (may be less than
00234                             len)
00235     */
00236     ssize_t (*sendto)(net_socket_t *s, const void *msg, size_t len, int flags,
00237                       const struct sockaddr *addr, socklen_t alen);
00238
00239     /** \brief  Shut down a socket created with the protocol.
00240
00241         This function should implement the ::shutdown() system call for the
00242         protocol. The semantics are exactly as expected for that function.
00243
00244         \param  s           The socket to shut down
00245         \param  how         What should be shut down on the socket
00246         \retval -1          On error (set errno appropriately)
00247         \retval 0           On success
00248     */
00249     int (*shutdownsock)(net_socket_t *s, int how);
00250
00251     /** \brief  Input a packet into a protocol.
00252
00253         This function should read in the packet specified by the arguments and
00254         sort out what exactly to do with it. This usually involves checking if
00255         there is an open socket with the source address and adding it to a
00256         packet queue if there is.
00257
00258         \param  src         The interface the packet was input on
00259         \param  domain      The low-level protocol used (AF_INET or AF_INET6)
00260         \param  hdr         The low-level protocol header
00261         \param  data        The packet itself, including any protcol headers,
00262                             but not any from lower-level protocols
00263         \param  size        The size of the packet, not including any lower-
00264                             level protocol headers
00265         \retval -1          On error (the packet is discarded)
00266         \retval 0           On success
00267     */
00268     int (*input)(netif_t *src, int domain, const void *hdr, const uint8 *data,
00269                  size_t size);
00270
00271     /** \brief  Get socket options.
00272
00273         This function should implement the ::getsockopt() system call for the
00274         given protocol. The semantics are exactly as defined for that function.
00275
00276         Currently all options (regardless of level) are passed onto the
00277         protocol handler.
00278
00279         \param  s               The socket to get options for.
00280         \param  level           The protocol level to get options at.
00281         \param  option_name     The option to look up.
00282         \param  option_value    Storage for the value of the option.
00283         \param  option_len      The length of option_value on call, and the real
00284                                 option length (if less than the original value)
00285                                 on return.
00286         \retval -1              On error (set errno appropriately).
00287         \retval 0               On success.
00288     */
00289     int (*getsockopt)(net_socket_t *s, int level, int option_name,
00290                       void *option_value, socklen_t *option_len);
00291
00292     /** \brief  Set socket options.
00293
00294         This function should implement the ::setsockopt() system call for the
00295         given protocol. The semantics are exactly as defined for that function.
00296
00297         Currently all options (regardless of level) are passed onto the
00298         protocol handler.
00299
00300         \param  s               The socket to set options for.
00301         \param  level           The protocol level to set options at.
00302         \param  option_name     The option to set.
00303         \param  option_value    The value to set for the option.
00304         \param  option_len      The length of the option_value value.
00305         \retval -1              On error (set errno appropriately).
00306         \retval 0               On success.
00307     */
```

```
00308     int (*setsockopt)(net_socket_t *s, int level, int option_name,
00309                       const void *option_value, socklen_t option_len);
00310
00311     /** \brief  Get socket name.
00312
00313         This function should implement the ::getsockname() system call for the
00314         given protocol. The semantics are exactly as defined for that function.
00315
00316         Currently all options (regardless of level) are passed onto the
00317         protocol handler.
00318
00319         \param  s               The socket to get the name of.
00320         \param  name            Pointer to a sockaddr structure which will hold
00321                                 the resulting address information.
00322         \param  name_len        The amount of space pointed to by name, in
00323                                 bytes. On return, this is set to the actual size
00324                                 of the returned address information.
00325         \retval -1              On error (set errno appropriately).
00326         \retval 0               On success.
00327     */
00328     int (*getsockname)(net_socket_t *s, struct sockaddr *name, socklen_t *name_len);
00329
00330     /** \brief  Manipulate file options.
00331
00332         This function should implement the fcntl() system call for the given
00333         protocol. The semantics are exactly as defined for that function.
00334
00335         \param  s           The socket to manipulate.
00336         \param  cmd         The fcntl command to run.
00337         \param  ap          Arguments to the command.
00338         \retval -1          On error (generally, set errno appropriately).
00339     */
00340     int (*fcntl)(net_socket_t *s, int cmd, va_list ap);
00341
00342     /** \brief  Poll for events.
00343
00344         This function should check the given socket for any events that may have
00345         already occured that are specified. This is used to back the ::poll()
00346         system call. This function should not block to wait for any events. This
00347         function may be called in an interrupt.
00348
00349         \param  s           The socket to poll.
00350         \param  events      The events to check for.
00351         \retval             A mask of any of the events specified that are
00352                             currently true in the socket. 0 if none are true.
00353     */
00354     short (*poll)(net_socket_t *s, short events);
00355 } fs_socket_proto_t;
00356
00357 /** \brief  Initializer for the entry field in the fs_socket_proto_t struct. */
00358 #define FS_SOCKET_PROTO_ENTRY { NULL, NULL }
00359
00360 /* \cond */
00361 /* Init/shutdown */
00362 int fs_socket_init(void);
00363 int fs_socket_shutdown(void);
00364 /* \endcond */
00365
00366 /** \brief  Open a socket without calling the protocol initializer.
00367
00368     This function creates a new socket, but does not call the protocol's
00369     socket() function. This is meant to be used for things like accepting an
00370     incoming connection, where calling the regular socket initializer could
00371     cause issues. You shouldn't really have any need to call this function
00372     unless you are implementing a new protocol handler.
00373
00374     \param  proto       The protocol to use for the socket.
00375     \return             The newly created socket on success, NULL on failure.
00376
00377     \par    Error Conditions:
00378     \em     EWOULDBLOCK - if the function would block in an IRQ \n
00379     \em     ENOMEM - out of memory \n
00380     \em     EMFILE - too many files open
00381 */
00382 net_socket_t *fs_socket_open_sock(fs_socket_proto_t *proto);
00383
00384 /** \defgroup sock_flags                Socket flags
00385
00386     These are the available flags defined for sockets.
00387
00388     Every flag after FS_SOCKET_FAM_MAX is for internal-use only, and should
```

```
00389     never be passed into any functions.
00390     @{
00391 */
00392 #define FS_SOCKET_NONBLOCK  0x00000001  /** \brief Non-blocking operations */
00393 #define FS_SOCKET_V6ONLY    0x00000002  /** \brief IPv6 Only */
00394
00395 #define FS_SOCKET_GEN_MAX   0x00008000  /** \brief Maximum generic flag */
00396 #define FS_SOCKET_FAM_MAX   0x00800000  /** \brief Maximum family flag */
00397 /** @} */
00398
00399 /** \brief  Input a packet into some socket family handler.
00400
00401     This function is used by the lower-level network protocol handlers to input
00402     packets for further processing by upper-level protocols. This will call the
00403     input function on the family handler, if one is found.
00404
00405     \param  src         The network interface the packet came in on
00406     \param  domain      The low-level protocol used (AF_INET or AF_INET6)
00407     \param  protocol    The upper-level protocol that we're looking for
00408     \param  hdr         The low-level protocol header
00409     \param  data        The upper-level packet, without any lower-level protocol
00410                         headers, but with the upper-level ones intact
00411     \param  size        The size of the packet (the data parameter)
00412     \retval -2          The protocol is not known
00413     \retval -1          Protocol-level error processing packet
00414     \retval 0           On success
00415 */
00416 int fs_socket_input(netif_t *src, int domain, int protocol, const void *hdr,
00417                     const uint8 *data, size_t size);
00418
00419 /** \brief  Add a new protocol for use with fs_socket.
00420
00421     This function registers a protocol handler with fs_socket for use when
00422     creating and using sockets. This protocol handler must implement all of the
00423     functions in the fs_socket_proto_t structure. See the code in
00424     kos/kernel/net/net_udp.c for an example of how to do this.
00425
00426     This function is NOT safe to call inside an interrupt.
00427
00428     \param  proto       The new protocol handler to register
00429     \retval 0           On success (no error conditions are currently defined)
00430 */
00431 int fs_socket_proto_add(fs_socket_proto_t *proto);
00432
00433 /** \brief  Unregister a protocol from fs_socket.
00434
00435     This function does the exact opposite of fs_socket_proto_add, and removes
00436     a protocol from use with fs_socket. It is the programmer's responsibility to
00437     make sure that no sockets are still around that are registered with the
00438     protocol to be removed (as they will not work properly once the handler has
00439     been removed).
00440
00441     \param  proto       The protocol handler to remove
00442     \retval -1          On error (This function does not directly change errno)
00443     \retval 0           On success
00444 */
00445 int fs_socket_proto_remove(fs_socket_proto_t *proto);
00446
00447 __END_DECLS
00448
00449 #endif  /* __KOS_FS_SOCKET_H */
00450
```

9.54 include/kos/genwait.h File Reference

Generic wait system.

```
#include <sys/cdefs.h>
#include <kos/thread.h>
```

Functions

- int genwait_wait (void *obj, const char *mesg, int timeout, void(*callback)(void *))
  Sleep on an object.
- int genwait_wake_cnt (void *obj, int cnt, int err)
  Wake up a number of threads sleeping on an object.
- void genwait_wake_all (void *obj)
  Wake up all threads sleeping on an object.
- void genwait_wake_one (void *obj)
  Wake up one thread sleeping on an object.
- void genwait_wake_all_err (void *obj, int err)
  Wake up all threads sleeping on an object, with an error.
- void genwait_wake_one_err (void *obj, int err)
  Wake up one thread sleeping on an object, with an error.
- int genwait_wake_thd (void *obj, kthread_t *thd, int err)
  Wake up a specific thread that is sleeping on an object.
- void genwait_check_timeouts (uint64 now)
  Look for timed out genwait_wait() calls.
- uint64 genwait_next_timeout (void)
  Look for the next timeout event time.

9.54.1 Detailed Description

Generic wait system.

The generic wait system in KOS, like many other portions of KOS, is based on an idea from the BSD kernel. It allows you to sleep on any random object and later wake up any threads that happen to be sleeping on thta object. All of KOS' sync primatives (other than spinlocks) are based on this concept, and it can be used for some fairly useful things.

Author

Megan Potter

Lawrence Sebald

9.54.2 Function Documentation

genwait_check_timeouts()

```
void genwait_check_timeouts (
            uint64 now )
```

Look for timed out genwait_wait() calls.

There should be no reason you need to call this function, it is called internally by the scheduler for you.

Parameters

now	The current system time, in milliseconds since boot

genwait_next_timeout()

```
uint64 genwait_next_timeout (
            void  )
```

Look for the next timeout event time.

This function looks up when the next genwait_wait() call will timeout. This function is for the internal use of the scheduler, and should not be called from user code.

Returns

The next timeout time in milliseconds since boot, or 0 if there are no pending genwait_wait() calls

genwait_wait()

```
int genwait_wait (
            void * obj,
            const char * mesg,
            int timeout,
            void(*)(void *) callback )
```

Sleep on an object.

This function sleeps on the specified object. You are not allowed to call this function inside an interrupt.

Parameters

obj	The object to sleep on
mesg	A message to show in the status
timeout	If not woken before this many milliseconds have passed, wake up anyway
callback	If non-NULL, call this function with obj as its argument if the wait times out (but before the calling thread has been woken back up)

Return values

0	On successfully being woken up (not by timeout)
-1	On error or being woken by timeout

Error Conditions:

EAGAIN - on timeout

genwait_wake_all()

```
void genwait_wake_all (
            void * obj )
```

Wake up all threads sleeping on an object.

This function simply calls genwait_wake_cnt(obj, -1, 0).

Parameters

obj	The object to wake threads that are sleeping on it

See also

genwait_wake_cnt()

genwait_wake_all_err()

```
void genwait_wake_all_err (
            void * obj,
            int err )
```

Wake up all threads sleeping on an object, with an error.

This function simply calls genwait_wake_cnt(obj, -1, err).

Parameters

obj	The object to wake threads that are sleeping on it
err	The value to set in the threads' errno values

See also

genwait_wake_cnt()

genwait_wake_cnt()

```
int genwait_wake_cnt (
            void * obj,
```

```
            int cnt,
            int err )
```

Wake up a number of threads sleeping on an object.

This function wakes up the specified number of threads sleeping on the object specified.

Parameters

obj	The object to wake threads that are sleeping on it
cnt	The number of threads to wake, if <= 0, wake all
err	The errno code to set as the errno value on the woken threads. If this is 0 (EOK), then the thread's errno will not be changed, and the thread will get a return value of 0 from the genwait_wait(). If it is non-zero, the thread will get a return value of -1 and errno will be set to this value for the woken threads.

Returns

The number of threads woken

genwait_wake_one()

```
void genwait_wake_one (
            void * obj )
```

Wake up one thread sleeping on an object.

This function simply calls genwait_wake_cnt(obj, 1, 0).

Parameters

obj	The object to wake threads that are sleeping on it

See also

genwait_wake_cnt()

genwait_wake_one_err()

```
void genwait_wake_one_err (
            void * obj,
            int err )
```

Wake up one thread sleeping on an object, with an error.

This function simply calls genwait_wake_cnt(obj, 1, err).

Parameters

obj	The object to wake threads that are sleeping on it
err	The value to set in the threads' errno values

See also

genwait_wake_cnt()

genwait_wake_thd()

```
int genwait_wake_thd (
            void * obj,
            kthread_t * thd,
            int err )
```

Wake up a specific thread that is sleeping on an object.

This function wakes up the specfied thread, assuming it is sleeping on the specified object.

Parameters

obj	The object to wake the thread from
thd	The specific thread to wake
err	The errno code to set as the errno value on the woken thread. If this is 0 (EOK), then the thread's errno will not be changed, and the thread will get a return value of 0 from the genwait_wait(). If it is non-zero, the thread will get a return value of -1 and errno will be set to this value for the woken threads.

Returns

The number of threads woken, which should be 1 on success.

9.55 genwait.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    include/kos/genwait.h
00004    Copyright (C) 2003 Megan Potter
00005    Copyright (C) 2012 Lawrence Sebald
00006
00007 */
00008
00009 /** \file   kos/genwait.h
00010     \brief  Generic wait system.
00011     \ingroup kthreads
00012
00013     The generic wait system in KOS, like many other portions of KOS, is based on
00014     an idea from the BSD kernel. It allows you to sleep on any random object and
00015     later wake up any threads that happen to be sleeping on thta object. All of
00016     KOS' sync primatives (other than spinlocks) are based on this concept, and
00017     it can be used for some fairly useful things.
```

```
00018
00019     \author Megan Potter
00020     \author Lawrence Sebald
00021 */
00022
00023 #ifndef __KOS_GENWAIT_H
00024 #define __KOS_GENWAIT_H
00025
00026 #include <sys/cdefs.h>
00027 __BEGIN_DECLS
00028
00029 #include <kos/thread.h>
00030
00031 /** \brief  Sleep on an object.
00032
00033     This function sleeps on the specified object. You are not allowed to call
00034     this function inside an interrupt.
00035
00036     \param  obj             The object to sleep on
00037     \param  mesg            A message to show in the status
00038     \param  timeout         If not woken before this many milliseconds have
00039                             passed, wake up anyway
00040     \param  callback        If non-NULL, call this function with obj as its
00041                             argument if the wait times out (but before the
00042                             calling thread has been woken back up)
00043     \retval 0               On successfully being woken up (not by timeout)
00044     \retval -1              On error or being woken by timeout
00045
00046     \par    Error Conditions:
00047     \em     EAGAIN - on timeout
00048 */
00049 int genwait_wait(void * obj, const char * mesg, int timeout, void (*callback)(void *));
00050
00051 /* Wake up N threads waiting on the given object. If cnt is <=0, then we
00052    wake all threads. Returns the number of threads actually woken. */
00053 /** \brief  Wake up a number of threads sleeping on an object.
00054
00055     This function wakes up the specified number of threads sleeping on the
00056     object specified.
00057
00058     \param  obj             The object to wake threads that are sleeping on it
00059     \param  cnt             The number of threads to wake, if <= 0, wake all
00060     \param  err             The errno code to set as the errno value on the
00061                             woken threads. If this is 0 (EOK), then the thread's
00062                             errno will not be changed, and the thread will get a
00063                             return value of 0 from the genwait_wait(). If it is
00064                             non-zero, the thread will get a return value of -1
00065                             and errno will be set to this value for the woken
00066                             threads.
00067     \return                 The number of threads woken
00068 */
00069 int genwait_wake_cnt(void * obj, int cnt, int err);
00070
00071 /** \brief  Wake up all threads sleeping on an object.
00072
00073     This function simply calls genwait_wake_cnt(obj, -1, 0).
00074
00075     \param  obj             The object to wake threads that are sleeping on it
00076     \see    genwait_wake_cnt()
00077 */
00078 void genwait_wake_all(void * obj);
00079
00080 /** \brief  Wake up one thread sleeping on an object.
00081
00082     This function simply calls genwait_wake_cnt(obj, 1, 0).
00083
00084     \param  obj             The object to wake threads that are sleeping on it
00085     \see    genwait_wake_cnt()
00086 */
00087 void genwait_wake_one(void * obj);
00088
00089 /** \brief  Wake up all threads sleeping on an object, with an error.
00090
00091     This function simply calls genwait_wake_cnt(obj, -1, err).
00092
00093     \param  obj             The object to wake threads that are sleeping on it
00094     \param  err             The value to set in the threads' errno values
00095     \see    genwait_wake_cnt()
00096 */
00097 void genwait_wake_all_err(void *obj, int err);
00098
```

```
00099 /** \brief  Wake up one thread sleeping on an object, with an error.
00100
00101     This function simply calls genwait_wake_cnt(obj, 1, err).
00102
00103     \param  obj             The object to wake threads that are sleeping on it
00104     \param  err             The value to set in the threads' errno values
00105     \see    genwait_wake_cnt()
00106 */
00107 void genwait_wake_one_err(void *obj, int err);
00108
00109 /** \brief  Wake up a specific thread that is sleeping on an object.
00110
00111     This function wakes up the specfied thread, assuming it is sleeping on the
00112     specified object.
00113
00114     \param  obj             The object to wake the thread from
00115     \param  thd             The specific thread to wake
00116     \param  err             The errno code to set as the errno value on the
00117                             woken thread. If this is 0 (EOK), then the thread's
00118                             errno will not be changed, and the thread will get a
00119                             return value of 0 from the genwait_wait(). If it is
00120                             non-zero, the thread will get a return value of -1
00121                             and errno will be set to this value for the woken
00122                             threads.
00123     \return                 The number of threads woken, which should be 1 on
00124                             success.
00125 */
00126 int genwait_wake_thd(void *obj, kthread_t *thd, int err);
00127
00128 /** \brief  Look for timed out genwait_wait() calls.
00129
00130     There should be no reason you need to call this function, it is called
00131     internally by the scheduler for you.
00132
00133     \param  now             The current system time, in milliseconds since boot
00134 */
00135 void genwait_check_timeouts(uint64 now);
00136
00137 /** \brief  Look for the next timeout event time.
00138
00139     This function looks up when the next genwait_wait() call will timeout. This
00140     function is for the internal use of the scheduler, and should not be called
00141     from user code.
00142
00143     \return                 The next timeout time in milliseconds since boot, or
00144                             0 if there are no pending genwait_wait() calls
00145 */
00146 uint64 genwait_next_timeout(void);
00147
00148 /** \cond */
00149 /* Initialize the genwait system */
00150 int genwait_init(void);
00151
00152 /* Shut down the genwait system */
00153 void genwait_shutdown(void);
00154 /** \endcond */
00155
00156
00157 __END_DECLS
00158
00159 #endif  /* __KOS_GENWAIT_H */
00160
```

9.56 include/kos/init.h File Reference

Initialization-related flags and macros.

```
#include <kos/cdefs.h>
#include <arch/init_flags.h>
#include <kos/init_base.h>
#include <stdint.h>
```

Macros

- #define KOS_INIT_FLAGS(flags) _KOS_INIT_FLAGS(flags, __kos_cplusplus)

  Exports and initializes the given KOS subsystems.
- #define KOS_INIT_ROMDISK(rd)

  Deprecated and not useful anymore.
- #define KOS_INIT_ROMDISK_NONE NULL

  State that you don't want a romdisk.
- #define KOS_INIT_EARLY(func) void (*__kos_init_early_fn)(void) = (func)

  Register a single function to be called very early in the boot process, before the BSS section is cleared.
- #define INIT_DEFAULT

  Default init flags (IRQs on, preemption enabled, romdisks).
- #define INIT_NONE 0x00000000

  Don't init optional things.
- #define INIT_IRQ 0x00000001

  Enable IRQs at startup.
- #define INIT_THD_PREEMPT 0x00000002
- #define INIT_NET 0x00000004

  Enable built-in networking.
- #define INIT_MALLOCSTATS 0x00000008

  Enable malloc statistics.
- #define INIT_QUIET 0x00000010

  Disable dbgio.
- #define INIT_EXPORT 0x00000020

  Export kernel symbols.
- #define INIT_FS_ROMDISK 0x00000040

  Enable support for romdisks.

Variables

- void * __kos_romdisk

  Built-in romdisk. Do not modify this directly!

9.56.1 Detailed Description

Initialization-related flags and macros.

This file provides initialization-related flags and macros that can be used to set up various subsystems of KOS on startup. Only flags that are architecture-independent are specified here, however this file also includes the architecture-specific file to bring in those flags as well.

See also

arch/init_flags.h

kos/init_base.h

Author

Megan Potter

Lawrence Sebald

Paul Cercueil

Falco Girgis

9.56.2 Macro Definition Documentation

KOS_INIT_EARLY

```
#define KOS_INIT_EARLY(
              func ) void (*__kos_init_early_fn)(void) = (func)
```

Register a single function to be called very early in the boot process, before the BSS section is cleared.

Parameters

func	The function to register. The prototype should be void func(void)

KOS_INIT_FLAGS

```
#define KOS_INIT_FLAGS(
              flags )  _KOS_INIT_FLAGS(flags, __kos_cplusplus)
```

Exports and initializes the given KOS subsystems.

KOS_INIT_FLAGS() provides a mechanism through which various components of KOS can be enabled and initialized depending on whether their flag has been included within the list.

Note

When no KOS_INIT_FLAGS() have been explicitly provided, the default flags used by KOS are equivalent to KOS_INIT_FLAGS(INIT_DEFAULT).

Parameters

flags	Parts of KOS to init.

KOS_INIT_ROMDISK

```
#define KOS_INIT_ROMDISK(
              rd )
```

Value:

```
    void *__kos_romdisk = (rd); \
    extern void fs_romdisk_mount_builtin_legacy(void); \
    void (*fs_romdisk_mount_builtin_legacy_weak)(void) = fs_romdisk_mount_builtin_legacy
```

Deprecated and not useful anymore.

KOS_INIT_ROMDISK_NONE

```
#define KOS_INIT_ROMDISK_NONE NULL
```

State that you don't want a romdisk.

9.56.3 Variable Documentation

__kos_romdisk

```
void* __kos_romdisk  [extern]
```

Built-in romdisk. Do not modify this directly!

9.57 init.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    include/kos/init.h
00004    Copyright (C) 2001 Megan Potter
00005    Copyright (C) 2023 Lawrence Sebald
00006    Copyright (C) 2023 Paul Cercueil
00007    Copyright (C) 2023 Falco Girgis
00008
00009 */
00010
00011 /** \file   kos/init.h
00012     \brief  Initialization-related flags and macros.
00013
00014     This file provides initialization-related flags and macros that can be used
00015     to set up various subsystems of KOS on startup. Only flags that are
00016     architecture-independent are specified here, however this file also includes
00017     the architecture-specific file to bring in those flags as well.
00018
00019     \sa     arch/init_flags.h
00020     \sa     kos/init_base.h
00021
00022     \author Megan Potter
00023     \author Lawrence Sebald
00024     \author Paul Cercueil
00025     \author Falco Girgis
00026 */
00027
00028 #ifndef __KOS_INIT_H
00029 #define __KOS_INIT_H
00030
00031 #ifdef __cplusplus
00032 extern "C" {
00033 #endif
00034
00035 #include <kos/cdefs.h>
00036 __BEGIN_DECLS
00037
00038 #include <arch/init_flags.h>
00039 #include <kos/init_base.h>
00040 #include <stdint.h>
00041
00042 /** \cond */
00043 #ifdef __cplusplus
00044 #define __kos_cplusplus 1
00045 #else
00046 #define __kos_cplusplus 0
00047 #endif
00048
00049  #define __KOS_INIT_FLAGS_0(flags) \
```

```
00050      const uint32_t __kos_init_flags = (flags); \
00051      KOS_INIT_FLAG(flags, INIT_NET, arch_init_net); \
00052      KOS_INIT_FLAG(flags, INIT_NET, net_shutdown); \
00053      KOS_INIT_FLAG(flags, INIT_NET, bba_la_init); \
00054      KOS_INIT_FLAG(flags, INIT_NET, bba_la_shutdown); \
00055      KOS_INIT_FLAG(flags, INIT_FS_ROMDISK, fs_romdisk_init); \
00056      KOS_INIT_FLAG(flags, INIT_FS_ROMDISK, fs_romdisk_shutdown); \
00057      KOS_INIT_FLAG(flags, INIT_EXPORT, export_init); \
00058      KOS_INIT_FLAGS_ARCH(flags)
00059
00060 #define __KOS_INIT_FLAGS_1(flags) \
00061     extern "C" { \
00062         __KOS_INIT_FLAGS_0(flags); \
00063     }
00064
00065 #define __KOS_INIT_FLAGS(flags, cp) \
00066     __KOS_INIT_FLAGS_##cp(flags)
00067
00068 #define _KOS_INIT_FLAGS(flags, cp) \
00069     __KOS_INIT_FLAGS(flags, cp)
00070
00071 extern const uint32_t __kos_init_flags;
00072 /** \endcond */
00073
00074 /** \brief  Exports and initializes the given KOS subsystems.
00075
00076     KOS_INIT_FLAGS() provides a mechanism through which various components
00077     of KOS can be enabled and initialized depending on whether their flag
00078     has been included within the list.
00079
00080     \note
00081     When no KOS_INIT_FLAGS() have been explicitly provided, the default
00082     flags used by KOS are equivalent to KOS_INIT_FLAGS(INIT_DEFAULT).
00083
00084     \param  flags           Parts of KOS to init.
00085  */
00086 #define KOS_INIT_FLAGS(flags) \
00087     _KOS_INIT_FLAGS(flags, __kos_cplusplus)
00088
00089 /** \brief  Deprecated and not useful anymore. */
00090 #define KOS_INIT_ROMDISK(rd) \
00091     void *__kos_romdisk = (rd); \
00092     extern void fs_romdisk_mount_builtin_legacy(void); \
00093     void (*fs_romdisk_mount_builtin_legacy_weak)(void) = fs_romdisk_mount_builtin_legacy
00094
00095
00096 /** \brief  Built-in romdisk. Do not modify this directly! */
00097 extern void * __kos_romdisk;
00098
00099 /** \brief  State that you don't want a romdisk. */
00100 #define KOS_INIT_ROMDISK_NONE   NULL
00101
00102 /** \brief  Register a single function to be called very early in the boot
00103             process, before the BSS section is cleared.
00104
00105     \param  func            The function to register. The prototype should be
00106                             void func(void)
00107 */
00108 #define KOS_INIT_EARLY(func) void (*__kos_init_early_fn)(void) = (func)
00109
00110 /** \defgroup kos_initflags     Available flags for initialization
00111
00112     These are the architecture-independent flags that can be specified with
00113     KOS_INIT_FLAGS.
00114
00115     \see    dreamcast_initflags
00116     @{
00117 */
00118 /** \brief  Default init flags (IRQs on, preemption enabled, romdisks). */
00119 #define INIT_DEFAULT    (INIT_IRQ | INIT_THD_PREEMPT | INIT_FS_ROMDISK | \
00120                          INIT_DEFAULT_ARCH)
00121
00122 #define INIT_NONE        0x00000000  /**< \brief Don't init optional things */
00123 #define INIT_IRQ         0x00000001  /**< \brief Enable IRQs at startup */
00124 /* Preemptive mode is the only mode now. Keeping define for compatability. */
00125 #define INIT_THD_PREEMPT 0x00000002  /**< \deprecated Already default mode */
00126 #define INIT_NET         0x00000004  /**< \brief Enable built-in networking */
00127 #define INIT_MALLOCSTATS 0x00000008  /**< \brief Enable malloc statistics */
00128 #define INIT_QUIET       0x00000010  /**< \brief Disable dbgio */
00129 #define INIT_EXPORT      0x00000020  /**< \brief Export kernel symbols */
00130 #define INIT_FS_ROMDISK  0x00000040  /**< \brief Enable support for romdisks */
```

```
00131 /** @} */
00132
00133 __END_DECLS
00134
00135 #ifdef __cplusplus
00136 };
00137 #endif
00138
00139 #endif /* !__KOS_INIT_H */
```

9.58 include/kos/init_base.h File Reference

Shared initialization macros and utilities.

```
#include <kos/cdefs.h>
```

9.58.1 Detailed Description

Shared initialization macros and utilities.

This file contains common utilities which can be included within architecture-specific `init_flags.h` files, providing a base layer of infrastructure for managing initialization flags.

See also

kos/init.h

arch/init_flags.h

Author

Falco Girgis

9.59 init_base.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    include/kos/init_base.h
00004    Copyright (C) 2023 Falco Girgis
00005
00006 */
00007
00008 /** \file   kos/init_base.h
00009     \brief  Shared initialization macros and utilities
00010
00011    This file contains common utilities which can be included within
00012    architecture-specific `init_flags.h` files, providing a base
00013    layer of infrastructure for managing initialization flags.
00014
00015     \sa    kos/init.h
00016     \sa    arch/init_flags.h
00017
00018     \author Falco Girgis
00019 */
00020
00021 #ifndef __KOS_INIT_BASE_H
```

```
00022 #define __KOS_INIT_BASE_H
00023
00024 #include <kos/cdefs.h>
00025 __BEGIN_DECLS
00026
00027 /** \cond */
00028
00029 /* Declares a weak function pointer which can be optionally
00030    overridden and given a value later. */
00031 #define KOS_INIT_FLAG_WEAK(func, dft_on) \
00032     void (*func##_weak)(void) __weak = (dft_on) ? func : NULL
00033
00034 /* Invokes the given function if its weak function pointer
00035    has been overridden to point to a valid function. */
00036 #define KOS_INIT_FLAG_CALL(func) ({ \
00037     int ret = 0; \
00038     if(func##_weak) { \
00039         (*func##_weak)(); \
00040         ret = 1; \
00041     } \
00042     ret; \
00043 })
00044
00045 /* Export the given function pointer by assigning it to the weak function
00046  * pointer if the given flags value contains all the flags set in the mask. */
00047 #define KOS_INIT_FLAG_ALL(flags, mask, func) \
00048     extern void func(void); \
00049     void (*func##_weak)(void) = ((flags) & (mask)) == (mask) ? func : NULL
00050
00051 /* Export the given function by assigning it to the weak function pointer if the
00052    given flags value contains none of the flags set in the mask. */
00053 #define KOS_INIT_FLAG_NONE(flags, mask, func) \
00054     extern void func(void); \
00055     void (*func##_weak)(void) = ((flags) & (mask)) ? NULL : func
00056
00057 /* Export the given function by assigning it to the weak function pointer if the
00058    given flags value contains the selected flag */
00059 #define KOS_INIT_FLAG(flags, mask, func) \
00060     extern void func(void); \
00061     void (*func##_weak)(void) = ((flags) & (mask)) ? func : NULL
00062
00063 /** \endcond */
00064
00065 __END_DECLS
00066
00067 #endif /* !__ARCH_INIT_BASE_H */
```

9.60 include/kos/iovec.h File Reference

Deprecated header file for I/O scatter/gather arrays.

`#include <sys/uio.h>`

9.60.1 Detailed Description

Deprecated header file for I/O scatter/gather arrays.

See <sys/uio.h> for what used to be here.

9.61 iovec.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/iovec.h
00004    Copyright (C) 2001 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/iovec.h
00009     \brief  Deprecated header file for I/O scatter/gather arrays.
00010
00011     See <sys/uio.h> for what used to be here.
00012 */
00013
00014 #ifndef __KOS_IOVEC_H
00015 #define __KOS_IOVEC_H
00016
00017 #include <sys/uio.h>
00018
00019 /** \brief  Compatibility typedef for old code. */
00020 typedef struct iovec iovec_t;
00021
00022 #endif  /* __KOS_IOVEC_H */
```

9.62 include/kos/library.h File Reference

Dynamically loadable library support.

```
#include <sys/cdefs.h>
#include <kos/thread.h>
#include <kos/elf.h>
#include <kos/fs.h>
```

Data Structures

- struct klibrary_t

  Loaded library structure.

Macros

- #define LIBRARY_DEFAULTS 0

  Defaults: no flags.

Typedefs

- typedef tid_t libid_t

  Library ID type.

Functions

- klibrary_t * library_by_libid (libid_t libid)

  Look up a library by ID.
- klibrary_t * library_create (int flags)

  Create a new library shell.
- int library_destroy (klibrary_t *lib)

  Destroy a library.
- klibrary_t * library_open (const char *name, const char *fn)

  Try to open a library by name.
- klibrary_t * library_lookup (const char *name)

  Look up a library by name.
- int library_close (klibrary_t *lib)

  Close a previously opened library.
- libid_t library_get_libid (klibrary_t *lib)

  Retrieve the specified library's runtime-assigned ID.
- int library_get_refcnt (klibrary_t *lib)

  Retrieve the specified library's reference count.
- const char * library_get_name (klibrary_t *lib)

  Retrieve the specified library's name.
- uint32 library_get_version (klibrary_t *lib)

  Retrieve the specified library's version.

9.62.1 Detailed Description

Dynamically loadable library support.

This file contains definitions for accessing loadable libraries at runtime. Each library has a name and a version number that it can be referenced by. One must be careful with these dynamic libraries as there is no private storage per instance, and all memory space is shared.

Libraries can both export and import symbols. Imported symbols may require other libraries to be loaded earlier. Libraries are reference counted so that they can be opened multiple times without actually loading them multiple times, and so that a close will act as expected in situations like this.

Author

Megan Potter

9.62.2 Macro Definition Documentation

LIBRARY_DEFAULTS

```
#define LIBRARY_DEFAULTS 0
```

Defaults: no flags.

9.62.3 Typedef Documentation

libid_t

```
typedef tid_t libid_t
```

Library ID type.

9.62.4 Function Documentation

library_by_libid()

```
klibrary_t * library_by_libid (
             libid_t libid )
```

Look up a library by ID.

This function looks up a library by its library ID.

Parameters

libid	The library ID to look up

Returns

The specified library, or NULL if not found

library_close()

```
int library_close (
             klibrary_t * lib )
```

Close a previously opened library.

This function will close the specified library. This may involve simply decrementing its reference count, however, it may also involve actually closing and freeing the library. Thus, don't try to use the library after calling this without reopening it first.

Parameters

lib	The library to close

Return values

0	On success

Return values

-1	On error, errno may be set to an appropriate code

Error Conditions:

EINVAL - the library is not valid

library_create()

```
klibrary_t * library_create (
             int flags )
```

Create a new library shell.

This function creates a new library, adding it to the list of libraries. You generally should not call this function directly, unless you have some good reason to do so.

Parameters

flags	Flags to create the library with.

Returns

The newly created library, or NULL on error

library_destroy()

```
int library_destroy (
             klibrary_t * lib )
```

Destroy a library.

This function will take a loaded library and destroy it, unloading it completely. Generally, you should not call this function, but rather use library_close() to make sure that you're not closing something that is still in use.

Parameters

lib	The library to close

Return values

0	Upon successfully destroying the library

library_get_libid()

```
libid_t library_get_libid (
            klibrary_t * lib )
```

Retrieve the specified library's runtime-assigned ID.

Parameters

lib	The library to examine

Returns

The library's ID, or -1 on error

Error Conditions:

EINVAL - the library is not valid

library_get_name()

```
const char * library_get_name (
            klibrary_t * lib )
```

Retrieve the specified library's name.

Parameters

lib	The library to examine

Returns

The library's symbolic name, or NULL on error

Error Conditions:

EINVAL - the library is not valid

library_get_refcnt()

```
int library_get_refcnt (
            klibrary_t * lib )
```

Retrieve the specified library's reference count.

Parameters

lib	The library to examine

Returns

The library's reference count, or -1 on error

Error Conditions:

EINVAL - the library is not valid

library_get_version()

```
uint32 library_get_version (
            klibrary_t * lib )
```

Retrieve the specified library's version.

Parameters

lib	The library to examine

Returns

The library's version number, or 0 on error

Error Conditions

EINVAL - the library is not valid

library_lookup()

```
klibrary_t * library_lookup (
            const char * name )
```

Look up a library by name.

This function looks up a library by its symbolic name without trying to actually load or open it. This is useful if you want to open a library but not keep around a handle to it (which isn't necessarily encouraged).

Parameters

name	The name of the library to search for

Returns

The library, if found. NULL if not found, errno set as appropriate.

Error Conditions:

ENOENT - the library was not found

library_open()

```
klibrary_t * library_open (
             const char * name,
             const char * fn )
```

Try to open a library by name.

This function attempts to open a library by its name. If it cannot be found by name, this function will attempt to load the library from the specified filename.

Parameters

name	The symbolic name of the library
fn	The filename to load the library from

Returns

A handle to the library, or NULL on error with errno set as appropriate

Error Conditions:

EINVAL - the library was found or loaded, but invalid
ENOMEM - out of memory
ENOENT - library not found and no filename given

9.63 library.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    include/kos/library.h
00004    Copyright (C)2003 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/library.h
00009     \brief  Dynamically loadable library support.
00010
00011     This file contains definitions for accessing loadable libraries at runtime.
00012     Each library has a name and a version number that it can be referenced by.
00013     One must be careful with these dynamic libraries as there is no private
00014     storage per instance, and all memory space is shared.
00015
```

```
00016     Libraries can both export and import symbols. Imported symbols may require
00017     other libraries to be loaded earlier. Libraries are reference counted so
00018     that they can be opened multiple times without actually loading them
00019     multiple times, and so that a close will act as expected in situations like
00020     this.
00021
00022     \author Megan Potter
00023 */
00024
00025 #ifndef __KOS_LIBRARY_H
00026 #define __KOS_LIBRARY_H
00027
00028 #include <sys/cdefs.h>
00029 __BEGIN_DECLS
00030
00031 #include <kos/thread.h>
00032 #include <kos/elf.h>
00033 #include <kos/fs.h>
00034
00035 /** \cond */
00036 /* Pre-define list/queue types */
00037 struct klibrary;
00038 TAILQ_HEAD(klqueue, klibrary);
00039 LIST_HEAD(kllist, klibrary);
00040 /** \endcond */
00041
00042 /* Thread IDs are ok for us */
00043 typedef tid_t libid_t;                  /**< \brief Library ID type. */
00044
00045 /** \brief  Loaded library structure.
00046
00047     This structure represents a single loaded library. Each library is
00048     essentially a loaded binary of code and a set of exported entry points that
00049     are standardized.
00050
00051     Each loaded library should export at least the functions described in this
00052     structure:
00053     \li     const char *lib_get_name()
00054     \li     uint32 %lib_get_version()
00055     \li     int lib_open(struct klibrary *lib)
00056     \li     int lib_close(struct klibrary *lib)
00057
00058     You should not modify any members of this structure yourself (except if you
00059     are implementing a library).
00060
00061     \headerfile kos/library.h
00062 */
00063 typedef struct klibrary {
00064     /** \brief  Library list handle.
00065
00066         Contrary to what doxygen might think, this is not a function.
00067     */
00068     LIST_ENTRY(klibrary) list;
00069
00070     /** \brief  Library ID (assigned at runtime). */
00071     libid_t libid;
00072
00073     /** \brief  Library flags. */
00074     uint32  flags;
00075
00076     /** \brief  ELF image for this library.
00077
00078         This can be used to look up additional entry points in the library.
00079     */
00080     elf_prog_t image;
00081
00082     /** \brief  Library reference count.
00083
00084         This value is incremented every time the library is opened, and
00085         decremented each time it is closed. Once the library's reference count
00086         hits 0, a close will actually destroy the library.
00087     */
00088     int refcnt;
00089
00090     /* Standard library entry points. Every loaded library must provide
00091        at least these things. */
00092
00093     /** \brief  Retrieve the library's symbolic name.
00094
00095         This function must be implemented by all loadable libraries to fetch the
00096         library's symbolic name. This function must work before calling
```

```
00097         lib_open() on the library.
00098
00099         \return             The library's symbolic name
00100     */
00101     const char * (*lib_get_name)(void);
00102
00103     /** \brief  Retrieve the library's version.
00104
00105         This function must be implemented by all loadble libraries to fetch the
00106         library's version number. This function must work before calling
00107         lib_open() on the library.
00108
00109         \return             The library's version number
00110     */
00111     uint32(*lib_get_version)(void);
00112
00113     /** \brief  Open a library.
00114
00115         This function must be implemented by all loadable libraries to
00116         initialize the library on load. If the library is already opened, this
00117         may only involve increasing the reference count.
00118
00119         \param  lib         The library structure
00120         \return             Values >= 0 indicate success, < 0 indicates failure.
00121                             A failure on the first lib_open is indicative that
00122                             the library should be removed from memory.
00123     */
00124     int (*lib_open)(struct klibrary * lib);
00125
00126     /** \brief  Close an opened library.
00127
00128         This function must be implemented by all loadable libraries to close and
00129         deinitialize a library. If the library's reference count is > 1 when
00130         this function is called, this may involve simply decrementing the
00131         reference count.
00132
00133         \param  lib         The library structure
00134         \return             Values >= 0 indicate success, < 0 indicates failure
00135     */
00136     int (*lib_close)(struct klibrary * lib);
00137 } klibrary_t;
00138
00139 /* Library flag values */
00140 #define LIBRARY_DEFAULTS    0           /**< \brief Defaults: no flags */
00141
00142 /** \cond */
00143 /* Library list; note: do not manipulate directly */
00144 extern struct kllist library_list;
00145 /** \endcond */
00146
00147 /** \brief  Look up a library by ID.
00148
00149     This function looks up a library by its library ID.
00150
00151     \param  libid           The library ID to look up
00152     \return                 The specified library, or NULL if not found
00153 */
00154 klibrary_t *library_by_libid(libid_t libid);
00155
00156 /** \brief  Create a new library shell.
00157
00158     This function creates a new library, adding it to the list of libraries. You
00159     generally should not call this function directly, unless you have some good
00160     reason to do so.
00161
00162     \param  flags           Flags to create the library with.
00163     \return                 The newly created library, or NULL on error
00164 */
00165 klibrary_t *library_create(int flags);
00166
00167 /** \brief  Destroy a library.
00168
00169     This function will take a loaded library and destroy it, unloading it
00170     completely. Generally, you should not call this function, but rather use
00171     library_close() to make sure that you're not closing something that is still
00172     in use.
00173
00174     \param  lib             The library to close
00175     \retval 0               Upon successfully destroying the library
00176 */
00177 int library_destroy(klibrary_t *lib);
```

```
00178
00179 /** \brief  Try to open a library by name.
00180
00181     This function attempts to open a library by its name. If it cannot be found
00182     by name, this function will attempt to load the library from the specified
00183     filename.
00184
00185     \param  name            The symbolic name of the library
00186     \param  fn              The filename to load the library from
00187     \return                 A handle to the library, or NULL on error with errno
00188                             set as appropriate
00189
00190     \par    Error Conditions:
00191     \em     EINVAL - the library was found or loaded, but invalid \n
00192     \em     ENOMEM - out of memory \n
00193     \em     ENOENT - library not found and no filename given
00194 */
00195 klibrary_t * library_open(const char * name, const char * fn);
00196
00197 /** \brief  Look up a library by name.
00198
00199     This function looks up a library by its symbolic name without trying to
00200     actually load or open it. This is useful if you want to open a library but
00201     not keep around a handle to it (which isn't necessarily encouraged).
00202
00203     \param  name            The name of the library to search for
00204     \return                 The library, if found. NULL if not found, errno set
00205                             as appropriate.
00206
00207     \par    Error Conditions:
00208     \em     ENOENT - the library was not found
00209 */
00210 klibrary_t * library_lookup(const char * name);
00211
00212 /** \brief  Close a previously opened library.
00213
00214     This function will close the specified library. This may involve simply
00215     decrementing its reference count, however, it may also involve actually
00216     closing and freeing the library. Thus, don't try to use the library after
00217     calling this without reopening it first.
00218
00219     \param  lib             The library to close
00220     \retval 0               On success
00221     \retval -1              On error, errno may be set to an appropriate code
00222
00223     \par    Error Conditions:
00224     \em     EINVAL - the library is not valid
00225 */
00226 int library_close(klibrary_t * lib);
00227
00228 /** \brief  Retrieve the specified library's runtime-assigned ID.
00229     \param  lib             The library to examine
00230     \return                 The library's ID, or -1 on error
00231
00232     \par    Error Conditions:
00233     \em     EINVAL - the library is not valid
00234 */
00235 libid_t library_get_libid(klibrary_t * lib);
00236
00237 /** \brief  Retrieve the specified library's reference count.
00238     \param  lib             The library to examine
00239     \return                 The library's reference count, or -1 on error
00240
00241     \par    Error Conditions:
00242     \em     EINVAL - the library is not valid
00243 */
00244 int library_get_refcnt(klibrary_t * lib);
00245
00246 /** \brief  Retrieve the specified library's name.
00247     \param  lib             The library to examine
00248     \return                 The library's symbolic name, or NULL on error
00249
00250     \par    Error Conditions:
00251     \em     EINVAL - the library is not valid
00252 */
00253 const char * library_get_name(klibrary_t * lib);
00254
00255 /** \brief  Retrieve the specified library's version.
00256     \param  lib             The library to examine
00257     \return                 The library's version number, or 0 on error
00258
```

```
00259     \par    Error Conditions
00260     \em     EINVAL - the library is not valid
00261 */
00262 uint32 library_get_version(klibrary_t * lib);
00263
00264 /** \cond */
00265 /* Init */
00266 int library_init(void);
00267
00268 /* Shutdown */
00269 void library_shutdown(void);
00270 /** \endcond */
00271
00272 __END_DECLS
00273
00274 #endif  /* __KOS_LIBRARY_H */
00275
```

9.64 include/kos/limits.h File Reference

Limits.

Macros

- #define NAME_MAX 256

  Max filename length.
- #define MAX_FN_LEN NAME_MAX
- #define PATH_MAX 4096

  Max path length.
- #define SYMLOOP_MAX 16

  Max number of symlinks resolved.

9.64.1 Detailed Description

Limits.

This file contains definitions of limits of various things.

Author

Megan Potter

9.64.2 Macro Definition Documentation

MAX_FN_LEN

```
#define MAX_FN_LEN NAME_MAX
```

NAME_MAX

```
#define NAME_MAX 256
```

Max filename length.

PATH_MAX

```
#define PATH_MAX 4096
```

Max path length.

SYMLOOP_MAX

```
#define SYMLOOP_MAX 16
```

Max number of symlinks resolved.

9.65 limits.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/limits.h
00004    (c)2000-2001 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/limits.h
00009     \brief  Limits.
00010
00011     This file contains definitions of limits of various things.
00012
00013     \author Megan Potter
00014 */
00015
00016 #ifndef __KOS_LIMITS_H
00017 #define __KOS_LIMITS_H
00018
00019 #ifndef NAME_MAX
00020 #define NAME_MAX    256     /**< \brief Max filename length */
00021 #endif
00022
00023 /* MAX_FN_LEN defined for legacy code compatibility */
00024 #ifndef MAX_FN_LEN
00025 #define MAX_FN_LEN  NAME_MAX
00026 #endif
00027
00028 #ifndef PATH_MAX
00029 #define PATH_MAX    4096    /**< \brief Max path length */
00030 #endif
00031
00032 #ifndef SYMLOOP_MAX
00033 #define SYMLOOP_MAX 16      /**< \brief Max number of symlinks resolved */
00034 #endif
00035
00036 #endif  /* __KOS_LIMITS_H */
```

9.66 include/kos/mutex.h File Reference

Mutual exclusion locks.

```
#include <kos/cdefs.h>
#include <kos/thread.h>
```

Data Structures

- struct mutex_t

  Mutual exclusion lock type.

Macros

- #define MUTEX_INITIALIZER { MUTEX_TYPE_NORMAL, 0, NULL, 0 }

  Initializer for a transient mutex.
- #define ERRORCHECK_MUTEX_INITIALIZER { MUTEX_TYPE_ERRORCHECK, 0, NULL, 0 }

  Initializer for a transient error-checking mutex.
- #define RECURSIVE_MUTEX_INITIALIZER { MUTEX_TYPE_RECURSIVE, 0, NULL, 0 }

  Initializer for a transient recursive mutex.

Mutex types

Types of Mutexes supported by KOS

The values defined in here are the various types of mutexes that KallistiOS supports.

- #define MUTEX_TYPE_NORMAL 0

  Normal mutex type.
- #define MUTEX_TYPE_OLDNORMAL 1

  Alias for MUTEX_TYPE_NORMAL.
- #define MUTEX_TYPE_ERRORCHECK 2

  Error-checking mutex type.
- #define MUTEX_TYPE_RECURSIVE 3

  Recursive mutex type.
- #define MUTEX_TYPE_DEFAULT MUTEX_TYPE_NORMAL

  Default mutex type.

Functions

- mutex_t * mutex_create (void) __depr("Use mutex_init or an initializer.")

  Allocate a new mutex.
- int mutex_init (mutex_t *m, int mtype)

  Initialize a new mutex.
- int mutex_destroy (mutex_t *m)

  Destroy a mutex.
- int mutex_lock (mutex_t *m)

  Lock a mutex.

- int mutex_lock_timed (mutex_t *m, int timeout)

  Lock a mutex (with a timeout).
- int mutex_is_locked (mutex_t *m)

  Check if a mutex is locked.
- int mutex_trylock (mutex_t *m)

  Attempt to lock a mutex.
- int mutex_unlock (mutex_t *m)

  Unlock a mutex.
- int mutex_unlock_as_thread (mutex_t *m, kthread_t *thd)

  Unlock a mutex under another thread's authority.

9.66.1 Detailed Description

Mutual exclusion locks.

This file defines mutual exclusion locks (or mutexes for short). The concept of a mutex is one of the most common types of locks in a multi-threaded environment. Mutexes do exactly what they sound like, they keep two (or more) threads mutually exclusive from one another. A mutex is used around a block of code to prevent two threads from interfereing with one another when only one would be appropriate to be in the block at a time.

KallistiOS implments 3 types of mutexes, to bring it roughly in-line with POSIX. The types of mutexes that can be made are normal, error-checking, and recursive. Each has its own strengths and weaknesses, which are briefly discussed below.

A normal mutex (MUTEX_TYPE_NORMAL) is the fastest and simplest mutex of the bunch. This is roughly equivalent to a semaphore that has been initialized with a count of 1. There is no protection against threads unlocking normal mutexes they didn't lock, nor is there any protection against deadlocks that would arise from locking the mutex twice.

An error-checking mutex (MUTEX_TYPE_ERRORCHECK) adds a small amount of error checking on top of a normal mutex. This type will not allow you to lock the mutex twice (it will return an error if the same thread tries to lock it two times so it will not deadlock), and it will not allow a different thread to unlock the mutex if it isn't the one holding the lock.

A recursive mutex (MUTEX_TYPE_RECURSIVE) extends the error checking mutex by allowing you to lock the mutex twice in the same thread, but you must also unlock it twice (this works for any number of locks – lock it n times, you must unlock it n times). Still only one thread can hold the lock, but it may hold it as many times as it needs to. This is equivalent to the recursive_lock_t type that was available in KallistiOS for a while (before it was basically merged back into a normal mutex).

There is a fourth type of mutex defined (MUTEX_TYPE_DEFAULT), which maps to the MUTEX_TYPE_NORMAL type. This is simply for alignment with POSIX.

Author

Lawrence Sebald

See also

kos/sem.h

9.66.2 Macro Definition Documentation

ERRORCHECK_MUTEX_INITIALIZER

```
#define ERRORCHECK_MUTEX_INITIALIZER { MUTEX_TYPE_ERRORCHECK, 0, NULL, 0 }
```

Initializer for a transient error-checking mutex.

MUTEX_INITIALIZER

```
#define MUTEX_INITIALIZER { MUTEX_TYPE_NORMAL, 0, NULL, 0 }
```

Initializer for a transient mutex.

MUTEX_TYPE_DEFAULT

```
#define MUTEX_TYPE_DEFAULT MUTEX_TYPE_NORMAL
```

Default mutex type.

MUTEX_TYPE_ERRORCHECK

```
#define MUTEX_TYPE_ERRORCHECK 2
```

Error-checking mutex type.

MUTEX_TYPE_NORMAL

```
#define MUTEX_TYPE_NORMAL 0
```

Normal mutex type.

MUTEX_TYPE_OLDNORMAL

```
#define MUTEX_TYPE_OLDNORMAL 1
```

Alias for MUTEX_TYPE_NORMAL.

MUTEX_TYPE_RECURSIVE

```
#define MUTEX_TYPE_RECURSIVE 3
```

Recursive mutex type.

RECURSIVE_MUTEX_INITIALIZER

```
#define RECURSIVE_MUTEX_INITIALIZER { MUTEX_TYPE_RECURSIVE, 0, NULL, 0 }
```

Initializer for a transient recursive mutex.

9.66.3 Function Documentation

mutex_create()

```
mutex_t * mutex_create (
            void  )
```

Allocate a new mutex.

Deprecated This function allocates and initializes a new mutex for use. This function will always create mutexes of the type MUTEX_TYPE_NORMAL.

Returns

The newly created mutex on success, or NULL on failure (errno will be set as appropriate).

Note

This function is formally deprecated. It should not be used in any future code, and may be removed in the future. You should instead use mutex_init().

mutex_destroy()

```
int mutex_destroy (
            mutex_t * m )
```

Destroy a mutex.

This function destroys a mutex, releasing any memory that may have been allocated internally for it. It is your responsibility to make sure that all threads waiting on the mutex are taken care of before destroying the mutex.

This function can be called on statically initialized as well as dyanmically initialized mutexes.

Return values

0	On success
-1	On error, errno will be set as appropriate

Error Conditions:

EBUSY - the mutex is currently locked

mutex_init()

```
int mutex_init (
              mutex_t * m,
              int mtype )
```

Initialize a new mutex.

This function initializes a new mutex for use.

Parameters

m	The mutex to initialize
mtype	The type of the mutex to initialize it to

Return values

0	On success
-1	On error, errno will be set as appropriate

Error Conditions:

EINVAL - an invalid type of mutex was specified

See also

mutex_types

mutex_is_locked()

```
int mutex_is_locked (
              mutex_t * m )
```

Check if a mutex is locked.

This function will check whether or not a mutex is currently locked. This is not a thread-safe way to determine if the mutex will be locked by the time you get around to doing it. If you wish to attempt to lock a mutex without blocking, look at mutex_trylock(), not this.

Parameters

m	The mutex to check

Return values

0	If the mutex is not currently locked
1	If the mutex is currently locked

mutex_lock()

```
int mutex_lock (
              mutex_t * m )
```

Lock a mutex.

This function will lock a mutex, if it is not already locked by another thread. If it is locked by another thread already, this function will block until the mutex has been acquired for the calling thread.

The semantics of this function depend on the type of mutex that is used.

Parameters

m	The mutex to acquire

Return values

0	On success
-1	On error, sets errno as appropriate

Error Conditions:

EPERM - called inside an interrupt
EINVAL - the mutex has not been initialized properly
EAGAIN - lock has been acquired too many times (recursive)
EDEADLK - would deadlock (error-checking)

mutex_lock_timed()

```
int mutex_lock_timed (
              mutex_t * m,
              int timeout )
```

Lock a mutex (with a timeout).

This function will attempt to lock a mutex. If the lock can be acquired immediately, the function will return immediately. If not, the function will block for up to the specified number of milliseconds to wait for the lock. If the lock cannot be acquired in this timeframe, this function will return an error.

Parameters

m	The mutex to acquire
timeout	The number of milliseconds to wait for the lock

Return values

0	On success
-1	On error, errno will be set as appropriate

Error Conditions:

EPERM - called inside an interrupt
EINVAL - the mutex has not been initialized properly
EINVAL - the timeout value was invalid (less than 0)
ETIMEDOUT - the timeout expired
EAGAIN - lock has been acquired too many times (recursive)
EDEADLK - would deadlock (error-checking)

mutex_trylock()

```
int mutex_trylock (
            mutex_t * m )
```

Attempt to lock a mutex.

This function will attempt to acquire the mutex for the calling thread, returning immediately whether or not it could be acquired. If the mutex cannot be acquired, an error will be returned.

This function is safe to call inside an interrupt.

Parameters

m	The mutex to attempt to acquire

Return values

0	On successfully acquiring the mutex
-1	If the mutex cannot be acquired without blocking

Error Conditions:

EAGAIN - the mutex is already locked (mutex_lock() would block)
EINVAL - the mutex has not been initialized properly
EAGAIN - lock has been acquired too many times (recursive)
EDEADLK - would deadlock (error-checking)

mutex_unlock()

```
int mutex_unlock (
              mutex_t * m )
```

Unlock a mutex.

This function will unlock a mutex, allowing other threads to acquire it. The semantics of this operation depend on the mutex type in use.

Parameters

m	The mutex to unlock

Return values

0	On success
-1	On error, errno will be set as appropriate.

Error Conditions:

EPERM - the current thread does not own the mutex (error-checking or recursive)

mutex_unlock_as_thread()

```
int mutex_unlock_as_thread (
              mutex_t * m,
              kthread_t * thd )
```

Unlock a mutex under another thread's authority.

This function allows an IRQ handler to unlock a mutex that was locked by a normal kernel thread. This function is only for use in IRQ handlers, so it will generally not be of much use outside of the kernel itself.

Parameters

m	The mutex to unlock
thd	The thread owning the mutex

Return values

0	On success
-1	On error, errno will be set as appropriate.

Error Conditions:

EPERM - the specified thread does not own the mutex
EACCES - called outside an IRQ handler

9.67 mutex.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    include/kos/mutex.h
00004    Copyright (C) 2001, 2003 Megan Potter
00005    Copyright (C) 2012, 2015 Lawrence Sebald
00006
00007 */
00008
00009 /** \file   kos/mutex.h
00010     \brief  Mutual exclusion locks.
00011     \ingroup kthreads
00012
00013     This file defines mutual exclusion locks (or mutexes for short). The concept
00014     of a mutex is one of the most common types of locks in a multi-threaded
00015     environment. Mutexes do exactly what they sound like, they keep two (or
00016     more) threads mutually exclusive from one another. A mutex is used around
00017     a block of code to prevent two threads from interfereing with one another
00018     when only one would be appropriate to be in the block at a time.
00019
00020     KallistiOS implments 3 types of mutexes, to bring it roughly in-line with
00021     POSIX. The types of mutexes that can be made are normal, error-checking, and
00022     recursive. Each has its own strengths and weaknesses, which are briefly
00023     discussed below.
00024
00025     A normal mutex (MUTEX_TYPE_NORMAL) is the fastest and simplest mutex of the
00026     bunch. This is roughly equivalent to a semaphore that has been initialized
00027     with a count of 1. There is no protection against threads unlocking normal
00028     mutexes they didn't lock, nor is there any protection against deadlocks that
00029     would arise from locking the mutex twice.
00030
00031     An error-checking mutex (MUTEX_TYPE_ERRORCHECK) adds a small amount of error
00032     checking on top of a normal mutex. This type will not allow you to lock the
00033     mutex twice (it will return an error if the same thread tries to lock it two
00034     times so it will not deadlock), and it will not allow a different thread to
00035     unlock the mutex if it isn't the one holding the lock.
00036
00037     A recursive mutex (MUTEX_TYPE_RECURSIVE) extends the error checking mutex
00038     by allowing you to lock the mutex twice in the same thread, but you must
00039     also unlock it twice (this works for any number of locks -- lock it n times,
00040     you must unlock it n times). Still only one thread can hold the lock, but it
00041     may hold it as many times as it needs to. This is equivalent to the
00042     recursive_lock_t type that was available in KallistiOS for a while (before
00043     it was basically merged back into a normal mutex).
00044
00045     There is a fourth type of mutex defined (MUTEX_TYPE_DEFAULT), which maps to
00046     the MUTEX_TYPE_NORMAL type. This is simply for alignment with POSIX.
00047
00048     \author Lawrence Sebald
00049     \see    kos/sem.h
00050 */
00051
00052 #ifndef __KOS_MUTEX_H
00053 #define __KOS_MUTEX_H
00054
00055 #include <kos/cdefs.h>
00056
00057 __BEGIN_DECLS
00058
00059 #include <kos/thread.h>
00060
00061 /** \brief  Mutual exclusion lock type.
00062
00063     All members of this structure should be considered to be private. It is
00064     unsafe to change anything in here yourself.
00065
00066     \headerfile kos/mutex.h
00067 */
00068 typedef struct kos_mutex {
```

```
00069     int type;
00070     int dynamic;
00071     kthread_t *holder;
00072     int count;
00073 } mutex_t;
00074
00075 /** \name  Mutex types
00076     \brief Types of Mutexes supported by KOS
00077
00078     The values defined in here are the various types of mutexes that KallistiOS
00079     supports.
00080
00081     @{
00082 */
00083 #define MUTEX_TYPE_NORMAL       0   /**< \brief Normal mutex type */
00084 #define MUTEX_TYPE_OLDNORMAL    1   /**< \brief Alias for MUTEX_TYPE_NORMAL */
00085 #define MUTEX_TYPE_ERRORCHECK   2   /**< \brief Error-checking mutex type */
00086 #define MUTEX_TYPE_RECURSIVE    3   /**< \brief Recursive mutex type */
00087
00088 /** \brief Default mutex type */
00089 #define MUTEX_TYPE_DEFAULT      MUTEX_TYPE_NORMAL
00090 /** @} */
00091
00092 /** \brief  Initializer for a transient mutex. */
00093 #define MUTEX_INITIALIZER               { MUTEX_TYPE_NORMAL, 0, NULL, 0 }
00094
00095 /** \brief  Initializer for a transient error-checking mutex. */
00096 #define ERRORCHECK_MUTEX_INITIALIZER    { MUTEX_TYPE_ERRORCHECK, 0, NULL, 0 }
00097
00098 /** \brief  Initializer for a transient recursive mutex. */
00099 #define RECURSIVE_MUTEX_INITIALIZER     { MUTEX_TYPE_RECURSIVE, 0, NULL, 0 }
00100
00101 /** \brief  Allocate a new mutex.
00102
00103     \deprecated
00104     This function allocates and initializes a new mutex for use. This function
00105     will always create mutexes of the type MUTEX_TYPE_NORMAL.
00106
00107     \return                 The newly created mutex on success, or NULL on
00108                             failure (errno will be set as appropriate).
00109
00110     \note                   This function is formally deprecated. It should not
00111                             be used in any future code, and may be removed in
00112                             the future. You should instead use mutex_init().
00113 */
00114 mutex_t *mutex_create(void) __depr("Use mutex_init or an initializer.");
00115
00116 /** \brief  Initialize a new mutex.
00117
00118     This function initializes a new mutex for use.
00119
00120     \param  m               The mutex to initialize
00121     \param  mtype           The type of the mutex to initialize it to
00122
00123     \retval 0               On success
00124     \retval -1              On error, errno will be set as appropriate
00125
00126     \par    Error Conditions:
00127     \em     EINVAL - an invalid type of mutex was specified
00128
00129     \sa     mutex_types
00130 */
00131 int mutex_init(mutex_t *m, int mtype);
00132
00133 /** \brief  Destroy a mutex.
00134
00135     This function destroys a mutex, releasing any memory that may have been
00136     allocated internally for it. It is your responsibility to make sure that all
00137     threads waiting on the mutex are taken care of before destroying the mutex.
00138
00139     This function can be called on statically initialized as well as dyanmically
00140     initialized mutexes.
00141
00142     \retval 0               On success
00143     \retval -1              On error, errno will be set as appropriate
00144
00145     \par    Error Conditions:
00146     \em     EBUSY - the mutex is currently locked
00147 */
00148 int mutex_destroy(mutex_t *m);
00149
```

```
00150 /** \brief  Lock a mutex.
00151
00152     This function will lock a mutex, if it is not already locked by another
00153     thread. If it is locked by another thread already, this function will block
00154     until the mutex has been acquired for the calling thread.
00155
00156     The semantics of this function depend on the type of mutex that is used.
00157
00158     \param  m               The mutex to acquire
00159     \retval 0               On success
00160     \retval -1              On error, sets errno as appropriate
00161
00162     \par    Error Conditions:
00163     \em     EPERM - called inside an interrupt \n
00164     \em     EINVAL - the mutex has not been initialized properly \n
00165     \em     EAGAIN - lock has been acquired too many times (recursive) \n
00166     \em     EDEADLK - would deadlock (error-checking)
00167 */
00168 int mutex_lock(mutex_t *m);
00169
00170 /** \brief  Lock a mutex (with a timeout).
00171
00172     This function will attempt to lock a mutex. If the lock can be acquired
00173     immediately, the function will return immediately. If not, the function will
00174     block for up to the specified number of milliseconds to wait for the lock.
00175     If the lock cannot be acquired in this timeframe, this function will return
00176     an error.
00177
00178     \param  m               The mutex to acquire
00179     \param  timeout         The number of milliseconds to wait for the lock
00180     \retval 0               On success
00181     \retval -1              On error, errno will be set as appropriate
00182
00183     \par    Error Conditions:
00184     \em     EPERM - called inside an interrupt \n
00185     \em     EINVAL - the mutex has not been initialized properly \n
00186     \em     EINVAL - the timeout value was invalid (less than 0) \n
00187     \em     ETIMEDOUT - the timeout expired \n
00188     \em     EAGAIN - lock has been acquired too many times (recursive) \n
00189     \em     EDEADLK - would deadlock (error-checking)
00190 */
00191 int mutex_lock_timed(mutex_t *m, int timeout);
00192
00193 /** \brief  Check if a mutex is locked.
00194
00195     This function will check whether or not a mutex is currently locked. This is
00196     not a thread-safe way to determine if the mutex will be locked by the time
00197     you get around to doing it. If you wish to attempt to lock a mutex without
00198     blocking, look at mutex_trylock(), not this.
00199
00200     \param  m               The mutex to check
00201     \retval 0               If the mutex is not currently locked
00202     \retval 1               If the mutex is currently locked
00203 */
00204 int mutex_is_locked(mutex_t *m);
00205
00206 /** \brief  Attempt to lock a mutex.
00207
00208     This function will attempt to acquire the mutex for the calling thread,
00209     returning immediately whether or not it could be acquired. If the mutex
00210     cannot be acquired, an error will be returned.
00211
00212     This function is safe to call inside an interrupt.
00213
00214     \param  m               The mutex to attempt to acquire
00215     \retval 0               On successfully acquiring the mutex
00216     \retval -1              If the mutex cannot be acquired without blocking
00217
00218     \par    Error Conditions:
00219     \em     EAGAIN - the mutex is already locked (mutex_lock() would block) \n
00220     \em     EINVAL - the mutex has not been initialized properly \n
00221     \em     EAGAIN - lock has been acquired too many times (recursive) \n
00222     \em     EDEADLK - would deadlock (error-checking)
00223 */
00224 int mutex_trylock(mutex_t *m);
00225
00226 /** \brief  Unlock a mutex.
00227
00228     This function will unlock a mutex, allowing other threads to acquire it.
00229     The semantics of this operation depend on the mutex type in use.
00230
```

```
00231     \param  m               The mutex to unlock
00232     \retval 0               On success
00233     \retval -1              On error, errno will be set as appropriate.
00234
00235     \par    Error Conditions:
00236     \em     EPERM - the current thread does not own the mutex (error-checking or
00237                     recursive)
00238 */
00239 int mutex_unlock(mutex_t *m);
00240
00241 /** \brief  Unlock a mutex under another thread's authority.
00242
00243     This function allows an IRQ handler to unlock a mutex that was locked by a
00244     normal kernel thread. This function is only for use in IRQ handlers, so it
00245     will generally not be of much use outside of the kernel itself.
00246
00247     \param  m               The mutex to unlock
00248     \param  thd             The thread owning the mutex
00249     \retval 0               On success
00250     \retval -1              On error, errno will be set as appropriate.
00251
00252     \par    Error Conditions:
00253     \em     EPERM - the specified thread does not own the mutex \n
00254     \em     EACCES - called outside an IRQ handler
00255 */
00256 int mutex_unlock_as_thread(mutex_t *m, kthread_t *thd);
00257
00258 __END_DECLS
00259
00260 #endif  /* __KOS_MUTEX_H */
```

9.68 include/kos/net.h File Reference

Network support.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <sys/queue.h>
#include <netinet/in.h>
```

Data Structures

- struct netif_t

  Structure describing one usable network device.
- struct ip_hdr_t

  IPv4 Packet header.
- struct ipv6_hdr_t

  IPv6 Packet header.
- struct net_ipv4_stats_t

  IPv4 statistics structure.
- struct net_ipv6_stats_t

  IPv6 statistics structure.
- struct net_udp_stats_t

  UDP statistics structure.

Macros

- #define NETIF_NO_FLAGS 0x00000000

  No flags set.
- #define NETIF_REGISTERED 0x00000001

  Is it registered?
- #define NETIF_DETECTED 0x00000002

  Is it detected?
- #define NETIF_INITIALIZED 0x00000004

  Has it been initialized?
- #define NETIF_RUNNING 0x00000008

  Has start() been called?
- #define NETIF_PROMISC 0x00010000

  Promiscuous mode.
- #define NETIF_NEEDSPOLL 0x01000000

  Needs to be polled for input.
- #define NETIF_NOETH 0x10000000

  Does not use ethernet.
- #define NETIF_TX_OK 0

  Tx success.
- #define NETIF_TX_ERROR -1

  Tx general error.
- #define NETIF_TX_AGAIN -2

  Retry Tx later.
- #define NETIF_NOBLOCK 0

  Don't block for Tx.
- #define NETIF_BLOCK 1

  Blocking is OK for Tx.
- #define ICMP_PROTOCOL_UNREACHABLE 2

  Protocol unreachable.
- #define ICMP_PORT_UNREACHABLE 3

  Port unreachable.
- #define ICMP_REASSEMBLY_TIME_EXCEEDED 1

  Reassembly time gone.
- #define ICMP6_DEST_UNREACH_NO_ROUTE 0

  No route available.
- #define ICMP6_DEST_UNREACH_PROHIBITED 1

  Access prohibited.
- #define ICMP6_DEST_UNREACH_BEYOND_SCOPE 2

  Gone beyond scope.
- #define ICMP6_DEST_UNREACH_ADDR_UNREACH 3

  Address unreachable.
- #define ICMP6_DEST_UNREACH_PORT_UNREACH 4

  Port unreachable.
- #define ICMP6_DEST_UNREACH_FAIL_EGRESS 5

  Egress failure.
- #define ICMP6_DEST_UNREACH_BAD_ROUTE 6

Bad route specified.

- #define ICMP6_TIME_EXCEEDED_HOPS_EXC 0

  Hops exceeded.
- #define ICMP6_TIME_EXCEEDED_FRAGMENT 1

  Reassembly time gone.
- #define ICMP6_PARAM_PROB_BAD_HEADER 0

  Malformed header.
- #define ICMP6_PARAM_PROB_UNK_HEADER 1

  Unknown header.
- #define ICMP6_PARAM_PROB_UNK_OPTION 2

  Unknown header option.

Typedefs

- typedef int(∗ net_input_func) (netif_t ∗nif, const uint8 ∗pkt, int len)

  Network input callback type.
- typedef void(∗ net_echo_cb) (const uint8 ∗ip, uint16 seq, uint64 delta_us, uint8 ttl, const uint8 ∗data, size_t len)

  ICMPv4 echo reply callback type.
- typedef void(∗ net6_echo_cb) (const struct in6_addr ∗ip, uint16 seq, uint64 delta_us, uint8 hlim, const uint8 ∗data, size_t len)

  ICMPv6 echo reply callback type.

Functions

- int net_arp_init (void)

  Init ARP.
- void net_arp_shutdown (void)

  Shutdown ARP.
- int net_arp_insert (netif_t ∗nif, const uint8 mac[6], const uint8 ip[4], uint64 timestamp)

  Add an entry to the ARP cache manually.
- int net_arp_lookup (netif_t ∗nif, const uint8 ip_in[4], uint8 mac_out[6], const ip_hdr_t ∗pkt, const uint8 ∗data, int data_size)

  Look up an entry from the ARP cache.
- int net_arp_revlookup (netif_t ∗nif, uint8 ip_out[4], const uint8 mac_in[6])

  Do a reverse ARP lookup.
- int net_arp_input (netif_t ∗nif, const uint8 ∗pkt, int len)

  Receive an ARP packet and process it (called by net_input).
- int net_arp_query (netif_t ∗nif, const uint8 ip[4])

  Generate an ARP who-has query on the given device.
- int net_input (netif_t ∗device, const uint8 ∗data, int len)

  Device drivers should call this function to submit packets received in the background.
- net_input_func net_input_set_target (net_input_func t)

  Setup a network input target.
- int net_icmp_send_echo (netif_t ∗net, const uint8 ipaddr[4], uint16 ident, uint16 seq, const uint8 ∗data, size_t size)

  Send an ICMP Echo packet to the specified IP.

- int net_icmp_send_dest_unreach (netif_t *net, uint8 code, const uint8 *msg)

  Send an ICMP Destination Unreachable packet in reply to the given message.
- int net_icmp_send_time_exceeded (netif_t *net, uint8 code, const uint8 *msg)

  Send an ICMP Time Exceeded packet in reply to the given message.
- net_ipv4_stats_t net_ipv4_get_stats (void)

  Retrieve statistics from the IPv4 layer.
- uint32 net_ipv4_address (const uint8 addr[4])

  Create a 32-bit IP address, based on the individual numbers contained within the IP.
- void net_ipv4_parse_address (uint32 addr, uint8 out[4])

  Parse an IP address that is packet into a uint32 into an array of the individual bytes.
- int net_icmp6_send_echo (netif_t *net, const struct in6_addr *dst, uint16 ident, uint16 seq, const uint8 *data, size_t size)

  Send an ICMPv6 Echo (PING6) packet to the specified device.
- int net_icmp6_send_nsol (netif_t *net, const struct in6_addr *dst, const struct in6_addr *target, int dupdet)

  Send a Neighbor Solicitation packet on the specified device.
- int net_icmp6_send_nadv (netif_t *net, const struct in6_addr *dst, const struct in6_addr *target, int sol)

  Send a Neighbor Advertisement packet on the specified device.
- int net_icmp6_send_rsol (netif_t *net)

  Send a Router Solicitation request on the specified interface.
- int net_icmp6_send_dest_unreach (netif_t *net, uint8 code, const uint8 *ppkt, size_t psz)

  Send a destination unreachable packet on the specified interface.
- int net_icmp6_send_time_exceeded (netif_t *net, uint8 code, const uint8 *ppkt, size_t psz)

  Send a time exceeded message on the specified interface.
- int net_icmp6_send_param_prob (netif_t *net, uint8 code, uint32 ptr, const uint8 *ppkt, size_t psz)

  Send an ICMPv6 Parameter Problem about the given packet.
- net_ipv6_stats_t net_ipv6_get_stats (void)

  Retrieve statistics from the IPv6 layer.
- int net_ndp_init (void)

  Init NDP.
- void net_ndp_shutdown (void)

  Shutdown NDP.
- void net_ndp_gc (void)

  Garbage collect timed out NDP entries. This will be called periodically as NDP queries come in.
- int net_ndp_insert (netif_t *nif, const uint8 mac[6], const struct in6_addr *ip, int unsol)

  Add an entry to the NDP cache.
- int net_ndp_lookup (netif_t *net, const struct in6_addr *ip, uint8 mac_out[6], const ipv6_hdr_t *pkt, const uint8 *data, int data_size)

  Look up an entry from the NDP cache.
- net_udp_stats_t net_udp_get_stats (void)

  Retrieve statistics from the UDP layer.
- int net_udp_init (void)

  Init UDP.
- void net_udp_shutdown (void)

  Shutdown UDP.
- int net_tcp_init (void)

  Init TCP.
- void net_tcp_shutdown (void)

Shutdown TCP.

- uint32 net_crc32le (const uint8 *data, int size)

  Calculate a "little-endian" CRC-32 over a block of data.
- uint32 net_crc32be (const uint8 *data, int size)

  Calculate a "big-endian" CRC-32 over a block of data.
- uint16 net_crc16ccitt (const uint8 *data, int size, uint16 start)

  Calculate a CRC16-CCITT over a block of data.
- int net_multicast_add (const uint8 mac[6])

  Add a entry to our multicast list.
- int net_multicast_del (const uint8 mac[6])

  Delete a entry from our multicast list.
- int net_multicast_check (const uint8 mac[6])

  Check if an address is on the multicast list.
- int net_multicast_init (void)

  Init multicast support.
- void net_multicast_shutdown (void)

  Shutdown multicast support.
- struct netif_list * net_get_if_list (void)

  Function to retrieve the interface list.
- netif_t * net_set_default (netif_t *n)

  Set our default device to an arbitrary device.
- int net_reg_device (netif_t *device)

  Register a network device.
- int net_unreg_device (netif_t *device)

  Unregister a network device.
- int net_init (uint32 ip)

  Init network support.
- void net_shutdown (void)

  Shutdown network support.

Variables

- net_input_func net_input_target

  Where will input packets be routed?
- net_echo_cb net_icmp_echo_cb

  Where will we handle possibly notifying the user of ping replies?
- net6_echo_cb net_icmp6_echo_cb

  Where will we handle possibly notifying the user of ping replies?
- struct netif_list net_if_list

  Interface list; note: do not manipulate directly!
- netif_t * net_default_dev

  The default network device, used with sockets (read-only).

9.68.1 Detailed Description

Network support.

This file contains declarations related to networking support. KOS' built-in network stack supports UDP over IPv4, and to some degree has some basic IPv6 support as well. This will change over time, hopefully leaving us with full TCP and UDP support both over IPv4 and IPv6.

Author

Lawrence Sebald

Megan Potter

9.68.2 Macro Definition Documentation

ICMP6_DEST_UNREACH_ADDR_UNREACH

```
#define ICMP6_DEST_UNREACH_ADDR_UNREACH 3
```

Address unreachable.

ICMP6_DEST_UNREACH_BAD_ROUTE

```
#define ICMP6_DEST_UNREACH_BAD_ROUTE 6
```

Bad route specified.

ICMP6_DEST_UNREACH_BEYOND_SCOPE

```
#define ICMP6_DEST_UNREACH_BEYOND_SCOPE 2
```

Gone beyond scope.

ICMP6_DEST_UNREACH_FAIL_EGRESS

```
#define ICMP6_DEST_UNREACH_FAIL_EGRESS 5
```

Egress failure.

ICMP6_DEST_UNREACH_NO_ROUTE

```
#define ICMP6_DEST_UNREACH_NO_ROUTE 0
```

No route available.

ICMP6_DEST_UNREACH_PORT_UNREACH

```
#define ICMP6_DEST_UNREACH_PORT_UNREACH 4
```

Port unreachable.

ICMP6_DEST_UNREACH_PROHIBITED

```
#define ICMP6_DEST_UNREACH_PROHIBITED 1
```

Access prohibited.

ICMP6_PARAM_PROB_BAD_HEADER

```
#define ICMP6_PARAM_PROB_BAD_HEADER 0
```

Malformed header.

ICMP6_PARAM_PROB_UNK_HEADER

```
#define ICMP6_PARAM_PROB_UNK_HEADER 1
```

Unknown header.

ICMP6_PARAM_PROB_UNK_OPTION

```
#define ICMP6_PARAM_PROB_UNK_OPTION 2
```

Unknown header option.

ICMP6_TIME_EXCEEDED_FRAGMENT

```
#define ICMP6_TIME_EXCEEDED_FRAGMENT 1
```

Reassembly time gone.

ICMP6_TIME_EXCEEDED_HOPS_EXC

```
#define ICMP6_TIME_EXCEEDED_HOPS_EXC 0
```

Hops exceeded.

ICMP_PORT_UNREACHABLE

```
#define ICMP_PORT_UNREACHABLE 3
```

Port unreachable.

ICMP_PROTOCOL_UNREACHABLE

```
#define ICMP_PROTOCOL_UNREACHABLE 2
```

Protocol unreachable.

ICMP_REASSEMBLY_TIME_EXCEEDED

```
#define ICMP_REASSEMBLY_TIME_EXCEEDED 1
```

Reassembly time gone.

NETIF_BLOCK

```
#define NETIF_BLOCK 1
```

Blocking is OK for Tx.

NETIF_NOBLOCK

```
#define NETIF_NOBLOCK 0
```

Don't block for Tx.

NETIF_TX_AGAIN

```
#define NETIF_TX_AGAIN -2
```

Retry Tx later.

NETIF_TX_ERROR

```
#define NETIF_TX_ERROR -1
```

Tx general error.

NETIF_TX_OK

```
#define NETIF_TX_OK 0
```

Tx success.

9.68.3 Typedef Documentation

net6_echo_cb

```
typedef void(* net6_echo_cb) (const struct in6_addr *ip, uint16 seq, uint64 delta_us, uint8 hlim,
const uint8 *data, size_t len)
```

ICMPv6 echo reply callback type.

Parameters

ip	The IPv6 address the reply is from.
seq	The sequence number of the packet.
delta_us	The time difference, in microseconds.
hlim	The hop limit value in the packet.
data	Any data in the packet.
len	The length of the data, in bytes.

net_echo_cb

```
typedef void(* net_echo_cb) (const uint8 *ip, uint16 seq, uint64 delta_us, uint8 ttl, const uint8
*data, size_t len)
```

ICMPv4 echo reply callback type.

Parameters

ip	The IPv4 address the reply is from.
seq	The sequence number of the packet.
delta_us	The time difference, in microseconds.
ttl	The TTL value in the packet.
data	Any data in the packet.
len	The length of the data, in bytes.

net_input_func

```
typedef int(* net_input_func) (netif_t *nif, const uint8 *pkt, int len)
```

Network input callback type.

Parameters

nif	The network device in use.
pkt	The packet received.
len	The length of the packet, in bytes.

Returns

0 on success, <0 on failure.

9.68.4 Function Documentation

net_arp_init()

```
int net_arp_init (
            void  )
```

Init ARP.

Return values

0	On success (no error conditions defined).

net_arp_input()

```
int net_arp_input (
            netif_t * nif,
            const uint8 * pkt,
            int len )
```

Receive an ARP packet and process it (called by net_input).

Parameters

nif	The network device in use.
pkt	The packet received.
len	The length of the packet.

Return values

0	On success (no error conditions defined).

net_arp_insert()

```
int net_arp_insert (
            netif_t * nif,
            const uint8 mac[6],
            const uint8 ip[4],
            uint64 timestamp )
```

Add an entry to the ARP cache manually.

Parameters

nif	The network device in use.
mac	The MAC address of the entry.
ip	The IPv4 address of the entry.
timestamp	The entry's timestamp. Set to 0 for a permanent entry, otherwise set to the current number of milliseconds since boot (i.e, timer_ms_gettime64()).

Return values

0	On success.
-1	Error allocating memory.

net_arp_lookup()

```
int net_arp_lookup (
            netif_t * nif,
            const uint8 ip_in[4],
            uint8 mac_out[6],
            const ip_hdr_t * pkt,
            const uint8 * data,
            int data_size )
```

Look up an entry from the ARP cache.

If no entry is found, then an ARP query will be sent and an error will be returned. If you specify a packet with the call, it will be sent when the reply comes in.

Parameters

nif	The network device in use.
ip_in	The IP address to lookup.
mac_out	Storage for the MAC address, if found.
pkt	A simple IPv4 header, if you want to send one when a response comes in (if not found immediately).
data	Packet data to go with the header.
data_size	The size of data.

Return values

0	On success.
-1	A query is outstanding for that address.
-2	Address not found, query generated.
-3	Error allocating memory.

net_arp_query()

```
int net_arp_query (
            netif_t * nif,
            const uint8 ip[4] )
```

Generate an ARP who-has query on the given device.

Parameters

nif	The network device to use.
ip	The IP to query.

Return values

0	On success (no error conditions defined).

net_arp_revlookup()

```
int net_arp_revlookup (
            netif_t * nif,
            uint8 ip_out[4],
            const uint8 mac_in[6] )
```

Do a reverse ARP lookup.

This function looks for an IP for a given mac address; note that if this fails, you have no recourse.

Parameters

nif	The network device in use.
ip_out	Storage for the IPv4 address.
mac_in	The MAC address to look up.

Return values

0	On success.
-1	On failure.

net_arp_shutdown()

```
void net_arp_shutdown (
            void  )
```

Shutdown ARP.

net_crc16ccitt()

```
uint16 net_crc16ccitt (
            const uint8 * data,
            int size,
            uint16 start )
```

Calculate a CRC16-CCITT over a block of data.

Parameters

data	The data to calculate over.
size	The size of the data, in bytes.
start	The value to start with. This could be a previous return value from this function (if continuing a previous calculation) or some initial seed value (typically 0xFFFF or 0x0000).

Returns

The calculated CRC16-CCITT.

Note

Based on code found online at http://www.ccsinfo.com/forum/viewtopic.php?t=24977

net_crc32be()

```
uint32 net_crc32be (
            const uint8 * data,
            int size )
```

Calculate a "big-endian" CRC-32 over a block of data.

Parameters

data	The data to calculate over.
size	The size of the data, in bytes.

Returns

The calculated CRC-32.

net_crc32le()

```
uint32 net_crc32le (
            const uint8 * data,
            int size )
```

Calculate a "little-endian" CRC-32 over a block of data.

Parameters

data	The data to calculate over.
size	The size of the data, in bytes.

Returns

The calculated CRC-32.

net_get_if_list()

```
struct netif_list * net_get_if_list (
            void  )
```

Function to retrieve the interface list.

Do not manipulate what this returns to you!

Returns

The network interface list.

net_icmp6_send_dest_unreach()

```
int net_icmp6_send_dest_unreach (
            netif_t * net,
            uint8 code,
            const uint8 * ppkt,
            size_t psz )
```

Send a destination unreachable packet on the specified interface.

Parameters

net	The network device to use.
code	The type of message this is.
ppkt	The message that caused this error.
psz	Size of the original message.

Returns

0 on success, $<$0 on failure.

net_icmp6_send_echo()

```
int net_icmp6_send_echo (
             netif_t * net,
             const struct in6_addr * dst,
             uint16 ident,
             uint16 seq,
             const uint8 * data,
             size_t size )
```

Send an ICMPv6 Echo (PING6) packet to the specified device.

Parameters

net	The network device to use.
dst	The address to send to.
ident	A packet identifier.
seq	A packet sequence number.
data	Data to send with the packet.
size	Length of the data, in bytes.

Returns

0 on success, <0 on failure.

net_icmp6_send_nadv()

```
int net_icmp6_send_nadv (
             netif_t * net,
             const struct in6_addr * dst,
             const struct in6_addr * target,
             int sol )
```

Send a Neighbor Advertisement packet on the specified device.

Parameters

net	The network device to use.
dst	The destination address.
target	The target address.
sol	1 if solicited, 0 otherwise.

Returns

0 on success, <0 on failure.

net_icmp6_send_nsol()

```
int net_icmp6_send_nsol (
            netif_t * net,
            const struct in6_addr * dst,
            const struct in6_addr * target,
            int dupdet )
```

Send a Neighbor Solicitation packet on the specified device.

Parameters

net	The network device to use.
dst	The destination address.
target	The target address.
dupdet	1 if this is for duplicate detection.

Returns

0 on success, <0 on failure.

net_icmp6_send_param_prob()

```
int net_icmp6_send_param_prob (
            netif_t * net,
            uint8 code,
            uint32 ptr,
            const uint8 * ppkt,
            size_t psz )
```

Send an ICMPv6 Parameter Problem about the given packet.

Parameters

net	The network device to use.
code	The error code.
ptr	Where in the packet is the error?
ppkt	The message that caused the error.
psz	Size of the original packet.

Returns

0 on success, <0 on failure.

net_icmp6_send_rsol()

```
int net_icmp6_send_rsol (
```

```
            netif_t * net )
```

Send a Router Solicitation request on the specified interface.

Parameters

net	The network device to use.

Returns

0 on success, <0 on failure.

net_icmp6_send_time_exceeded()

```
int net_icmp6_send_time_exceeded (
            netif_t * net,
            uint8 code,
            const uint8 * ppkt,
            size_t psz )
```

Send a time exceeded message on the specified interface.

Parameters

net	The network device to use.
code	The error code.
ppkt	The message that caused this error.
psz	Size of the original packet.

Returns

0 on success, <0 on failure.

net_icmp_send_dest_unreach()

```
int net_icmp_send_dest_unreach (
            netif_t * net,
            uint8 code,
            const uint8 * msg )
```

Send an ICMP Destination Unreachable packet in reply to the given message.

Parameters

net	The network device to use.
code	The type of message this is.
msg	The message that caused this error.

Returns

0 on success, <0 on failure.

net_icmp_send_echo()

```
int net_icmp_send_echo (
            netif_t * net,
            const uint8 ipaddr[4],
            uint16 ident,
            uint16 seq,
            const uint8 * data,
            size_t size )
```

Send an ICMP Echo packet to the specified IP.

Parameters

net	The network device to use.
ipaddr	The IPv4 address to send to.
ident	A packet identifier.
seq	A packet sequence number.
data	Data to send with the packet.
size	The size of the data to send.

Returns

0 on success, <0 on failure.

net_icmp_send_time_exceeded()

```
int net_icmp_send_time_exceeded (
            netif_t * net,
            uint8 code,
            const uint8 * msg )
```

Send an ICMP Time Exceeded packet in reply to the given message.

Parameters

net	The network device to use.
code	The type of message this is.
msg	The message that caused this error.

Returns

0 on success, <0 on failure.

net_init()

```
int net_init (
            uint32 ip )
```

Init network support.

Parameters

ip	The IPv4 address to set on the default device, in host byte order.

Returns

0 on success, <0 on failure.

Note

To auto-detect the IP address to assign to the default device (i.e, over DHCP or from the flashrom on the Dreamcast), pass 0 as the IP parameter.

net_input()

```
int net_input (
            netif_t * device,
            const uint8 * data,
            int len )
```

Device drivers should call this function to submit packets received in the background.

This function may or may not return immidiately but it won't take an infinitely long time (so it's safe to call inside interrupt handlers).

Parameters

device	The network device submitting packets.
data	The packet to submit.
len	The length of the packet, in bytes.

Returns

0 on success, <0 on failure.

net_input_set_target()

```
net_input_func net_input_set_target (
            net_input_func t )
```

Setup a network input target.

Parameters

t	The new target callback.

Returns

The old target.

net_ipv4_address()

```
uint32 net_ipv4_address (
            const uint8 addr[4] )
```

Create a 32-bit IP address, based on the individual numbers contained within the IP.

Parameters

addr	Array of IP address octets.

Returns

The address, in host byte order.

net_ipv4_get_stats()

```
net_ipv4_stats_t net_ipv4_get_stats (
            void  )
```

Retrieve statistics from the IPv4 layer.

Returns

The net_ipv4_stats_t structure.

net_ipv4_parse_address()

```
void net_ipv4_parse_address (
            uint32 addr,
            uint8 out[4] )
```

Parse an IP address that is packet into a uint32 into an array of the individual bytes.

Parameters

addr	The full address, in host byte order.
out	The output buffer.

net_ipv6_get_stats()

```
net_ipv6_stats_t net_ipv6_get_stats (
            void  )
```

Retrieve statistics from the IPv6 layer.

Returns

The global IPv6 stats structure.

net_multicast_add()

```
int net_multicast_add (
            const uint8 mac[6] )
```

Add a entry to our multicast list.

This function will auto-commit the multicast list to the network interface in the process.

Parameters

mac	The MAC address to add.

Returns

0 on success, <0 on failure.

net_multicast_check()

```
int net_multicast_check (
            const uint8 mac[6] )
```

Check if an address is on the multicast list.

Parameters

mac	The MAC address to check for.

Return values

0	The address is not in the list.
1	The address is in the list.
-1	On error.

net_multicast_del()

```
int net_multicast_del (
            const uint8 mac[6] )
```

Delete a entry from our multicast list.

This function will auto-commit the multicast list to the network interface in the process.

Parameters

mac	The MAC address to add.

Returns

0 on success, <0 on failure.

net_multicast_init()

```
int net_multicast_init (
            void  )
```

Init multicast support.

Returns

0 on success, !0 on error.

net_multicast_shutdown()

```
void net_multicast_shutdown (
            void  )
```

Shutdown multicast support.

net_ndp_gc()

```
void net_ndp_gc (
            void  )
```

Garbage collect timed out NDP entries. This will be called periodically as NDP queries come in.

net_ndp_init()

```
int net_ndp_init (
            void  )
```

Init NDP.

Return values

0	On success (no error conditions defined).

net_ndp_insert()

```
int net_ndp_insert (
            netif_t * nif,
            const uint8 mac[6],
            const struct in6_addr * ip,
            int unsol )
```

Add an entry to the NDP cache.

Parameters

nif	The network device in question.
mac	The MAC address for the entry.
ip	The IPv6 address for the entry.
unsol	Was this unsolicited?

Returns

0 on success, $<$0 on failure.

net_ndp_lookup()

```
int net_ndp_lookup (
            netif_t * net,
            const struct in6_addr * ip,
```

```
            uint8 mac_out[6],
            const ipv6_hdr_t * pkt,
            const uint8 * data,
            int data_size )
```

Look up an entry from the NDP cache.

If no entry is found, then an NDP query will be sent and an error will be returned. If you specify a packet with the call, it will be sent when the reply comes in.

Parameters

net	The network device to use.
ip	The IPv6 address to query.
mac_out	Storage for the MAC address on success.
pkt	A simple IPv6 header, if you want to send a packet when a reply comes in.
data	Anything that comes after the header.
data_size	The size of data.

Returns

0 on success, <0 on failure.

net_ndp_shutdown()

```
void net_ndp_shutdown (
            void  )
```

Shutdown NDP.

net_reg_device()

```
int net_reg_device (
            netif_t * device )
```

Register a network device.

Parameters

device	The device to register.

Returns

0 on success, <0 on failure.

net_set_default()

```
netif_t * net_set_default (
              netif_t * n )
```

Set our default device to an arbitrary device.

Parameters

n	The device to set as default.

Returns

The old default device.

net_shutdown()

```
void net_shutdown (
              void  )
```

Shutdown network support.

net_tcp_init()

```
int net_tcp_init (
              void  )
```

Init TCP.

Return values

0	On success (no error conditions defined).

net_tcp_shutdown()

```
void net_tcp_shutdown (
              void  )
```

Shutdown TCP.

net_udp_get_stats()

```
net_udp_stats_t net_udp_get_stats (
              void  )
```

Retrieve statistics from the UDP layer.

Returns

The global UDP stats struct.

net_udp_init()

```
int net_udp_init (
             void  )
```

Init UDP.

Return values

0	On success (no error conditions defined).

net_udp_shutdown()

```
void net_udp_shutdown (
             void  )
```

Shutdown UDP.

net_unreg_device()

```
int net_unreg_device (
             netif_t * device )
```

Unregister a network device.

Parameters

device	The device to unregister.

Returns

0 on success, <0 on failure.

9.68.5 Variable Documentation

net_default_dev

```
netif_t* net_default_dev  [extern]
```

The default network device, used with sockets (read-only).

net_icmp6_echo_cb

net6_echo_cb net_icmp6_echo_cb [extern]

Where will we handle possibly notifying the user of ping replies?

net_icmp_echo_cb

net_echo_cb net_icmp_echo_cb [extern]

Where will we handle possibly notifying the user of ping replies?

net_if_list

struct netif_list net_if_list [extern]

Interface list; note: do not manipulate directly!

net_input_target

net_input_func net_input_target [extern]

Where will input packets be routed?

9.69 net.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    include/kos/net.h
00004    Copyright (C) 2002 Megan Potter
00005    Copyright (C) 2005, 2006, 2007, 2008, 2009, 2010, 2012, 2013,
00006                  2016 Lawrence Sebald
00007
00008 */
00009
00010 /** \file   kos/net.h
00011     \brief  Network support.
00012
00013     This file contains declarations related to networking support. KOS' built-in
00014     network stack supports UDP over IPv4, and to some degree has some basic IPv6
00015     support as well. This will change over time, hopefully leaving us with full
00016     TCP and UDP support both over IPv4 and IPv6.
00017
00018     \author Lawrence Sebald
00019     \author Megan Potter
00020 */
00021
00022 #ifndef __KOS_NET_H
00023 #define __KOS_NET_H
00024
00025 #include <sys/cdefs.h>
00026 __BEGIN_DECLS
00027
00028 #include <arch/types.h>
00029 #include <sys/queue.h>
00030 #include <netinet/in.h>
```

```
00031
00032 /* All functions in this header return < 0 on failure, and 0 on success. */
00033
00034
00035
00036 /** \brief Structure describing one usable network device.
00037
00038     Each usable network device should have one of these describing it. These
00039     must be registered to the network layer before the device is useable.
00040
00041     \headerfile kos/net.h
00042 */
00043 typedef struct knetif {
00044     /** \brief  Device list handle (not a function!) */
00045     LIST_ENTRY(knetif)  if_list;
00046
00047     /** \brief  Device name ("bba", "la", etc) */
00048     const char          *name;
00049
00050     /** \brief  Long description of the device */
00051     const char          *descr;
00052
00053     /** \brief  Unit index (starts at zero and counts upwards for multiple
00054                         network devices of the same type) */
00055     int                 index;
00056
00057     /** \brief  Internal device ID (for whatever the driver wants) */
00058     uint32              dev_id;
00059
00060     /** \brief  Interface flags */
00061     uint32              flags;
00062
00063     /** \brief  The device's MAC address */
00064     uint8               mac_addr[6];
00065
00066     /** \brief  The device's IP address (if any) */
00067     uint8               ip_addr[4];
00068
00069     /** \brief  The device's netmask */
00070     uint8               netmask[4];
00071
00072     /** \brief  The device's gateway's IP address */
00073     uint8               gateway[4];
00074
00075     /** \brief  The device's broadcast address */
00076     uint8               broadcast[4];
00077
00078     /** \brief  The device's DNS server address */
00079     uint8               dns[4];
00080
00081     /** \brief  The device's MTU */
00082     int                 mtu;
00083
00084     /** \brief  The device's Link-local IPv6 address */
00085     struct in6_addr     ip6_lladdr;
00086
00087     /** \brief  Any further IPv6 addresses the device has.
00088         The first address in this list will always be used, unless otherwise
00089         specified. */
00090     struct in6_addr     *ip6_addrs;
00091     int                 ip6_addr_count;
00092
00093     /** \brief  The device's gateway's IPv6 address */
00094     struct in6_addr     ip6_gateway;
00095
00096     /** \brief  Default MTU over IPv6 */
00097     uint32              mtu6;
00098
00099     /** \brief  Default hop limit over IPv6 */
00100     int                 hop_limit;
00101
00102     /* All of the following callback functions should return a negative
00103        value on failure, and a zero or positive value on success. Some
00104        functions have special values, as noted. */
00105
00106     /** \brief  Attempt to detect the device.
00107         \param  self        The network device in question.
00108         \return             0 on success, <0 on failure.
00109     */
00110     int (*if_detect)(struct knetif * self);
00111
```

```
00112     /** \brief  Initialize the device.
00113         \param  self        The network device in question.
00114         \return             0 on success, <0 on failure.
00115     */
00116     int (*if_init)(struct knetif * self);
00117
00118     /** \brief  Shutdown the device.
00119         \param  self        The network device in question.
00120         \return             0 on success, <0 on failure.
00121     */
00122     int (*if_shutdown)(struct knetif * self);
00123
00124     /** \brief  Start the device (after init or stop).
00125         \param  self        The network device in question.
00126         \return             0 on success, <0 on failure.
00127     */
00128     int (*if_start)(struct knetif * self);
00129
00130     /** \brief  Stop (hibernate) the device.
00131         \param  self        The network device in question.
00132         \return             0 on success, <0 on failure
00133     */
00134     int (*if_stop)(struct knetif * self);
00135
00136     /** \brief  Queue a packet for transmission.
00137         \param  self        The network device in question.
00138         \param  data        The packet to transmit.
00139         \param  len         The length of the packet in bytes.
00140         \param  blocking    1 if we should block if needed, 0 otherwise.
00141         \retval NETIF_TX_OK     On success.
00142         \retval NETIF_TX_ERROR  On general failure.
00143         \retval NETIF_TX_AGAIN  If non-blocking and we must block to send.
00144     */
00145     int (*if_tx)(struct knetif * self, const uint8 * data, int len,
00146                  int blocking);
00147
00148     /** \brief  Commit any queued output packets.
00149         \param  self        The network device in question.
00150         \return             0 on success, <0 on failure.
00151     */
00152     int (*if_tx_commit)(struct knetif * self);
00153
00154     /** \brief  Poll for queued receive packets, if neccessary.
00155         \param  self        The network device in question.
00156         \return             0 on success, <0 on failure.
00157     */
00158     int (*if_rx_poll)(struct knetif * self);
00159
00160     /** \brief  Set flags; you should generally manipulate flags through here so
00161                 that the driver gets a chance to act on the info.
00162         \param  self        The network device in question.
00163         \param  flags_and   Bitmask to and with the flags.
00164         \param  flags_or    Bitmask to or with the flags.
00165     */
00166     int (*if_set_flags)(struct knetif * self, uint32 flags_and, uint32 flags_or);
00167
00168     /** \brief  Set the device's multicast list.
00169         \param  self        The network device in question.
00170         \param  list        The list of MAC addresses (6 * count bytes).
00171         \param  count       The number of addresses in list.
00172     */
00173     int (*if_set_mc)(struct knetif *self, const uint8 *list, int count);
00174 } netif_t;
00175
00176 /** \defgroup net_flags Flags for netif_t
00177     @{
00178 */
00179 #define NETIF_NO_FLAGS      0x00000000      /**< \brief No flags set */
00180 #define NETIF_REGISTERED    0x00000001      /**< \brief Is it registered? */
00181 #define NETIF_DETECTED      0x00000002      /**< \brief Is it detected? */
00182 #define NETIF_INITIALIZED   0x00000004      /**< \brief Has it been initialized? */
00183 #define NETIF_RUNNING       0x00000008      /**< \brief Has start() been called? */
00184 #define NETIF_PROMISC       0x00010000      /**< \brief Promiscuous mode */
00185 #define NETIF_NEEDSPOLL     0x01000000      /**< \brief Needs to be polled for input */
00186 #define NETIF_NOETH         0x10000000      /**< \brief Does not use ethernet */
00187 /** @} */
00188
00189 /* Return types for if_tx */
00190 #define NETIF_TX_OK     0   /**< \brief Tx success */
00191 #define NETIF_TX_ERROR  -1  /**< \brief Tx general error */
00192 #define NETIF_TX_AGAIN  -2  /**< \brief Retry Tx later */
```

```
00193
00194 /* Blocking types */
00195 #define NETIF_NOBLOCK   0   /**< \brief Don't block for Tx */
00196 #define NETIF_BLOCK     1   /**< \brief Blocking is OK for Tx */
00197
00198 /* \cond */
00199 /* Define the list type */
00200 LIST_HEAD(netif_list, knetif);
00201
00202 #ifdef PACKED
00203 #undef PACKED
00204 #endif
00205
00206 #define PACKED __attribute__((packed))
00207 /* \endcond */
00208
00209 /** \brief IPv4 Packet header.
00210     \headerfile kos/net.h
00211 */
00212 typedef struct ip_hdr_s {
00213     uint8   version_ihl;        /**< \brief IP version and header length */
00214     uint8   tos;                /**< \brief Type of Service */
00215     uint16  length;             /**< \brief Length */
00216     uint16  packet_id;          /**< \brief Packet ID */
00217     uint16  flags_frag_offs;    /**< \brief Flags and fragment offset */
00218     uint8   ttl;                /**< \brief Time to live */
00219     uint8   protocol;           /**< \brief IP protocol */
00220     uint16  checksum;           /**< \brief IP checksum */
00221     uint32  src;                /**< \brief Source IP address */
00222     uint32  dest;               /**< \brief Destination IP address */
00223 } PACKED ip_hdr_t;
00224
00225 /** \brief  IPv6 Packet header.
00226     \headerfile kos/net.h
00227 */
00228 typedef struct ipv6_hdr_s {
00229     uint8           version_lclass; /**< \brief Version and low-order class
00230                                          byte */
00231     uint8           hclass_lflow;   /**< \brief High-order class byte, low-order
00232                                          flow byte */
00233     uint16          lclass;         /**< \brief Low-order class byte */
00234     uint16          length;         /**< \brief Length */
00235     uint8           next_header;    /**< \brief Next header type */
00236     uint8           hop_limit;      /**< \brief Hop limit */
00237     struct in6_addr src_addr;       /**< \brief Source IP address */
00238     struct in6_addr dst_addr;       /**< \brief Destination IP address */
00239 } PACKED ipv6_hdr_t;
00240
00241 #undef PACKED
00242
00243
00244 /***** net_arp.c ************************************************************/
00245
00246 /** \brief  Init ARP.
00247     \retval 0               On success (no error conditions defined).
00248 */
00249 int net_arp_init(void);
00250
00251 /** \brief  Shutdown ARP. */
00252 void net_arp_shutdown(void);
00253
00254 /** \brief  Add an entry to the ARP cache manually.
00255
00256     \param  nif             The network device in use.
00257     \param  mac             The MAC address of the entry.
00258     \param  ip              The IPv4 address of the entry.
00259     \param  timestamp       The entry's timestamp. Set to 0 for a permanent
00260                             entry, otherwise set to the current number of
00261                             milliseconds since boot (i.e, timer_ms_gettime64()).
00262     \retval 0               On success.
00263     \retval -1              Error allocating memory.
00264 */
00265 int net_arp_insert(netif_t *nif, const uint8 mac[6], const uint8 ip[4],
00266                    uint64 timestamp);
00267
00268 /** \brief  Look up an entry from the ARP cache.
00269
00270     If no entry is found, then an ARP query will be sent and an error will be
00271     returned. If you specify a packet with the call, it will be sent when the
00272     reply comes in.
00273
```

```
00274     \param  nif             The network device in use.
00275     \param  ip_in           The IP address to lookup.
00276     \param  mac_out         Storage for the MAC address, if found.
00277     \param  pkt             A simple IPv4 header, if you want to send one when a
00278                             response comes in (if not found immediately).
00279     \param  data            Packet data to go with the header.
00280     \param  data_size       The size of data.
00281     \retval 0               On success.
00282     \retval -1              A query is outstanding for that address.
00283     \retval -2              Address not found, query generated.
00284     \retval -3              Error allocating memory.
00285 */
00286 int net_arp_lookup(netif_t *nif, const uint8 ip_in[4], uint8 mac_out[6],
00287                    const ip_hdr_t *pkt, const uint8 *data, int data_size);
00288
00289 /** \brief  Do a reverse ARP lookup.
00290
00291     This function looks for an IP for a given mac address; note that if this
00292     fails, you have no recourse.
00293
00294     \param  nif             The network device in use.
00295     \param  ip_out          Storage for the IPv4 address.
00296     \param  mac_in          The MAC address to look up.
00297     \retval 0               On success.
00298     \retval -1              On failure.
00299 */
00300 int net_arp_revlookup(netif_t *nif, uint8 ip_out[4], const uint8 mac_in[6]);
00301
00302 /** \brief  Receive an ARP packet and process it (called by net_input).
00303     \param  nif             The network device in use.
00304     \param  pkt             The packet received.
00305     \param  len             The length of the packet.
00306     \retval 0               On success (no error conditions defined).
00307 */
00308 int net_arp_input(netif_t *nif, const uint8 *pkt, int len);
00309
00310 /** \brief  Generate an ARP who-has query on the given device.
00311     \param  nif             The network device to use.
00312     \param  ip              The IP to query.
00313     \retval 0               On success (no error conditions defined).
00314 */
00315 int net_arp_query(netif_t *nif, const uint8 ip[4]);
00316
00317
00318 /***** net_input.c ************************************************************/
00319
00320 /** \brief  Network input callback type.
00321     \param  nif             The network device in use.
00322     \param  pkt             The packet received.
00323     \param  len             The length of the packet, in bytes.
00324     \return                 0 on success, <0 on failure.
00325 */
00326 typedef int (*net_input_func)(netif_t *nif, const uint8 *pkt, int len);
00327
00328 /** \brief  Where will input packets be routed? */
00329 extern net_input_func net_input_target;
00330
00331 /** \brief  Device drivers should call this function to submit packets received
00332             in the background.
00333
00334     This function may or may not return immidiately but it won't take an
00335     infinitely long time (so it's safe to call inside interrupt handlers).
00336
00337     \param  device          The network device submitting packets.
00338     \param  data            The packet to submit.
00339     \param  len             The length of the packet, in bytes.
00340     \return                 0 on success, <0 on failure.
00341 */
00342 int net_input(netif_t *device, const uint8 *data, int len);
00343
00344 /** \brief  Setup a network input target.
00345     \param  t               The new target callback.
00346     \return                 The old target.
00347 */
00348 net_input_func net_input_set_target(net_input_func t);
00349
00350 /***** net_icmp.c *************************************************************/
00351
00352 /** \brief  ICMPv4 echo reply callback type.
00353     \param  ip              The IPv4 address the reply is from.
00354     \param  seq             The sequence number of the packet.
```

```
00355     \param  delta_us        The time difference, in microseconds.
00356     \param  ttl             The TTL value in the packet.
00357     \param  data            Any data in the packet.
00358     \param  len             The length of the data, in bytes.
00359 */
00360 typedef void (*net_echo_cb)(const uint8 *ip, uint16 seq, uint64 delta_us,
00361                             uint8 ttl, const uint8 *data, size_t len);
00362
00363 /** \brief  Where will we handle possibly notifying the user of ping replies? */
00364 extern net_echo_cb net_icmp_echo_cb;
00365
00366 /** \brief  Send an ICMP Echo packet to the specified IP.
00367     \param  net             The network device to use.
00368     \param  ipaddr          The IPv4 address to send to.
00369     \param  ident           A packet identifier.
00370     \param  seq             A packet sequence number.
00371     \param  data            Data to send with the packet.
00372     \param  size            The size of the data to send.
00373     \return                 0 on success, <0 on failure.
00374 */
00375 int net_icmp_send_echo(netif_t *net, const uint8 ipaddr[4], uint16 ident,
00376                        uint16 seq, const uint8 *data, size_t size);
00377
00378 /* Valid values for the code in the net_icmp_send_dest_unreach() function. */
00379 #define ICMP_PROTOCOL_UNREACHABLE       2   /**< \brief Protocol unreachable */
00380 #define ICMP_PORT_UNREACHABLE           3   /**< \brief Port unreachable */
00381
00382 /** \brief  Send an ICMP Destination Unreachable packet in reply to the given
00383             message.
00384     \param  net             The network device to use.
00385     \param  code            The type of message this is.
00386     \param  msg             The message that caused this error.
00387     \return                 0 on success, <0 on failure.
00388 */
00389 int net_icmp_send_dest_unreach(netif_t *net, uint8 code, const uint8 *msg);
00390
00391 /* Valid values for the code in the net_icmp_send_time_exceeded() function. */
00392 #define ICMP_REASSEMBLY_TIME_EXCEEDED   1   /**< \brief Reassembly time gone */
00393
00394 /** \brief  Send an ICMP Time Exceeded packet in reply to the given message.
00395     \param  net             The network device to use.
00396     \param  code            The type of message this is.
00397     \param  msg             The message that caused this error.
00398     \return                 0 on success, <0 on failure.
00399 */
00400 int net_icmp_send_time_exceeded(netif_t *net, uint8 code, const uint8 *msg);
00401
00402 /***** net_ipv4.c ************************************************************/
00403
00404 /** \brief IPv4 statistics structure.
00405
00406     This structure holds some basic statistics about the IPv4 layer of the
00407     stack, and can be retrieved with the appropriate function.
00408
00409     \headerfile kos/net.h
00410 */
00411 typedef struct net_ipv4_stats {
00412     uint32  pkt_sent;               /** \brief Packets sent out successfully */
00413     uint32  pkt_send_failed;        /** \brief Packets that failed to send */
00414     uint32  pkt_recv;               /** \brief Packets received successfully */
00415     uint32  pkt_recv_bad_size;      /** \brief Packets of a bad size */
00416     uint32  pkt_recv_bad_chksum;    /** \brief Packets with a bad checksum */
00417     uint32  pkt_recv_bad_proto;     /** \brief Packets with an unknown proto */
00418 } net_ipv4_stats_t;
00419
00420 /** \brief  Retrieve statistics from the IPv4 layer.
00421     \return                 The net_ipv4_stats_t structure.
00422 */
00423 net_ipv4_stats_t net_ipv4_get_stats(void);
00424
00425 /** \brief  Create a 32-bit IP address, based on the individual numbers
00426             contained within the IP.
00427     \param  addr            Array of IP address octets.
00428     \return                 The address, in host byte order.
00429 */
00430 uint32 net_ipv4_address(const uint8 addr[4]);
00431
00432 /** \brief  Parse an IP address that is packet into a uint32 into an array of
00433             the individual bytes.
00434     \param  addr            The full address, in host byte order.
00435     \param  out             The output buffer.
```

```
00436 */
00437 void net_ipv4_parse_address(uint32 addr, uint8 out[4]);
00438
00439 /***** net_icmp6.c ********************************************************/
00440
00441 /** \brief  ICMPv6 echo reply callback type.
00442     \param  ip              The IPv6 address the reply is from.
00443     \param  seq             The sequence number of the packet.
00444     \param  delta_us        The time difference, in microseconds.
00445     \param  hlim            The hop limit value in the packet.
00446     \param  data            Any data in the packet.
00447     \param  len             The length of the data, in bytes.
00448 */
00449 typedef void (*net6_echo_cb)(const struct in6_addr *ip, uint16 seq,
00450                              uint64 delta_us, uint8 hlim, const uint8 *data,
00451                              size_t len);
00452
00453 /** \brief  Where will we handle possibly notifying the user of ping replies? */
00454 extern net6_echo_cb net_icmp6_echo_cb;
00455
00456 /** \brief  Send an ICMPv6 Echo (PING6) packet to the specified device.
00457     \param  net             The network device to use.
00458     \param  dst             The address to send to.
00459     \param  ident           A packet identifier.
00460     \param  seq             A packet sequence number.
00461     \param  data            Data to send with the packet.
00462     \param  size            Length of the data, in bytes.
00463     \return                 0 on success, <0 on failure.
00464 */
00465 int net_icmp6_send_echo(netif_t *net, const struct in6_addr *dst, uint16 ident,
00466                         uint16 seq, const uint8 *data, size_t size);
00467
00468 /** \brief  Send a Neighbor Solicitation packet on the specified device.
00469     \param  net             The network device to use.
00470     \param  dst             The destination address.
00471     \param  target          The target address.
00472     \param  dupdet          1 if this is for duplicate detection.
00473     \return                 0 on success, <0 on failure.
00474 */
00475 int net_icmp6_send_nsol(netif_t *net, const struct in6_addr *dst,
00476                         const struct in6_addr *target, int dupdet);
00477
00478 /** \brief  Send a Neighbor Advertisement packet on the specified device.
00479     \param  net             The network device to use.
00480     \param  dst             The destination address.
00481     \param  target          The target address.
00482     \param  sol             1 if solicited, 0 otherwise.
00483     \return                 0 on success, <0 on failure.
00484 */
00485 int net_icmp6_send_nadv(netif_t *net, const struct in6_addr *dst,
00486                         const struct in6_addr *target, int sol);
00487
00488 /** \brief  Send a Router Solicitation request on the specified interface.
00489     \param  net             The network device to use.
00490     \return                 0 on success, <0 on failure.
00491 */
00492 int net_icmp6_send_rsol(netif_t *net);
00493
00494 /* Destination Unreachable codes -- only port unreachable really makes sense */
00495 #define ICMP6_DEST_UNREACH_NO_ROUTE     0   /**< \brief No route available */
00496 #define ICMP6_DEST_UNREACH_PROHIBITED   1   /**< \brief Access prohibited */
00497 #define ICMP6_DEST_UNREACH_BEYOND_SCOPE 2   /**< \brief Gone beyond scope */
00498 #define ICMP6_DEST_UNREACH_ADDR_UNREACH 3   /**< \brief Address unreachable */
00499 #define ICMP6_DEST_UNREACH_PORT_UNREACH 4   /**< \brief Port unreachable */
00500 #define ICMP6_DEST_UNREACH_FAIL_EGRESS  5   /**< \brief Egress failure */
00501 #define ICMP6_DEST_UNREACH_BAD_ROUTE    6   /**< \brief Bad route specified */
00502
00503 /** \brief  Send a destination unreachable packet on the specified interface.
00504     \param  net             The network device to use.
00505     \param  code            The type of message this is.
00506     \param  ppkt            The message that caused this error.
00507     \param  psz             Size of the original message.
00508     \return                 0 on success, <0 on failure.
00509 */
00510 int net_icmp6_send_dest_unreach(netif_t *net, uint8 code, const uint8 *ppkt,
00511                                 size_t psz);
00512
00513 /* Time Exceeded codes -- only fragment reassembly time exceeded makes sense */
00514 #define ICMP6_TIME_EXCEEDED_HOPS_EXC    0   /**< \brief Hops exceeded */
00515 #define ICMP6_TIME_EXCEEDED_FRAGMENT    1   /**< \brief Reassembly time gone */
00516
```

```
00517 /** \brief  Send a time exceeded message on the specified interface.
00518     \param  net             The network device to use.
00519     \param  code            The error code.
00520     \param  ppkt            The message that caused this error.
00521     \param  psz             Size of the original packet.
00522     \return                 0 on success, <0 on failure.
00523 */
00524 int net_icmp6_send_time_exceeded(netif_t *net, uint8 code, const uint8 *ppkt,
00525                                  size_t psz);
00526
00527 /* Parameter Problem codes */
00528 #define ICMP6_PARAM_PROB_BAD_HEADER     0   /**< \brief Malformed header */
00529 #define ICMP6_PARAM_PROB_UNK_HEADER     1   /**< \brief Unknown header */
00530 #define ICMP6_PARAM_PROB_UNK_OPTION     2   /**< \brief Unknown header option */
00531
00532 /** \brief  Send an ICMPv6 Parameter Problem about the given packet.
00533     \param  net             The network device to use.
00534     \param  code            The error code.
00535     \param  ptr             Where in the packet is the error?
00536     \param  ppkt            The message that caused the error.
00537     \param  psz             Size of the original packet.
00538     \return                 0 on success, <0 on failure.
00539 */
00540 int net_icmp6_send_param_prob(netif_t *net, uint8 code, uint32 ptr,
00541                               const uint8 *ppkt, size_t psz);
00542
00543 /***** net_ipv6.c ************************************************************/
00544
00545 /** \brief  IPv6 statistics structure.
00546
00547     This structure holds some basic statistics about the IPv6 layer of the
00548     stack, and can be retrieved with the appropriate function.
00549
00550     \headerfile kos/net.h
00551 */
00552 typedef struct net_ipv6_stats {
00553     uint32  pkt_sent;               /**< \brief Packets sent out successfully */
00554     uint32  pkt_send_failed;        /**< \brief Packets that failed to send */
00555     uint32  pkt_recv;               /**< \brief Packets received successfully */
00556     uint32  pkt_recv_bad_size;      /**< \brief Packets of a bad size */
00557     uint32  pkt_recv_bad_proto;     /**< \brief Packets with an unknown proto */
00558     uint32  pkt_recv_bad_ext;       /**< \brief Packets with an unknown hdr */
00559 } net_ipv6_stats_t;
00560
00561 /** \brief  Retrieve statistics from the IPv6 layer.
00562     \return                 The global IPv6 stats structure.
00563 */
00564 net_ipv6_stats_t net_ipv6_get_stats(void);
00565
00566 /***** net_ndp.c *************************************************************/
00567
00568 /** \brief  Init NDP.
00569     \retval 0               On success (no error conditions defined).
00570 */
00571 int net_ndp_init(void);
00572
00573 /** \brief  Shutdown NDP. */
00574 void net_ndp_shutdown(void);
00575
00576 /** \brief  Garbage collect timed out NDP entries.
00577     This will be called periodically as NDP queries come in.
00578 */
00579 void net_ndp_gc(void);
00580
00581 /** \brief  Add an entry to the NDP cache.
00582     \param  nif             The network device in question.
00583     \param  mac             The MAC address for the entry.
00584     \param  ip              The IPv6 address for the entry.
00585     \param  unsol           Was this unsolicited?
00586     \return                 0 on success, <0 on failure.
00587 */
00588 int net_ndp_insert(netif_t *nif, const uint8 mac[6], const struct in6_addr *ip,
00589                    int unsol);
00590
00591 /** \brief  Look up an entry from the NDP cache.
00592
00593     If no entry is found, then an NDP query will be sent and an error will be
00594     returned. If you specify a packet with the call, it will be sent when the
00595     reply comes in.
00596
00597     \param  net             The network device to use.
```

```
00598     \param  ip              The IPv6 address to query.
00599     \param  mac_out         Storage for the MAC address on success.
00600     \param  pkt             A simple IPv6 header, if you want to send a packet
00601                             when a reply comes in.
00602     \param  data            Anything that comes after the header.
00603     \param  data_size       The size of data.
00604     \return                 0 on success, <0 on failure.
00605 */
00606 int net_ndp_lookup(netif_t *net, const struct in6_addr *ip, uint8 mac_out[6],
00607                    const ipv6_hdr_t *pkt, const uint8 *data, int data_size);
00608
00609 /***** net_udp.c ************************************************************/
00610
00611 /** \brief  UDP statistics structure.
00612
00613     This structure holds some basic statistics about the UDP layer of the stack,
00614     and can be retrieved with the appropriate function.
00615
00616     \headerfile kos/net.h
00617 */
00618 typedef struct net_udp_stats {
00619     uint32  pkt_sent;               /**< \brief Packets sent out successfully */
00620     uint32  pkt_send_failed;        /**< \brief Packets that failed to send */
00621     uint32  pkt_recv;               /**< \brief Packets received successfully */
00622     uint32  pkt_recv_bad_size;      /**< \brief Packets of a bad size */
00623     uint32  pkt_recv_bad_chksum;    /**< \brief Packets with a bad checksum */
00624     uint32  pkt_recv_no_sock;       /**< \brief Packets with to a closed port */
00625 } net_udp_stats_t;
00626
00627 /** \brief  Retrieve statistics from the UDP layer.
00628     \return                 The global UDP stats struct.
00629 */
00630 net_udp_stats_t net_udp_get_stats(void);
00631
00632 /** \brief  Init UDP.
00633     \retval 0               On success (no error conditions defined).
00634 */
00635 int net_udp_init(void);
00636
00637 /** \brief  Shutdown UDP. */
00638 void net_udp_shutdown(void);
00639
00640 /***** net_tcp.c ************************************************************/
00641
00642 /** \brief  Init TCP.
00643     \retval 0               On success (no error conditions defined).
00644 */
00645 int net_tcp_init(void);
00646
00647 /** \brief  Shutdown TCP. */
00648 void net_tcp_shutdown(void);
00649
00650 /***** net_crc.c ************************************************************/
00651
00652 /** \brief  Calculate a "little-endian" CRC-32 over a block of data.
00653     \param  data            The data to calculate over.
00654     \param  size            The size of the data, in bytes.
00655     \return                 The calculated CRC-32.
00656 */
00657 uint32 net_crc32le(const uint8 *data, int size);
00658
00659 /** \brief  Calculate a "big-endian" CRC-32 over a block of data.
00660     \param  data            The data to calculate over.
00661     \param  size            The size of the data, in bytes.
00662     \return                 The calculated CRC-32.
00663 */
00664 uint32 net_crc32be(const uint8 *data, int size);
00665
00666 /** \brief  Calculate a CRC16-CCITT over a block of data.
00667     \param  data            The data to calculate over.
00668     \param  size            The size of the data, in bytes.
00669     \param  start           The value to start with. This could be a previous
00670                             return value from this function (if continuing a
00671                             previous calculation) or some initial seed value
00672                             (typically 0xFFFF or 0x0000).
00673     \return                 The calculated CRC16-CCITT.
00674     \note                   Based on code found online at
00675                             http://www.ccsinfo.com/forum/viewtopic.php?t=24977
00676 */
00677 uint16 net_crc16ccitt(const uint8 *data, int size, uint16 start);
00678
```

```
00679 /***** net_multicast.c ***************************************************/
00680
00681 /** \brief  Add a entry to our multicast list.
00682
00683     This function will auto-commit the multicast list to the network interface
00684     in the process.
00685
00686     \param  mac             The MAC address to add.
00687     \return                 0 on success, <0 on failure.
00688 */
00689 int net_multicast_add(const uint8 mac[6]);
00690
00691 /** \brief  Delete a entry from our multicast list.
00692
00693     This function will auto-commit the multicast list to the network interface
00694     in the process.
00695
00696     \param  mac             The MAC address to add.
00697     \return                 0 on success, <0 on failure.
00698 */
00699 int net_multicast_del(const uint8 mac[6]);
00700
00701 /** \brief  Check if an address is on the multicast list.
00702     \param  mac             The MAC address to check for.
00703     \retval 0               The address is not in the list.
00704     \retval 1               The address is in the list.
00705     \retval -1              On error.
00706 */
00707 int net_multicast_check(const uint8 mac[6]);
00708
00709 /** \brief  Init multicast support.
00710     \return                 0 on success, !0 on error.
00711 */
00712 int net_multicast_init(void);
00713
00714 /** \brief  Shutdown multicast support. */
00715 void net_multicast_shutdown(void);
00716
00717 /***** net_core.c ***************************************************/
00718
00719 /** \brief  Interface list; note: do not manipulate directly! */
00720 extern struct netif_list net_if_list;
00721
00722 /** \brief  Function to retrieve the interface list.
00723
00724     Do not manipulate what this returns to you!
00725
00726     \return                 The network interface list.
00727 */
00728 struct netif_list * net_get_if_list(void);
00729
00730 /** \brief  The default network device, used with sockets (read-only). */
00731 extern netif_t *net_default_dev;
00732
00733 /** \brief  Set our default device to an arbitrary device.
00734     \param  n               The device to set as default.
00735     \return                 The old default device.
00736 */
00737 netif_t *net_set_default(netif_t *n);
00738
00739 /** \brief  Register a network device.
00740     \param  device          The device to register.
00741     \return                 0 on success, <0 on failure.
00742 */
00743 int net_reg_device(netif_t *device);
00744
00745 /** \brief  Unregister a network device.
00746     \param  device          The device to unregister.
00747     \return                 0 on success, <0 on failure.
00748 */
00749 int net_unreg_device(netif_t *device);
00750
00751 /** \brief  Init network support.
00752     \param  ip              The IPv4 address to set on the default device, in
00753                             host byte order.
00754
00755     \return                 0 on success, <0 on failure.
00756     \note                   To auto-detect the IP address to assign to the
00757                             default device (i.e, over DHCP or from the flashrom
00758                             on the Dreamcast), pass 0 as the IP parameter.
00759 */
```

```
00760 int net_init(uint32 ip);
00761
00762 /** \brief  Shutdown network support. */
00763 void net_shutdown(void);
00764
00765 __END_DECLS
00766
00767 #endif  /* __KOS_NET_H */
```

9.70 include/kos/nmmgr.h File Reference

Name manager.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <kos/limits.h>
#include <sys/queue.h>
```

Data Structures

- struct nmmgr_handler_t

  Name handler interface.

Macros

- #define NMMGR_LIST_INIT { NULL }

  List entry initializer for static structs.
- #define NMMGR_FLAGS_NEEDSFREE 0x00000001

  This structure must be freed when removed.
- #define NMMGR_TYPE_UNKNOWN 0x0000 /* ? */

  Unknown nmmgr type.
- #define NMMGR_TYPE_VFS 0x0010 /* Mounted file system */

  A mounted filesystem.
- #define NMMGR_TYPE_BLOCKDEV 0x0020 /* Block device */

  A block device.
- #define NMMGR_TYPE_SINGLETON 0x0030 /* Singleton service (e.g., /dev/irq) */

  A singleton service (e.g., /dev/irq)
- #define NMMGR_TYPE_SYMTAB 0x0040 /* Symbol table */

  A symbol table.
- #define NMMGR_SYS_MAX 0x10000 /* Here and above are user types */

  Everything this and above is a user type.

Functions

- typedef LIST_HEAD (nmmgr_list, nmmgr_handler) nmmgr_list_t

  Name handler list type.
- nmmgr_handler_t * nmmgr_lookup (const char *name)

  Retrieve a name handler by name.
- nmmgr_list_t * nmmgr_get_list (void)

  Get the head element of the name list.
- int nmmgr_handler_add (nmmgr_handler_t *hnd)

  Add a name handler.
- int nmmgr_handler_remove (nmmgr_handler_t *hnd)

  Remove a name handler.

9.70.1 Detailed Description

Name manager.

This file contains the definitions of KOS' name manager. A "name" is a generic identifier for some kind of module. These modules may include services provided by the kernel (such as VFS handlers).

Author

Megan Potter

9.70.2 Macro Definition Documentation

NMMGR_FLAGS_NEEDSFREE

```
#define NMMGR_FLAGS_NEEDSFREE 0x00000001
```

This structure must be freed when removed.

NMMGR_LIST_INIT

```
#define NMMGR_LIST_INIT { NULL }
```

List entry initializer for static structs.

If you are creating nmmgr handlers, this is what you should initialize the list_ent member with.

9.70.3 Function Documentation

LIST_HEAD()

```
typedef LIST_HEAD (
             nmmgr_list ,
             nmmgr_handler  )
```

Name handler list type.

Contrary to what doxygen may think, this is not a function.

nmmgr_get_list()

```
nmmgr_list_t * nmmgr_get_list (
             void  )
```

Get the head element of the name list.

DO NOT MODIFY THE VALUE RETURNED BY THIS FUNCTION! In fact, don't ever call this function.

Returns

The head of the name handler list

nmmgr_handler_add()

```
int nmmgr_handler_add (
             nmmgr_handler_t * hnd )
```

Add a name handler.

This function adds a new name handler to the list in the kernel.

Parameters

hnd	The handler to add

Return values

0	On success

nmmgr_handler_remove()

```
int nmmgr_handler_remove (
```

```
            nmmgr_handler_t * hnd )
```

Remove a name handler.

This function removes a name handler from the list in the kernel.

Parameters

hnd	The handler to remove

Return values

0	On success
-1	If the handler wasn't found

nmmgr_lookup()

```
nmmgr_handler_t * nmmgr_lookup (
            const char * name )
```

Retrieve a name handler by name.

This function will retrieve a name handler by its pathname.

Parameters

name	The handler to look up

Returns

The handler, or NULL on failure.

9.71 nmmgr.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/nmmgr.h
00004    Copyright (C)2003 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/nmmgr.h
00009     \brief  Name manager.
00010
00011     This file contains the definitions of KOS' name manager. A "name" is a
00012     generic identifier for some kind of module. These modules may include
00013     services provided by the kernel (such as VFS handlers).
00014
00015     \author Megan Potter
00016 */
00017
```

```
00018 #ifndef __KOS_NMMGR_H
00019 #define __KOS_NMMGR_H
00020
00021 #include <sys/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 #include <arch/types.h>
00025 #include <kos/limits.h>
00026 #include <sys/queue.h>
00027
00028 /* Pre-define list types */
00029 struct nmmgr_handler;
00030
00031 /** \brief  Name handler list type.
00032
00033     Contrary to what doxygen may think, this is not a function.
00034 */
00035 typedef LIST_HEAD(nmmgr_list, nmmgr_handler) nmmgr_list_t;
00036
00037 /** \brief  List entry initializer for static structs.
00038
00039     If you are creating nmmgr handlers, this is what you should initialize the
00040     list_ent member with.
00041 */
00042 #define NMMGR_LIST_INIT { NULL }
00043
00044 /** \brief  Name handler interface.
00045
00046     Every name handler must begin its information structures with this header.
00047     If the handler conforms to some well-defined interface (such as a VFS), then
00048     the struct must more specifically be of that type.
00049
00050     \headerfile kos/nmmgr.h
00051 */
00052 typedef struct nmmgr_handler {
00053     char    pathname[NAME_MAX];   /* Path name */
00054     int pid;            /* Process table ID for handler (0 == static) */
00055     uint32  version;        /* Version code */
00056     uint32  flags;          /* Bitmask of flags */
00057     uint32  type;           /* Type of handler */
00058     LIST_ENTRY(nmmgr_handler)   list_ent;   /* Linked list entry */
00059 } nmmgr_handler_t;
00060
00061 /* Version codes ('version') have two pieces: a major and minor revision.
00062    A major revision (top 16 bits) means that the interfaces are totally
00063    incompatible. A minor revision (lower 16 bits) diffrentiates between
00064    mostly-compatible but newer/older revisions of the implementing code. */
00065
00066 /* Flag bits */
00067 /** \brief  This structure must be freed when removed. */
00068 #define NMMGR_FLAGS_NEEDSFREE   0x00000001
00069
00070 /** \defgroup   nmmgr_types     Name handler types
00071
00072     This is the set of all defined types of name manager handlers. All system
00073     types are defined below NMMGR_SYS_MAX.
00074
00075     @{
00076 */
00077 /** \brief  Unknown nmmgr type. */
00078 #define NMMGR_TYPE_UNKNOWN  0x0000      /* ? */
00079 /** \brief  A mounted filesystem. */
00080 #define NMMGR_TYPE_VFS      0x0010      /* Mounted file system */
00081 /** \brief  A block device. */
00082 #define NMMGR_TYPE_BLOCKDEV 0x0020      /* Block device */
00083 /** \brief  A singleton service (e.g., /dev/irq) */
00084 #define NMMGR_TYPE_SINGLETON    0x0030      /* Singleton service (e.g., /dev/irq) */
00085 /** \brief  A symbol table. */
00086 #define NMMGR_TYPE_SYMTAB   0x0040      /* Symbol table */
00087 /** \brief  Everything this and above is a user type. */
00088 #define NMMGR_SYS_MAX       0x10000     /* Here and above are user types */
00089 /** @} */
00090
00091 /** \brief  Retrieve a name handler by name.
00092
00093     This function will retrieve a name handler by its pathname.
00094
00095     \param  name            The handler to look up
00096     \return                 The handler, or NULL on failure.
00097 */
00098 nmmgr_handler_t * nmmgr_lookup(const char *name);
```

```
00099
00100 /** \brief  Get the head element of the name list.
00101
00102     DO NOT MODIFY THE VALUE RETURNED BY THIS FUNCTION! In fact, don't ever call
00103     this function.
00104
00105     \return                 The head of the name handler list
00106 */
00107 nmmgr_list_t * nmmgr_get_list(void);
00108
00109 /** \brief  Add a name handler.
00110
00111     This function adds a new name handler to the list in the kernel.
00112
00113     \param  hnd             The handler to add
00114     \retval 0               On success
00115 */
00116 int nmmgr_handler_add(nmmgr_handler_t *hnd);
00117
00118 /** \brief  Remove a name handler.
00119
00120     This function removes a name handler from the list in the kernel.
00121
00122     \param  hnd             The handler to remove
00123     \retval 0               On success
00124     \retval -1              If the handler wasn't found
00125 */
00126 int nmmgr_handler_remove(nmmgr_handler_t *hnd);
00127
00128 /** \cond */
00129 /* Name manager init */
00130 void nmmgr_init(void);
00131 void nmmgr_shutdown(void);
00132 /** \endcond */
00133
00134 __END_DECLS
00135
00136 #endif  /* __KOS_NMMGR_H */
00137
```

9.72 include/kos/once.h File Reference

Dynamic package initialization.

```
#include <sys/cdefs.h>
```

Macros

- #define KTHREAD_ONCE_INIT 0

  Initializer for a kthread_once_t object.

Typedefs

- typedef volatile int kthread_once_t

  Object type backing kthread_once.

Functions

- int kthread_once (kthread_once_t *once_control, void(*init_routine)(void))

  Run a function once.

9.72.1 Detailed Description

Dynamic package initialization.

This file provides definitions for an object that functions the same way as the pthread_once_t function does from the POSIX specification. This object type and functionality is generally used to make sure that a given initialization function is run once, and only once, no matter how many threads attempt to call it.

Author

Lawrence Sebald

9.72.2 Macro Definition Documentation

KTHREAD_ONCE_INIT

```
#define KTHREAD_ONCE_INIT 0
```

Initializer for a kthread_once_t object.

9.72.3 Typedef Documentation

kthread_once_t

```
typedef volatile int kthread_once_t
```

Object type backing kthread_once.

This object type should always be initialized with the KTHREAD_ONCE_INIT macro.

9.72.4 Function Documentation

kthread_once()

```
int kthread_once (
            kthread_once_t * once_control,
            void(*)(void) init_routine )
```

Run a function once.

This function, when used with a kthread_once_t object (that should be shared amongst all threads) will run the init_↩ routine once, and set the once_control to make sure that the function will not be run again (as long as all threads attempt to call the init_routine through this function.

Parameters

once_control	The kthread_once_t object to run against.
init_routine	The function to call.

Return values

-1	On failure, and sets errno to one of the following: EPERM if called inside an interrupt or EINVAL if ∗once_control is not valid or was not initialized with KTHREAD_ONCE_INIT.
0	On success.

9.73 once.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    include/kos/once.h
00004    Copyright (C) 2009, 2010, 2023 Lawrence Sebald
00005
00006 */
00007
00008 #ifndef __KOS_ONCE_H
00009 #define __KOS_ONCE_H
00010
00011 /** \file   kos/once.h
00012     \brief  Dynamic package initialization.
00013     \ingroup kthreads
00014
00015     This file provides definitions for an object that functions the same way as
00016     the pthread_once_t function does from the POSIX specification. This object
00017     type and functionality is generally used to make sure that a given
00018     initialization function is run once, and only once, no matter how many
00019     threads attempt to call it.
00020
00021     \author Lawrence Sebald
00022 */
00023
00024 #include <sys/cdefs.h>
00025
00026 __BEGIN_DECLS
00027
00028 /** \brief  Object type backing kthread_once.
00029
00030     This object type should always be initialized with the KTHREAD_ONCE_INIT
00031     macro.
00032
00033     \headerfile kos/once.h
00034 */
00035 typedef volatile int kthread_once_t;
00036
00037 /** \brief  Initializer for a kthread_once_t object. */
00038 #define KTHREAD_ONCE_INIT   0
00039
00040 /** \brief  Run a function once.
00041
00042     This function, when used with a kthread_once_t object (that should be shared
00043     amongst all threads) will run the init_routine once, and set the
00044     once_control to make sure that the function will not be run again (as long
00045     as all threads attempt to call the init_routine through this function.
00046
00047     \param  once_control    The kthread_once_t object to run against.
00048     \param  init_routine    The function to call.
00049     \retval -1      On failure, and sets errno to one of the following: EPERM if
00050                     called inside an interrupt or EINVAL if *once_control is not
00051                     valid or was not initialized with KTHREAD_ONCE_INIT.
00052     \retval 0       On success. */
00053 int kthread_once(kthread_once_t *once_control, void (*init_routine)(void));
```

```
00054
00055 __END_DECLS
00056
00057 #endif /* !__KOS_ONCE_H */
```

9.74 include/kos/opts.h File Reference

Compile-time options regarding debugging and other topics.

```
#include <sys/cdefs.h>
```

Macros

- #define KOS_DEBUG 0
- #define FS_CD_MAX_FILES 8

  The maximum number of cd files that can be open at a time.
- #define FS_ROMDISK_MAX_FILES 16

  The maximum number of romdisk files that can be open at a time.
- #define FS_RAMDISK_MAX_FILES 8

  The maximum number of ramdisk files that can be open at a time.

9.74.1 Detailed Description

Compile-time options regarding debugging and other topics.

This file is meant to be a kind of Grand Central Station for all of the various compile-time options that can be set when building KOS. Each of the various compile-time macros that control things like additional debugging checks and such should be documented in this particular file. In addition, groups of related macros are provided here to make it easier to set these options at compile time.

Basically, this is here to make it easier to customize your copy of KOS. In the past, you would have had to search through all the various files to find potential options, which was both annoying and error-prone. In addition, often various debugging flags would end up being set in the main repository as people debugged individual pieces of code, leaving them set for everyone else, even if they aren't necessarily useful/helpful all the time (KM_DBG, I'm looking at you).

Author

Lawrence Sebald

9.74.2 Macro Definition Documentation

FS_CD_MAX_FILES

```
#define FS_CD_MAX_FILES 8
```

The maximum number of cd files that can be open at a time.

FS_RAMDISK_MAX_FILES

```
#define FS_RAMDISK_MAX_FILES 8
```

The maximum number of ramdisk files that can be open at a time.

FS_ROMDISK_MAX_FILES

```
#define FS_ROMDISK_MAX_FILES 16
```

The maximum number of romdisk files that can be open at a time.

KOS_DEBUG

```
#define KOS_DEBUG 0
```

9.75 opts.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/opts.h
00004    Copyright (C) 2014 Lawrence Sebald
00005 */
00006
00007 #ifndef __KOS_OPTS_H
00008 #define __KOS_OPTS_H
00009
00010 #include <sys/cdefs.h>
00011 __BEGIN_DECLS
00012
00013 /** \file   kos/opts.h
00014     \brief  Compile-time options regarding debugging and other topics.
00015
00016     This file is meant to be a kind of Grand Central Station for all of the
00017     various compile-time options that can be set when building KOS. Each of the
00018     various compile-time macros that control things like additional debugging
00019     checks and such should be documented in this particular file. In addition,
00020     groups of related macros are provided here to make it easier to set these
00021     options at compile time.
00022
00023     Basically, this is here to make it easier to customize your copy of KOS. In
00024     the past, you would have had to search through all the various files to find
00025     potential options, which was both annoying and error-prone. In addition,
00026     often various debugging flags would end up being set in the main repository
00027     as people debugged individual pieces of code, leaving them set for everyone
00028     else, even if they aren't necessarily useful/helpful all the time (KM_DBG,
00029     I'm looking at you).
00030
00031     \author Lawrence Sebald
00032 */
00033
00034 /* Various debug options. Uncomment the #define line to enable the specific
00035    option described. */
00036
00037 /* Enable debugging in fs_vmu. */
00038 /* #define VMUFS_DEBUG 1 */
00039
00040
00041 /* Enable to allow extra debugging checks in the malloc code itself. This
00042    sometimes catches corrupted blocks. Recommended during debugging phases. */
00043 /* #define MALLOC_DEBUG 1 */
00044
```

```
00045 /* Enable this define if you want costly malloc debugging (buffer sentinel
00046    checking, block leak checking, etc). Recommended during debugging phases, but
00047    you should probably take it out before you start your final testing. */
00048 /* #define KM_DBG 1 */
00049
00050 /* Enable this define if you want REALLY verbose malloc debugging (print every
00051    time a block is allocated or freed). Only enable this if you are having some
00052    serious issues. */
00053 /* #define KM_DBG_VERBOSE 1 */
00054
00055
00056 /* The following three macros are similar to the ones above, but for the PVR
00057    memory pool malloc. */
00058 /* #define PVR_MALLOC_DEBUG 1 */
00059 /* #define PVR_KM_DBG 1 */
00060 /* #define PVR_KM_DBG_VERBOSE 1 */
00061
00062 /* Enable this define to enable PVR error interrupts and to have the interrupt
00063    handler print them when they occur.  */
00064 /* #define PVR_RENDER_DBG */
00065
00066 /* Aggregate debugging levels. It's probably best to enable these with your
00067    KOS_CFLAGS when compiling KOS itself, but they're all documented here and
00068    can be enabled here, if you really want to. */
00069
00070 /* Enable all recommended options for normal debugging use. This enables the
00071    first level of malloc debugging for the main pool, as well as the internal
00072    dlmalloc debugging code for the main pool. This is essentially the set that
00073    was normally enabled in the repository tree. */
00074 /* #define KOS_DEBUG 1 */
00075
00076 /* Enable verbose debugging support. Basically, if your code is crashing and the
00077    stuff in the first level doesn't help, then use this one instead. This adds
00078    in verbose malloc debugging in the main pool, as well as basic PVR malloc
00079    debugging. */
00080 /* #define KOS_DEBUG 2 */
00081
00082 /* Enable verbose debugging support, plus additional debugging options that
00083    normally wouldn't be needed. Once again, try lighter levels before this one,
00084    as there's not much reason this one should ever be needed. This level
00085    includes the internal debugging stuff in fs_vmu, verbose debugging in the PVR
00086    malloc, as well as everything in level 2. */
00087 /* #define KOS_DEBUG 3 */
00088
00089 #ifndef KOS_DEBUG
00090 #define KOS_DEBUG 0
00091 #endif
00092
00093 #if KOS_DEBUG >= 1
00094 #define MALLOC_DEBUG 1
00095 #define KM_DBG 1
00096 #endif
00097
00098 #if KOS_DEBUG >= 2
00099 #define KM_DBG_VERBOSE 1
00100 #define PVR_MALLOC_DEBUG 1
00101 #define PVR_KM_DBG 1
00102 #endif
00103
00104 #if KOS_DEBUG >= 3
00105 #define PVR_KM_DBG_VERBOSE 1
00106 #define VMUFS_DEBUG 1
00107 #endif
00108
00109 /** \brief  The maximum number of cd files that can be open at a time. */
00110 #ifndef FS_CD_MAX_FILES
00111 #define FS_CD_MAX_FILES 8
00112 #endif
00113
00114 /** \brief  The maximum number of romdisk files that can be open at a time. */
00115 #ifndef FS_ROMDISK_MAX_FILES
00116 #define FS_ROMDISK_MAX_FILES 16
00117 #endif
00118
00119 /** \brief  The maximum number of ramdisk files that can be open at a time. */
00120 #ifndef FS_RAMDISK_MAX_FILES
00121 #define FS_RAMDISK_MAX_FILES 8
00122 #endif
00123
00124 __END_DECLS
00125
```

```
00126 #endif /* !__KOS_OPTS_H */
```

9.76 include/kos/recursive_lock.h File Reference

Definitions for a recursive mutex.

```
#include <kos/cdefs.h>
#include <kos/mutex.h>
```

Typedefs

- typedef mutex_t recursive_lock_t

  Recursive lock structure.

Functions

- recursive_lock_t * rlock_create (void) __depr("Use mutexes instead.")

  Allocate a new recursive lock.
- void rlock_destroy (recursive_lock_t *l) __depr("Use mutexes instead.")

  Destroy a recursive lock.
- int rlock_lock (recursive_lock_t *l) __depr("Use mutexes instead.")

  Lock a recursive lock.
- int rlock_lock_timed (recursive_lock_t *l, int timeout) __depr("Use mutexes instead.")

  Lock a recursive lock (with a timeout).
- int rlock_unlock (recursive_lock_t *l) __depr("Use mutexes instead.")

  Unlock a recursive lock.
- int rlock_trylock (recursive_lock_t *l) __depr("Use mutexes instead.")

  Attempt to lock a recursive lock without blocking.
- int rlock_is_locked (recursive_lock_t *l) __depr("Use mutexes instead.")

  Check if a recursive lock is currently held by any thread.

9.76.1 Detailed Description

Definitions for a recursive mutex.

This file defines a recursive lock mechanism, similar to a mutex, but that a single thread can obtain as many times as it wants. A single thread is still limited to holding the lock at a time, but it can hold it an "unlimited" number of times (actually limited to INT_MAX, but who's counting).

Deprecated These are now just wrappers around the MUTEX_TYPE_RECURSIVE that is now provided and will be removed at some point in the future. Please update your code to use that type instead.

Author

Lawrence Sebald

9.76.2 Typedef Documentation

recursive_lock_t

```
typedef mutex_t recursive_lock_t
```

Recursive lock structure.

Recursive locks are just a simple wrapper around mutexes at this point. You should not use this type in any new code.

9.76.3 Function Documentation

rlock_create()

```
recursive_lock_t * rlock_create (
            void  )
```

Allocate a new recursive lock.

Deprecated This function allocates a new recurisve lock that is initially not locked.

Returns

The created lock, or NULL on failure (errno will be set to ENOMEM to indicate that the system appears to be out of memory).

rlock_destroy()

```
void rlock_destroy (
            recursive_lock_t * l )
```

Destroy a recursive lock.

Deprecated This function cleans up a recursive lock. It is an error to attempt to destroy a locked recursive lock.

Parameters

l	The recursive lock to destroy. It must be unlocked.

rlock_is_locked()

```
int rlock_is_locked (
```

```
            recursive_lock_t * l )
```

Check if a recursive lock is currently held by any thread.

Deprecated This function checks whether or not a lock is currently held by any thread, including the calling thread. Note that this is **NOT** a safe way to check if a lock *will* be held by the time you get around to locking it.

Return values

TRUE	If the lock is held by any thread.
FALSE	If the lock is not currently held by any thread.

rlock_lock()

```
int rlock_lock (
            recursive_lock_t * l )
```

Lock a recursive lock.

Deprecated This function attempts to lock the requested lock, and if it cannot it will block until that is possible.

Parameters

l	The recursive lock to lock.

Return values

-1	On error, errno will be set to EPERM if this function is called inside an interrupt, or EINTR if it is interrupted.
0	On success.

See also

rlock_trylock

rlock_lock_timed

rlock_lock_timed()

```
int rlock_lock_timed (
            recursive_lock_t * l,
            int timeout )
```

Lock a recursive lock (with a timeout).

Deprecated This function attempts to lock the requested lock, and if it cannot it will block until either it is possible to acquire the lock or timeout milliseconds have elapsed.

Parameters

l	The recursive lock to lock.
timeout	The maximum number of milliseconds to wait. 0 is an unlimited timeout (equivalent to rlock_lock).

Return values

-1	On error, errno will be set to EPERM if this function is called inside an interrupt, EINTR if the function is interrupted, or EAGAIN if the timeout expires.
0	On success.

See also

rlock_trylock

rlock_lock_timed

rlock_trylock()

```
int rlock_trylock (
            recursive_lock_t * l )
```

Attempt to lock a recursive lock without blocking.

Deprecated This function attempts to lock a recursive lock without blocking. This function, unlike rlock_lock and rlock↩ _lock_timed is safe to call inside an interrupt.

Parameters

l	The recursive lock to lock.

Return values

-1	On error, errno will be set to EWOULDBLOCK if the lock is currently held by another thread.
0	On success.

See also

rlock_lock

rlock_lock_timed

rlock_unlock()

```
int rlock_unlock (
            recursive_lock_t * l )
```

Unlock a recursive lock.

Deprecated This function releases the lock one time from the current thread.

Parameters

l	The recursive lock to unlock.

Return values

-1	On error, errno will be set to EPERM if the lock is not held by the calling thread.
0	On success.

9.77 recursive_lock.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    include/kos/recursive_lock.h
00004    Copyright (C) 2008, 2010, 2012 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   kos/recursive_lock.h
00009     \brief  Definitions for a recursive mutex.
00010     \ingroup kthreads
00011
00012     This file defines a recursive lock mechanism, similar to a mutex, but that a
00013     single thread can obtain as many times as it wants. A single thread is still
00014     limited to holding the lock at a time, but it can hold it an "unlimited"
00015     number of times (actually limited to INT_MAX, but who's counting).
00016
00017     \deprecated
00018     These are now just wrappers around the MUTEX_TYPE_RECURSIVE that is now
00019     provided and will be removed at some point in the future. Please update your
00020     code to use that type instead.
00021
00022     \author Lawrence Sebald
00023 */
00024
00025 #ifndef __KOS_RECURSIVE_LOCK_H
00026 #define __KOS_RECURSIVE_LOCK_H
00027
00028 #include <kos/cdefs.h>
00029
00030 __BEGIN_DECLS
00031
00032 #include <kos/mutex.h>
00033
00034 /** \brief  Recursive lock structure.
00035
00036     Recursive locks are just a simple wrapper around mutexes at this point. You
00037     should not use this type in any new code.
00038
00039     \headerfile kos/recursive_lock.h
00040 */
00041 typedef mutex_t recursive_lock_t;
00042
00043 /** \brief  Allocate a new recursive lock.
00044
00045     \deprecated
00046     This function allocates a new recurisve lock that is initially not locked.
00047
00048     \return The created lock, or NULL on failure (errno will be set to ENOMEM to
```

```
00049             indicate that the system appears to be out of memory).
00050 */
00051 recursive_lock_t *rlock_create(void) __depr("Use mutexes instead.");
00052
00053 /** \brief  Destroy a recursive lock.
00054
00055     \deprecated
00056     This function cleans up a recursive lock. It is an error to attempt to
00057     destroy a locked recursive lock.
00058
00059     \param  l       The recursive lock to destroy. It must be unlocked.
00060 */
00061 void rlock_destroy(recursive_lock_t *l) __depr("Use mutexes instead.");
00062
00063 /** \brief  Lock a recursive lock.
00064
00065     \deprecated
00066     This function attempts to lock the requested lock, and if it cannot it will
00067     block until that is possible.
00068
00069     \param  l       The recursive lock to lock.
00070     \retval -1      On error, errno will be set to EPERM if this function is
00071                     called inside an interrupt, or EINTR if it is interrupted.
00072     \retval 0       On success.
00073     \sa rlock_trylock
00074     \sa rlock_lock_timed
00075 */
00076 int rlock_lock(recursive_lock_t *l) __depr("Use mutexes instead.");
00077
00078 /** \brief  Lock a recursive lock (with a timeout).
00079
00080     \deprecated
00081     This function attempts to lock the requested lock, and if it cannot it will
00082     block until either it is possible to acquire the lock or timeout
00083     milliseconds have elapsed.
00084
00085     \param  l       The recursive lock to lock.
00086     \param  timeout The maximum number of milliseconds to wait. 0 is an
00087                     unlimited timeout (equivalent to rlock_lock).
00088     \retval -1      On error, errno will be set to EPERM if this function is
00089                     called inside an interrupt, EINTR if the function is
00090                     interrupted, or EAGAIN if the timeout expires.
00091     \retval 0       On success.
00092     \sa rlock_trylock
00093     \sa rlock_lock_timed
00094 */
00095 int rlock_lock_timed(recursive_lock_t *l, int timeout)
00096     __depr("Use mutexes instead.");
00097
00098 /** \brief  Unlock a recursive lock.
00099
00100     \deprecated
00101     This function releases the lock one time from the current thread.
00102
00103     \param  l       The recursive lock to unlock.
00104     \retval -1      On error, errno will be set to EPERM if the lock is not held
00105                     by the calling thread.
00106     \retval 0       On success.
00107 */
00108 int rlock_unlock(recursive_lock_t *l) __depr("Use mutexes instead.");
00109
00110 /** \brief  Attempt to lock a recursive lock without blocking.
00111
00112     \deprecated
00113     This function attempts to lock a recursive lock without blocking. This
00114     function, unlike rlock_lock and rlock_lock_timed is safe to call inside an
00115     interrupt.
00116
00117     \param  l       The recursive lock to lock.
00118     \retval -1      On error, errno will be set to EWOULDBLOCK if the lock is
00119                     currently held by another thread.
00120     \retval 0       On success.
00121     \sa rlock_lock
00122     \sa rlock_lock_timed
00123 */
00124 int rlock_trylock(recursive_lock_t *l) __depr("Use mutexes instead.");
00125
00126 /** \brief  Check if a recursive lock is currently held by any thread.
00127
00128     \deprecated
00129     This function checks whether or not a lock is currently held by any thread,
```

```
00130     including the calling thread. Note that this is <b>NOT</b> a safe way to
00131     check if a lock <em>will</em> be held by the time you get around to locking
00132     it.
00133
00134     \retval TRUE    If the lock is held by any thread.
00135     \retval FALSE   If the lock is not currently held by any thread.
00136 */
00137 int rlock_is_locked(recursive_lock_t *l) __depr("Use mutexes instead.");
00138
00139 __END_DECLS
00140
00141 #endif /* !__KOS_RECURSIVE_LOCK_H */
```

9.78 include/kos/rwsem.h File Reference

Definition for a reader/writer semaphore.

```
#include <kos/cdefs.h>
#include <stddef.h>
#include <kos/thread.h>
```

Data Structures

- struct rw_semaphore_t

  Reader/writer semaphore structure.

Macros

- #define RWSEM_INITIALIZER { 0, 0, NULL, NULL }

  Initializer for a transient reader/writer semaphore.

Functions

- rw_semaphore_t * rwsem_create (void) __depr("Use rwsem_init or RWSEM_INITIALIZER.")

  Allocate a reader/writer semaphore.
- int rwsem_init (rw_semaphore_t *s)

  Initialize a reader/writer semaphore.
- int rwsem_destroy (rw_semaphore_t *s)

  Destroy a reader/writer semaphore.
- int rwsem_read_lock_timed (rw_semaphore_t *s, int timeout)

  Lock a reader/writer semaphore for reading (with a timeout).
- int rwsem_read_lock (rw_semaphore_t *s)

  Lock a reader/writer semaphore for reading.
- int rwsem_write_lock_timed (rw_semaphore_t *s, int timeout)

  Lock a reader/writer semaphore for writing (with a timeout).
- int rwsem_write_lock (rw_semaphore_t *s)

  Lock a reader/writer semaphore for writing.
- int rwsem_read_unlock (rw_semaphore_t *s)

  Unlock a reader/writer semaphore from a read lock.

- int rwsem_write_unlock (rw_semaphore_t *s)

  Unlock a reader/writer semaphore from a write lock.
- int rwsem_unlock (rw_semaphore_t *s)

  Unlock a reader/writer semaphore.
- int rwsem_read_trylock (rw_semaphore_t *s)

  Attempt to lock a reader/writer semaphore for reading.
- int rwsem_write_trylock (rw_semaphore_t *s)

  Attempt to lock a reader/writer semaphore for writing.
- int rwsem_read_upgrade_timed (rw_semaphore_t *s, int timeout)

  Upgrade a thread from reader status to writer status (with a timeout).
- int rwsem_read_upgrade (rw_semaphore_t *s)

  Upgrade a thread from reader status to writer status.
- int rwsem_read_tryupgrade (rw_semaphore_t *s)

  Attempt to upgrade a thread from reader status to writer status.
- int rwsem_read_count (rw_semaphore_t *s)

  Read the reader count on the reader/writer semaphore.
- int rwsem_write_locked (rw_semaphore_t *s)

  Read the state of the writer lock on the reader/writer semaphore.

9.78.1 Detailed Description

Definition for a reader/writer semaphore.

This file defines a concept of reader/writer semaphores. Basically, this type of lock allows an unlimited number of "readers" to acquire the lock at a time, but only one "writer" (and only if no readers hold the lock). Readers, by definition, should not change any global data (since they are defined to only be reading), and since this is the case it is safe to allow multiple readers to access global data that is shared amongst threads. Writers on the other hand require exclusive access since they will be changing global data in the critical section, and they cannot share with a reader either (since the reader might attempt to read while the writer is changing data).

Author

Lawrence Sebald

9.78.2 Macro Definition Documentation

RWSEM_INITIALIZER

```
#define RWSEM_INITIALIZER { 0, 0, NULL, NULL }
```

Initializer for a transient reader/writer semaphore.

9.78.3 Function Documentation

rwsem_create()

```
rw_semaphore_t * rwsem_create (
            void  )
```

Allocate a reader/writer semaphore.

This function allocates a new reader/writer lock that is initially not locked either for reading or writing.

Deprecated This function is formally deprecated, and should not be used in newly written code. Instead, please use rwsem_init().

Returns

The created semaphore, or NULL on failure (errno will be set as appropriate).

Error Conditions:

ENOMEM - out of memory

rwsem_destroy()

```
int rwsem_destroy (
            rw_semaphore_t * s )
```

Destroy a reader/writer semaphore.

This function cleans up a reader/writer semaphore. It is an error to attempt to destroy a r/w semaphore that is locked either for reading or writing.

Parameters

s	The r/w semaphore to destroy.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

EBUSY - the semaphore is still locked

rwsem_init()

```
int rwsem_init (
            rw_semaphore_t * s )
```

Initialize a reader/writer semaphore.

This function initializes a new reader/writer semaphore for use.

Return values

0	On success (no error conditions currently defined).

rwsem_read_count()

```
int rwsem_read_count (
            rw_semaphore_t * s )
```

Read the reader count on the reader/writer semaphore.

This function is not a safe way to see if the lock will be locked by any readers when you get around to locking it, so do not use it in this way.

Parameters

s	The r/w semaphore to count the readers on.

Returns

The number of readers holding the r/w semaphore.

rwsem_read_lock()

```
int rwsem_read_lock (
            rw_semaphore_t * s )
```

Lock a reader/writer semaphore for reading.

This function attempts to lock the r/w semaphore for reading. If the semaphore is locked for writing, this function will block until it is possible to obtain the lock for reading. This function is **NOT** safe to call inside of an interrupt.

Parameters

s	The r/w semaphore to lock.

Return values

0	On success
-1	On error, errno will be set as appropriate.

Error Conditions:

EPERM - called inside an interrupt
EINVAL - the semaphore is not initialized

rwsem_read_lock_timed()

```
int rwsem_read_lock_timed (
            rw_semaphore_t * s,
            int timeout )
```

Lock a reader/writer semaphore for reading (with a timeout).

This function attempts to lock the r/w semaphore for reading. If the semaphore is locked for writing, this function will block until it is possible to obtain the lock for reading or the timeout expires. This function is **NOT** safe to call inside of an interrupt.

Parameters

s	The r/w semaphore to lock.
timeout	The maximum time to wait (in milliseconds).

Return values

0	On success
-1	On error, errno will be set as appropriate.

Error Conditions:

EPERM - called inside an interrupt
ETIMEDOUT - the timeout expires before the lock can be acquired
EINVAL - the timeout value is invalid
EINVAL - the semaphore is not initialized

rwsem_read_trylock()

```
int rwsem_read_trylock (
            rw_semaphore_t * s )
```

Attempt to lock a reader/writer semaphore for reading.

This function attempts to lock the r/w semaphore for reading. If for any reason rwsem_read_lock would normally block, this function will return an error. This function is safe to call inside an interrupt.

Parameters

s	The r/w semaphore to attempt to lock.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

EWOULDBLOCK - a call to rwsem_read_lock would block
EINVAL - the semaphore is not initialized

rwsem_read_tryupgrade()

```
int rwsem_read_tryupgrade (
            rw_semaphore_t * s )
```

Attempt to upgrade a thread from reader status to writer status.

This function will attempt to upgrade the lock on the calling thread to writer status. If for any reason rwsem_read_↩ upgrade would block, this function will return an error. This function is safe to call inside an interrupt. Note that on error, the read lock is still held!

Parameters

s	The r/w semaphore to upgrade.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

EWOULDBLOCK - a call to rwsem_read_upgrade would block
EBUSY - another reader has already requested an upgrade
EINVAL - the sempahore is not initialized

rwsem_read_unlock()

```
int rwsem_read_unlock (
            rw_semaphore_t * s )
```

Unlock a reader/writer semaphore from a read lock.

This function releases one instance of the read lock on the r/w semaphore.

Parameters

s	The r/w semaphore to release the read lock on.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

EPERM - the read lock is not currently held
EINVAL - the semaphore is not initialized

rwsem_read_upgrade()

```
int rwsem_read_upgrade (
            rw_semaphore_t * s )
```

Upgrade a thread from reader status to writer status.

This function will upgrade the lock on the calling thread from a reader state to a writer state. If it cannot do this at the moment, it will block until it is possible. This function is **NOT** safe to call inside an interrupt.

You can only have one reader waiting to upgrade at a time, otherwise the state would potentially become corrupted between when this is called and when you get the lock. If you get -1 back from this, you must not assume that you can write safely! On error, the calling thread will still hold a read lock.

Parameters

s	The r/w semaphore to upgrade.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

EPERM - called inside an interrupt
EINVAL - the semaphore is not initialized
EBUSY - another reader has already requested an upgrade

rwsem_read_upgrade_timed()

```
int rwsem_read_upgrade_timed (
```

```
            rw_semaphore_t * s,
            int timeout )
```

Upgrade a thread from reader status to writer status (with a timeout).

This function will upgrade the lock on the calling thread from a reader state to a writer state. If it cannot do this at the moment, it will block until it is possible. This function is **NOT** safe to call inside an interrupt.

You can only have one reader waiting to upgrade at a time, otherwise the state would potentially become corrupted between when this is called and when you get the lock. If you get -1 back from this, you must not assume that you can write safely! On error, the calling thread will still hold a read lock.

Parameters

s	The r/w semaphore to upgrade.
timeout	The maximum time to wait (in milliseconds).

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

EPERM - called inside an interrupt
EINVAL - the semaphore is not initialized
EINVAL - the timeout value is invalid
EBUSY - another reader has already requested an upgrade
ETIMEDOUT - the timeout expired before the write lock could be acquired

rwsem_unlock()

```
int rwsem_unlock (
            rw_semaphore_t * s )
```

Unlock a reader/writer semaphore.

This function releases the lock held by the current thread on the specified reader/writer semaphore. This function will automatically determine which lock is held by the calling thread and release it as appropriate.

This function is **NOT** safe to call (in general) if you do not hold the lock!

Parameters

s	The r/w semaphore to release the lock on.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

EPERM - the lock is not currently held by the calling thread
EINVAL - the semaphore is not initialized

rwsem_write_lock()

```
int rwsem_write_lock (
            rw_semaphore_t * s )
```

Lock a reader/writer semaphore for writing.

This function attempts to lock the r/w semaphore for writing. If the semaphore is locked for reading or writing, this function will block until it is possible to obtain the lock for writing. This function is **NOT** safe to call inside of an interrupt.

Parameters

s	The r/w semaphore to lock.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error conditions:

EPERM - called inside an interrupt
EINVAL - the semaphore is not initialized

rwsem_write_lock_timed()

```
int rwsem_write_lock_timed (
            rw_semaphore_t * s,
            int timeout )
```

Lock a reader/writer semaphore for writing (with a timeout).

This function attempts to lock the r/w semaphore for writing. If the semaphore is locked for reading or writing, this function will block until it is possible to obtain the lock for writing or the timeout expires. This function is **NOT** safe to call inside of an interrupt.

Parameters

s	The r/w semaphore to lock.
timeout	The maximum time to wait (in milliseconds).

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

EPERM - called inside an interrupt
ETIMEDOUT - the timeout expires before the lock can be acquired
EINVAL - the timeout value is invalid
EINVAL - the semaphore is not initialized

rwsem_write_locked()

```
int rwsem_write_locked (
            rw_semaphore_t * s )
```

Read the state of the writer lock on the reader/writer semaphore.

This function is not a safe way to see if the lock will be locked by a writer by the time you get around to doing something with it, so don't try to use it for that purpose.

Parameters

s	The r/w semaphore to check the writer status on.

Returns

The status of the writer lock of the r/w semaphore.

rwsem_write_trylock()

```
int rwsem_write_trylock (
            rw_semaphore_t * s )
```

Attempt to lock a reader/writer semaphore for writing.

This function attempts to lock the r/w semaphore for writing. If for any reason rwsem_write_lock would normally block, this function will return an error. This function is safe to call inside an interrupt.

Parameters

s	The r/w semaphore to attempt to lock.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

EWOULDBLOCK - a call to rwsem_write_lock would block
EINVAL - the semaphore is not initialized

rwsem_write_unlock()

```
int rwsem_write_unlock (
            rw_semaphore_t * s )
```

Unlock a reader/writer semaphore from a write lock.

This function releases one instance of the write lock on the r/w semaphore.

Parameters

s	The r/w semaphore to release the write lock on.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

EPERM - the write lock is not currently held by the calling thread
EINVAL - the semaphore is not initialized

9.79 rwsem.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    include/kos/rwsem.h
00004    Copyright (C) 2008, 2010, 2012 Lawrence Sebald
00005
00006 */
```

```
00007
00008 /** \file   kos/rwsem.h
00009     \brief  Definition for a reader/writer semaphore.
00010     \ingroup kthreads
00011
00012     This file defines a concept of reader/writer semaphores. Basically, this
00013     type of lock allows an unlimited number of "readers" to acquire the lock at
00014     a time, but only one "writer" (and only if no readers hold the lock).
00015     Readers, by definition, should not change any global data (since they are
00016     defined to only be reading), and since this is the case it is safe to allow
00017     multiple readers to access global data that is shared amongst threads.
00018     Writers on the other hand require exclusive access since they will be
00019     changing global data in the critical section, and they cannot share with
00020     a reader either (since the reader might attempt to read while the writer is
00021     changing data).
00022
00023     \author Lawrence Sebald
00024 */
00025
00026 #ifndef __KOS_RWSEM_H
00027 #define __KOS_RWSEM_H
00028
00029 #include <kos/cdefs.h>
00030
00031 __BEGIN_DECLS
00032
00033 #include <stddef.h>
00034 #include <kos/thread.h>
00035
00036 /** \brief  Reader/writer semaphore structure.
00037
00038     All members of this structure should be considered to be private, it is not
00039     safe to change anything in here yourself.
00040
00041     \headerfile kos/rwsem.h
00042 */
00043 typedef struct rw_semaphore {
00044     /** \brief  Was this structure created with rwsem_create()? */
00045     int dynamic;
00046
00047     /** \brief  The number of readers that are currently holding the lock. */
00048     int read_count;
00049
00050     /** \brief  The thread holding the write lock. */
00051     kthread_t *write_lock;
00052
00053     /** \brief  Space for one reader who's trying to upgrade to a writer. */
00054     kthread_t *reader_waiting;
00055 } rw_semaphore_t;
00056
00057 /** \brief  Initializer for a transient reader/writer semaphore */
00058 #define RWSEM_INITIALIZER   { 0, 0, NULL, NULL }
00059
00060 /** \brief  Allocate a reader/writer semaphore.
00061
00062     This function allocates a new reader/writer lock that is initially not
00063     locked either for reading or writing.
00064
00065     \deprecated
00066     This function is formally deprecated, and should not be used in newly
00067     written code. Instead, please use rwsem_init().
00068
00069     \return The created semaphore, or NULL on failure (errno will be set as
00070             appropriate).
00071
00072     \par    Error Conditions:
00073     \em     ENOMEM - out of memory
00074 */
00075 rw_semaphore_t *rwsem_create(void) __depr("Use rwsem_init or RWSEM_INITIALIZER.");
00076
00077 /** \brief  Initialize a reader/writer semaphore.
00078
00079     This function initializes a new reader/writer semaphore for use.
00080
00081     \retval 0       On success (no error conditions currently defined).
00082 */
00083 int rwsem_init(rw_semaphore_t *s);
00084
00085 /** \brief  Destroy a reader/writer semaphore.
00086
00087     This function cleans up a reader/writer semaphore. It is an error to attempt
```

```
00088     to destroy a r/w semaphore that is locked either for reading or writing.
00089
00090     \param  s       The r/w semaphore to destroy.
00091     \retval 0       On success.
00092     \retval -1      On error, errno will be set as appropriate.
00093
00094     \par    Error Conditions:
00095     \em     EBUSY - the semaphore is still locked
00096 */
00097 int rwsem_destroy(rw_semaphore_t *s);
00098
00099 /** \brief  Lock a reader/writer semaphore for reading (with a timeout).
00100
00101     This function attempts to lock the r/w semaphore for reading. If the
00102     semaphore is locked for writing, this function will block until it is
00103     possible to obtain the lock for reading or the timeout expires. This
00104     function is <b>NOT</b> safe to call inside of an interrupt.
00105
00106     \param  s       The r/w semaphore to lock.
00107     \param  timeout The maximum time to wait (in milliseconds).
00108     \retval 0       On success
00109     \retval -1      On error, errno will be set as appropriate.
00110
00111     \par    Error Conditions:
00112     \em     EPERM - called inside an interrupt \n
00113     \em     ETIMEDOUT - the timeout expires before the lock can be acquired \n
00114     \em     EINVAL - the timeout value is invalid \n
00115     \em     EINVAL - the semaphore is not initialized
00116 */
00117 int rwsem_read_lock_timed(rw_semaphore_t *s, int timeout);
00118
00119 /** \brief  Lock a reader/writer semaphore for reading.
00120
00121     This function attempts to lock the r/w semaphore for reading. If the
00122     semaphore is locked for writing, this function will block until it is
00123     possible to obtain the lock for reading. This function is <b>NOT</b> safe to
00124     call inside of an interrupt.
00125
00126     \param  s       The r/w semaphore to lock.
00127     \retval 0       On success
00128     \retval -1      On error, errno will be set as appropriate.
00129
00130     \par    Error Conditions:
00131     \em     EPERM - called inside an interrupt \n
00132     \em     EINVAL - the semaphore is not initialized
00133 */
00134 int rwsem_read_lock(rw_semaphore_t *s);
00135
00136 /** \brief  Lock a reader/writer semaphore for writing (with a timeout).
00137
00138     This function attempts to lock the r/w semaphore for writing. If the
00139     semaphore is locked for reading or writing, this function will block until
00140     it is possible to obtain the lock for writing or the timeout expires. This
00141     function is <b>NOT</b> safe to call inside of an interrupt.
00142
00143     \param  s       The r/w semaphore to lock.
00144     \param  timeout The maximum time to wait (in milliseconds).
00145     \retval 0       On success.
00146     \retval -1      On error, errno will be set as appropriate.
00147
00148     \par    Error Conditions:
00149     \em     EPERM - called inside an interrupt \n
00150     \em     ETIMEDOUT - the timeout expires before the lock can be acquired \n
00151     \em     EINVAL - the timeout value is invalid \n
00152     \em     EINVAL - the semaphore is not initialized
00153 */
00154 int rwsem_write_lock_timed(rw_semaphore_t *s, int timeout);
00155
00156 /** \brief  Lock a reader/writer semaphore for writing.
00157
00158     This function attempts to lock the r/w semaphore for writing. If the
00159     semaphore is locked for reading or writing, this function will block until
00160     it is possible to obtain the lock for writing. This function is <b>NOT</b>
00161     safe to call inside of an interrupt.
00162
00163     \param  s       The r/w semaphore to lock.
00164     \retval 0       On success.
00165     \retval -1      On error, errno will be set as appropriate.
00166
00167     \par    Error conditions:
00168     \em     EPERM - called inside an interrupt \n
```

```
00169     \em     EINVAL - the semaphore is not initialized
00170 */
00171 int rwsem_write_lock(rw_semaphore_t *s);
00172
00173 /** \brief  Unlock a reader/writer semaphore from a read lock.
00174
00175     This function releases one instance of the read lock on the r/w semaphore.
00176
00177     \param  s       The r/w semaphore to release the read lock on.
00178     \retval 0       On success.
00179     \retval -1      On error, errno will be set as appropriate.
00180
00181     \par    Error Conditions:
00182     \em     EPERM - the read lock is not currently held \n
00183     \em     EINVAL - the semaphore is not initialized
00184 */
00185 int rwsem_read_unlock(rw_semaphore_t *s);
00186
00187 /** \brief  Unlock a reader/writer semaphore from a write lock.
00188
00189     This function releases one instance of the write lock on the r/w semaphore.
00190
00191     \param  s       The r/w semaphore to release the write lock on.
00192     \retval 0       On success.
00193     \retval -1      On error, errno will be set as appropriate.
00194
00195     \par    Error Conditions:
00196     \em     EPERM - the write lock is not currently held by the calling
00197                     thread \n
00198     \em     EINVAL - the semaphore is not initialized
00199 */
00200 int rwsem_write_unlock(rw_semaphore_t *s);
00201
00202 /** \brief  Unlock a reader/writer semaphore.
00203
00204     This function releases the lock held by the current thread on the specified
00205     reader/writer semaphore. This function will automatically determine which
00206     lock is held by the calling thread and release it as appropriate.
00207
00208     This function is <b>NOT</b> safe to call (in general) if you do not hold the
00209     lock!
00210
00211     \param  s       The r/w semaphore to release the lock on.
00212     \retval 0       On success.
00213     \retval -1      On error, errno will be set as appropriate.
00214
00215     \par    Error Conditions:
00216     \em     EPERM - the lock is not currently held by the calling thread \n
00217     \em     EINVAL - the semaphore is not initialized
00218 */
00219 int rwsem_unlock(rw_semaphore_t *s);
00220
00221 /** \brief  Attempt to lock a reader/writer semaphore for reading.
00222
00223     This function attempts to lock the r/w semaphore for reading. If for any
00224     reason rwsem_read_lock would normally block, this function will return an
00225     error. This function is safe to call inside an interrupt.
00226
00227     \param  s       The r/w semaphore to attempt to lock.
00228     \retval 0       On success.
00229     \retval -1      On error, errno will be set as appropriate.
00230
00231     \par    Error Conditions:
00232     \em     EWOULDBLOCK - a call to rwsem_read_lock would block \n
00233     \em     EINVAL - the semaphore is not initialized
00234 */
00235 int rwsem_read_trylock(rw_semaphore_t *s);
00236
00237 /** \brief  Attempt to lock a reader/writer semaphore for writing.
00238
00239     This function attempts to lock the r/w semaphore for writing. If for any
00240     reason rwsem_write_lock would normally block, this function will return an
00241     error. This function is safe to call inside an interrupt.
00242
00243     \param  s       The r/w semaphore to attempt to lock.
00244     \retval 0       On success.
00245     \retval -1      On error, errno will be set as appropriate.
00246
00247     \par    Error Conditions:
00248     \em     EWOULDBLOCK - a call to rwsem_write_lock would block \n
00249     \em     EINVAL - the semaphore is not initialized
```

```
00250 */
00251 int rwsem_write_trylock(rw_semaphore_t *s);
00252
00253 /** \brief  Upgrade a thread from reader status to writer status (with a
00254             timeout).
00255
00256     This function will upgrade the lock on the calling thread from a reader
00257     state to a writer state. If it cannot do this at the moment, it will block
00258     until it is possible. This function is <b>NOT</b> safe to call inside an
00259     interrupt.
00260
00261     You can only have one reader waiting to upgrade at a time, otherwise the
00262     state would potentially become corrupted between when this is called and
00263     when you get the lock. If you get -1 back from this, you must not assume
00264     that you can write safely! On error, the calling thread will still hold a
00265     read lock.
00266
00267     \param  s       The r/w semaphore to upgrade.
00268     \param  timeout The maximum time to wait (in milliseconds).
00269     \retval 0       On success.
00270     \retval -1      On error, errno will be set as appropriate.
00271
00272     \par    Error Conditions:
00273     \em     EPERM - called inside an interrupt \n
00274     \em     EINVAL - the semaphore is not initialized \n
00275     \em     EINVAL - the timeout value is invalid \n
00276     \em     EBUSY - another reader has already requested an upgrade \n
00277     \em     ETIMEDOUT - the timeout expired before the write lock could be
00278                         acquired
00279 */
00280 int rwsem_read_upgrade_timed(rw_semaphore_t *s, int timeout);
00281
00282 /** \brief  Upgrade a thread from reader status to writer status.
00283
00284     This function will upgrade the lock on the calling thread from a reader
00285     state to a writer state. If it cannot do this at the moment, it will block
00286     until it is possible. This function is <b>NOT</b> safe to call inside an
00287     interrupt.
00288
00289     You can only have one reader waiting to upgrade at a time, otherwise the
00290     state would potentially become corrupted between when this is called and
00291     when you get the lock. If you get -1 back from this, you must not assume
00292     that you can write safely! On error, the calling thread will still hold a
00293     read lock.
00294
00295     \param  s       The r/w semaphore to upgrade.
00296     \retval 0       On success.
00297     \retval -1      On error, errno will be set as appropriate.
00298
00299     \par    Error Conditions:
00300     \em     EPERM - called inside an interrupt \n
00301     \em     EINVAL - the semaphore is not initialized \n
00302     \em     EBUSY - another reader has already requested an upgrade
00303 */
00304 int rwsem_read_upgrade(rw_semaphore_t *s);
00305
00306 /** \brief  Attempt to upgrade a thread from reader status to writer status.
00307
00308     This function will attempt to upgrade the lock on the calling thread to
00309     writer status. If for any reason rwsem_read_upgrade would block, this
00310     function will return an error. This function is safe to call inside an
00311     interrupt. Note that on error, the read lock is still held!
00312
00313     \param  s       The r/w semaphore to upgrade.
00314     \retval 0       On success.
00315     \retval -1      On error, errno will be set as appropriate.
00316
00317     \par    Error Conditions:
00318     \em     EWOULDBLOCK - a call to rwsem_read_upgrade would block \n
00319     \em     EBUSY - another reader has already requested an upgrade \n
00320     \em     EINVAL - the sempahore is not initialized
00321 */
00322 int rwsem_read_tryupgrade(rw_semaphore_t *s);
00323
00324 /** \brief  Read the reader count on the reader/writer semaphore.
00325
00326     This function is not a safe way to see if the lock will be locked by any
00327     readers when you get around to locking it, so do not use it in this way.
00328
00329     \param  s       The r/w semaphore to count the readers on.
00330     \return The number of readers holding the r/w semaphore.
```

```
00331 */
00332 int rwsem_read_count(rw_semaphore_t *s);
00333
00334 /** \brief  Read the state of the writer lock on the reader/writer semaphore.
00335
00336     This function is not a safe way to see if the lock will be locked by a
00337     writer by the time you get around to doing something with it, so don't try
00338     to use it for that purpose.
00339
00340     \param  s       The r/w semaphore to check the writer status on.
00341     \return The status of the writer lock of the r/w semaphore.
00342 */
00343 int rwsem_write_locked(rw_semaphore_t *s);
00344
00345 __END_DECLS
00346
00347 #endif /* __KOS_RWSEM_H */
```

9.80 include/kos/sem.h File Reference

Semaphores.

```
#include <kos/cdefs.h>
```

Data Structures

- struct semaphore_t

  Semaphore type.

Macros

- #define SEM_INITIALIZER(value) { 1, value }

  Initializer for a transient semaphore.

Functions

- semaphore_t * sem_create (int value) __depr("Use sem_init or SEM_INITIALIZER.")

  Allocate a new semaphore.
- int sem_init (semaphore_t *sm, int count)

  Initialize a semaphore for use.
- int sem_destroy (semaphore_t *sem)

  Destroy a semaphore.
- int sem_wait (semaphore_t *sem)

  Wait on a semaphore.
- int sem_wait_timed (semaphore_t *sem, int timeout)

  Wait on a semaphore (with a timeout).
- int sem_trywait (semaphore_t *sem)

  "Wait" on a semaphore without blocking.
- int sem_signal (semaphore_t *sem)

  Signal a semaphore.
- int sem_count (semaphore_t *sem)

  Retrieve the number of available resources.

9.80.1 Detailed Description

Semaphores.

This file defines semaphores. A semaphore is a synchronization primitive that allows a spcified number of threads to be in its critical section at a single point of time. Another way to think of it is that you have a predetermined number of resources available, and the semaphore maintains the resources.

Author

Megan Potter

See also

kos/mutex.h

9.80.2 Macro Definition Documentation

SEM_INITIALIZER

```
#define SEM_INITIALIZER(
              value ) { 1, value }
```

Initializer for a transient semaphore.

Parameters

value	The initial count of the semaphore.

9.80.3 Function Documentation

sem_count()

```
int sem_count (
             semaphore_t * sem )
```

Retrieve the number of available resources.

This function will retrieve the count of available resources for a semaphore. This is not a thread-safe way to make sure resources will be available when you get around to waiting, so don't use it as such.

Parameters

sem	The semaphore to check

Returns

The count of the semaphore (the number of resources currently available)

sem_create()

```
semaphore_t * sem_create (
            int value )
```

Allocate a new semaphore.

This function allocates and initializes a new semaphore for use.

Deprecated This function is formally deprecated. Please update your code to use sem_init() or static initialization with SEM_INITIALIZER instead.

Parameters

value	The initial count of the semaphore (the number of threads to allow in the critical section at a time)

Returns

The created semaphore on success. NULL is returned on failure and errno is set as appropriate.

Error Conditions:

ENOMEM - out of memory
EINVAL - the semaphore's value is invalid (less than 0)

sem_destroy()

```
int sem_destroy (
            semaphore_t * sem )
```

Destroy a semaphore.

This function frees a semaphore, releasing any memory associated with it. If there are any threads currently waiting on the semaphore, they will be woken with an ENOTRECOVERABLE error.

Parameters

sem	The semaphore to destroy

Return values

0	On success (no error conditions currently defined)

sem_init()

```
int sem_init (
            semaphore_t * sm,
            int count )
```

Initialize a semaphore for use.

This function initializes the semaphore passed in with the starting count value specified.

Parameters

sm	The semaphore to initialize
count	The initial count of the semaphore

Return values

0	On success
-1	On error, errno will be set as appropriate

Error Conditions:

EINVAL - the semaphore's value is invalid (less than 0)

sem_signal()

```
int sem_signal (
            semaphore_t * sem )
```

Signal a semaphore.

This function will release resources associated with a semaphore, signalling a waiting thread to continue on, if any are waiting. It is your responsibility to make sure you only release resources you have.

Parameters

sem	The semaphore to signal

Return values

0	On success
-1	On error, sets errno as appropriate

Error Conditions:

EINVAL - the semaphore was not initialized

sem_trywait()

```
int sem_trywait (
            semaphore_t * sem )
```

"Wait" on a semaphore without blocking.

This function will decrement the semaphore's count and return, if resources are available. Otherwise, it will return an error.

This function does not protect you against doing things that will cause a deadlock. This function, unlike the other waiting functions is safe to call inside an interrupt.

Parameters

sem	The semaphore to "wait" on

Return values

0	On success
-1	On error, sets errno as appropriate

Error Conditions:

EWOULDBLOCK - a call to sem_wait() would block
EINVAL - the semaphore was not initialized

sem_wait()

```
int sem_wait (
            semaphore_t * sem )
```

Wait on a semaphore.

This function will decrement the semaphore's count and return, if resources are available. Otherwise, the function will block until the resources become available.

This function does not protect you against doing things that will cause a deadlock. This function is not safe to call in an interrupt. See sem_trywait() for a safe function to call in an interrupt.

Parameters

sem	The semaphore to wait on

Return values

0	On success
-1	On error, sets errno as appropriate

Error Conditions:

EPERM - called inside an interrupt
EINVAL - the semaphore was not initialized

sem_wait_timed()

```
int sem_wait_timed (
            semaphore_t * sem,
            int timeout )
```

Wait on a semaphore (with a timeout).

This function will decrement the semaphore's count and return, if resources are available. Otherwise, the function will block until the resources become available or the timeout expires.

This function does not protect you against doing things that will cause a deadlock. This function is not safe to call in an interrupt. See sem_trywait() for a safe function to call in an interrupt.

Parameters

sem	The semaphore to wait on
timeout	The maximum number of milliseconds to block (a value of 0 here will block indefinitely)

Return values

0	On success
-1	On error, sets errno as appropriate

Error Conditions:

EPERM - called inside an interrupt
EINVAL - the semaphore was not initialized
EINVAL - the timeout value was invalid (less than 0)
ETIMEDOUT - timed out while blocking

9.81 sem.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    include/kos/sem.h
00004    Copyright (C) 2001, 2003 Megan Potter
00005    Copyright (C) 2012 Lawrence Sebald
00006
00007 */
00008
00009 /** \file   kos/sem.h
00010     \brief  Semaphores.
00011     \ingroup kthreads
00012
00013     This file defines semaphores. A semaphore is a synchronization primitive
00014     that allows a spcified number of threads to be in its critical section at a
00015     single point of time. Another way to think of it is that you have a
00016     predetermined number of resources available, and the semaphore maintains the
00017     resources.
00018
00019     \author Megan Potter
00020     \see    kos/mutex.h
00021 */
00022
00023 #ifndef __KOS_SEM_H
00024 #define __KOS_SEM_H
00025
00026 #include <kos/cdefs.h>
00027
00028 __BEGIN_DECLS
00029
00030 /** \brief  Semaphore type.
00031
00032     This structure defines a semaphore. There are no public members of this
00033     structure for you to actually do anything with in your code, so don't try.
00034
00035     \headerfile kos/sem.h
00036 */
00037 typedef struct semaphore {
00038     int initialized;    /**< \brief Are we initialized? */
00039     int count;          /**< \brief The semaphore count */
00040 } semaphore_t;
00041
00042 /** \brief  Initializer for a transient semaphore.
00043     \param  value           The initial count of the semaphore. */
00044 #define SEM_INITIALIZER(value) { 1, value }
00045
00046 /** \brief  Allocate a new semaphore.
00047
00048     This function allocates and initializes a new semaphore for use.
00049
00050     \deprecated
00051     This function is formally deprecated. Please update your code to use
00052     sem_init() or static initialization with SEM_INITIALIZER instead.
00053
00054     \param  value           The initial count of the semaphore (the number of
00055                             threads to allow in the critical section at a time)
00056
00057     \return                 The created semaphore on success. NULL is returned
00058                             on failure and errno is set as appropriate.
00059
00060     \par    Error Conditions:
00061     \em     ENOMEM - out of memory \n
00062     \em     EINVAL - the semaphore's value is invalid (less than 0)
00063 */
00064 semaphore_t *sem_create(int value) __depr("Use sem_init or SEM_INITIALIZER.");
00065
00066 /** \brief  Initialize a semaphore for use.
00067
00068     This function initializes the semaphore passed in with the starting count
00069     value specified.
00070
00071     \param  sm              The semaphore to initialize
00072     \param  count           The initial count of the semaphore
00073     \retval 0               On success
00074     \retval -1              On error, errno will be set as appropriate
00075
00076     \par    Error Conditions:
```

```
00077     \em     EINVAL - the semaphore's value is invalid (less than 0)
00078 */
00079 int sem_init(semaphore_t *sm, int count);
00080
00081 /** \brief  Destroy a semaphore.
00082
00083     This function frees a semaphore, releasing any memory associated with it. If
00084     there are any threads currently waiting on the semaphore, they will be woken
00085     with an ENOTRECOVERABLE error.
00086
00087     \param  sem             The semaphore to destroy
00088     \retval 0               On success (no error conditions currently defined)
00089 */
00090 int sem_destroy(semaphore_t *sem);
00091
00092 /** \brief  Wait on a semaphore.
00093
00094     This function will decrement the semaphore's count and return, if resources
00095     are available. Otherwise, the function will block until the resources become
00096     available.
00097
00098     This function does not protect you against doing things that will cause a
00099     deadlock. This function is not safe to call in an interrupt. See
00100     sem_trywait() for a safe function to call in an interrupt.
00101
00102     \param  sem             The semaphore to wait on
00103     \retval 0               On success
00104     \retval -1              On error, sets errno as appropriate
00105
00106     \par    Error Conditions:
00107     \em     EPERM - called inside an interrupt \n
00108     \em     EINVAL - the semaphore was not initialized
00109 */
00110 int sem_wait(semaphore_t *sem);
00111
00112 /** \brief  Wait on a semaphore (with a timeout).
00113
00114     This function will decrement the semaphore's count and return, if resources
00115     are available. Otherwise, the function will block until the resources become
00116     available or the timeout expires.
00117
00118     This function does not protect you against doing things that will cause a
00119     deadlock. This function is not safe to call in an interrupt. See
00120     sem_trywait() for a safe function to call in an interrupt.
00121
00122     \param  sem             The semaphore to wait on
00123     \param  timeout         The maximum number of milliseconds to block (a value
00124                             of 0 here will block indefinitely)
00125     \retval 0               On success
00126     \retval -1              On error, sets errno as appropriate
00127
00128     \par    Error Conditions:
00129     \em     EPERM - called inside an interrupt \n
00130     \em     EINVAL - the semaphore was not initialized \n
00131     \em     EINVAL - the timeout value was invalid (less than 0) \n
00132     \em     ETIMEDOUT - timed out while blocking
00133  */
00134 int sem_wait_timed(semaphore_t *sem, int timeout);
00135
00136 /** \brief  "Wait" on a semaphore without blocking.
00137
00138     This function will decrement the semaphore's count and return, if resources
00139     are available. Otherwise, it will return an error.
00140
00141     This function does not protect you against doing things that will cause a
00142     deadlock. This function, unlike the other waiting functions is safe to call
00143     inside an interrupt.
00144
00145     \param  sem             The semaphore to "wait" on
00146     \retval 0               On success
00147     \retval -1              On error, sets errno as appropriate
00148
00149     \par    Error Conditions:
00150     \em     EWOULDBLOCK - a call to sem_wait() would block \n
00151     \em     EINVAL - the semaphore was not initialized
00152 */
00153 int sem_trywait(semaphore_t *sem);
00154
00155 /** \brief  Signal a semaphore.
00156
00157     This function will release resources associated with a semaphore, signalling
```

```
00158     a waiting thread to continue on, if any are waiting. It is your
00159     responsibility to make sure you only release resources you have.
00160
00161     \param  sem             The semaphore to signal
00162     \retval 0               On success
00163     \retval -1              On error, sets errno as appropriate
00164
00165     \par    Error Conditions:
00166     \em     EINVAL - the semaphore was not initialized
00167 */
00168 int sem_signal(semaphore_t *sem);
00169
00170 /** \brief  Retrieve the number of available resources.
00171
00172     This function will retrieve the count of available resources for a
00173     semaphore. This is not a thread-safe way to make sure resources will be
00174     available when you get around to waiting, so don't use it as such.
00175
00176     \param  sem             The semaphore to check
00177     \return                 The count of the semaphore (the number of resources
00178                             currently available)
00179 */
00180 int sem_count(semaphore_t *sem);
00181
00182 __END_DECLS
00183
00184 #endif  /* __KOS_SEM_H */
```

9.82 include/kos/stdlib.h File Reference

```
#include <kos/cdefs.h>
```

Functions

- int posix_memalign (void **memptr, size_t alignment, size_t size)

9.82.1 Function Documentation

posix_memalign()

```
int posix_memalign (
             void ** memptr,
             size_t alignment,
             size_t size )  [extern]
```

9.83 stdlib.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/stdlib.h
00004    Copyright (C) 2023 Falco Girgis
00005 */
00006
00007 /*
00008    Add select POSIX extensions to stdlib.h which are not present within Newlib.
00009 */
00010
```

```
00011 #ifndef _STDLIB_H_
00012 #error "Do not include this file directly. Use <stdlib.h> instead."
00013 #endif /* !_STDLIB_H_ */
00014
00015 #ifndef __KOS_STDLIB_H
00016 #define __KOS_STDLIB_H
00017
00018 #if !defined(__STRICT_ANSI__) || (_POSIX_C_SOURCE >= 200112L) || \
00019     (_XOPEN_SOURCE >= 600)
00020
00021 #include <kos/cdefs.h>
00022
00023 __BEGIN_DECLS
00024
00025 extern int posix_memalign(void **memptr, size_t alignment, size_t size);
00026
00027 __END_DECLS
00028
00029 #endif /* !defined(__STRICT_ANSI__) || (_POSIX_C_SOURCE >= 200112L) ||
00030           (_XOPEN_SOURCE >= 600) */
00031 #endif /* !__KOS_STDLIB_H */
```

9.84 include/kos/string.h File Reference

Variants on standard block memory copy/set functions.

```
#include <sys/cdefs.h>
#include <string.h>
```

Functions

- void * memcpy4 (void *dest, const void *src, size_t count)

  Copy a block of memory, 4 bytes at a time.
- void * memset4 (void *s, unsigned long c, size_t count)

  Set a block of memory, 4 bytes at a time.
- void * memcpy2 (void *dest, const void *src, size_t count)

  Copy a block of memory, 2 bytes at a time.
- void * memset2 (void *s, unsigned short c, size_t count)

  Set a block of memory, 2 bytes at a time.

9.84.1 Detailed Description

Variants on standard block memory copy/set functions.

This file contains variants on the standard block memory copy/set functions. These variants copy/set memory in the specified block sizes, which may be helpful for interacting with memory-mapped hardware.

Author

Megan Potter

9.84.2 Function Documentation

memcpy2()

```
void * memcpy2 (
            void * dest,
            const void * src,
            size_t count )
```

Copy a block of memory, 2 bytes at a time.

This function is identical to memcpy(), except it copies 2 bytes at a time.

Parameters

dest	The destination of the copy.
src	The source to copy.
count	The number of bytes to copy. This should be divisible by 2 (and will be rounded down if not).

Returns

The original value of dest.

memcpy4()

```
void * memcpy4 (
            void * dest,
            const void * src,
            size_t count )
```

Copy a block of memory, 4 bytes at a time.

This function is identical to memcpy(), except it copies 4 bytes at a time.

Parameters

dest	The destination of the copy.
src	The source to copy.
count	The number of bytes to copy. This should be divisible by 4 (and will be rounded down if not).

Returns

The original value of dest.

memset2()

```
void * memset2 (
            void * s,
            unsigned short c,
            size_t count )
```

Set a block of memory, 2 bytes at a time.

This function is identical to memset(), except it sets 2 bytes at a time. This implies that all 16-bits of c are used, not just the first 8 as in memset().

Parameters

s	The destination of the set.
c	The value to set to.
count	The number of bytes to set. This should be divisible by 2 (and will be rounded down if not).

Returns

The original value of dest.

memset4()

```
void * memset4 (
            void * s,
            unsigned long c,
            size_t count )
```

Set a block of memory, 4 bytes at a time.

This function is identical to memset(), except it sets 4 bytes at a time. This implies that all 32-bits of c are used, not just the first 8 as in memset().

Parameters

s	The destination of the set.
c	The value to set to.
count	The number of bytes to set. This should be divisible by 4 (and will be rounded down if not).

Returns

The original value of dest.

9.85 string.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/string.h
00004    Copyright (C)2004 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/string.h
00009     \brief  Variants on standard block memory copy/set functions.
00010
00011     This file contains variants on the standard block memory copy/set functions.
00012     These variants copy/set memory in the specified block sizes, which may be
00013     helpful for interacting with memory-mapped hardware.
00014
00015     \author Megan Potter
00016 */
00017
00018 #ifndef __KOS_STRING_H
00019 #define __KOS_STRING_H
00020
00021 #include <sys/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 #include <string.h>
00025
00026 /** \brief  Copy a block of memory, 4 bytes at a time.
00027
00028     This function is identical to memcpy(), except it copies 4 bytes at a time.
00029
00030     \param  dest            The destination of the copy.
00031     \param  src             The source to copy.
00032     \param  count           The number of bytes to copy. This should be
00033                             divisible by 4 (and will be rounded down if not).
00034     \return                 The original value of dest.
00035 */
00036 void * memcpy4(void * dest, const void *src, size_t count);
00037
00038 /** \brief  Set a block of memory, 4 bytes at a time.
00039
00040     This function is identical to memset(), except it sets 4 bytes at a time.
00041     This implies that all 32-bits of c are used, not just the first 8 as in
00042     memset().
00043
00044     \param  s               The destination of the set.
00045     \param  c               The value to set to.
00046     \param  count           The number of bytes to set. This should be
00047                             divisible by 4 (and will be rounded down if not).
00048     \return                 The original value of dest.
00049 */
00050 void * memset4(void * s, unsigned long c, size_t count);
00051
00052 /** \brief  Copy a block of memory, 2 bytes at a time.
00053
00054     This function is identical to memcpy(), except it copies 2 bytes at a time.
00055
00056     \param  dest            The destination of the copy.
00057     \param  src             The source to copy.
00058     \param  count           The number of bytes to copy. This should be
00059                             divisible by 2 (and will be rounded down if not).
00060     \return                 The original value of dest.
00061 */
00062 void * memcpy2(void * dest, const void *src, size_t count);
00063
00064 /** \brief  Set a block of memory, 2 bytes at a time.
00065
00066     This function is identical to memset(), except it sets 2 bytes at a time.
00067     This implies that all 16-bits of c are used, not just the first 8 as in
00068     memset().
00069
00070     \param  s               The destination of the set.
00071     \param  c               The value to set to.
00072     \param  count           The number of bytes to set. This should be
00073                             divisible by 2 (and will be rounded down if not).
00074     \return                 The original value of dest.
00075 */
00076 void * memset2(void * s, unsigned short c, size_t count);
00077
00078 __END_DECLS
00079
00080 #endif  /* __KOS_STRING_H */
00081
```

00082

9.86 include/kos/thread.h File Reference

Threading support.

```
#include <sys/cdefs.h>
#include <kos/cdefs.h>
#include <kos/tls.h>
#include <arch/irq.h>
#include <sys/queue.h>
#include <sys/reent.h>
#include <stdint.h>
```

Data Structures

- struct tcbhead_t

  Control Block Header.
- struct kthread_t

  Structure describing one running thread.
- struct kthread_attr_t

  Thread creation attributes.

Macros

- #define PRIO_MAX 4096

  Maximal thread priority.
- #define PRIO_DEFAULT 10

  Default thread priority.
- #define KTHREAD_LABEL_SIZE 256

  Size of a kthread's label.
- #define KTHREAD_PWD_SIZE 256

  Size of a kthread's current directory.

Thread flag values

kthread_t::flags values

These are possible values for the flags field on the kthread_t structure. These can be ORed together.

- #define THD_DEFAULTS 0

  Defaults: no flags.
- #define THD_USER 1

  Thread runs in user mode.
- #define THD_QUEUED 2

  Thread is in the run queue.
- #define THD_DETACHED 4

  Thread is detached.

Thread states

kthread_t::state values

Each thread in the system is in exactly one of this set of states.

- #define STATE_ZOMBIE 0x0000
  Waiting to die.
- #define STATE_RUNNING 0x0001
  Process is "current".
- #define STATE_READY 0x0002
  Ready to be scheduled.
- #define STATE_WAIT 0x0003
  Blocked on a genwait.
- #define STATE_FINISHED 0x0004
  Finished execution.

Threading system modes

kthread mode values

The threading system will always be in one of the following modes. This represents either pre-emptive scheduling or an un-initialized state.

- #define THD_MODE_NONE -1
  Threads not running.
- #define THD_MODE_COOP 0
  Cooperative mode.
- #define THD_MODE_PREEMPT 1
  Preemptive threading mode.

Functions

- int thd_block_now (irq_context_t *mycxt)
  Block the current thread.
- irq_context_t * thd_choose_new (void)
  Find a new thread to swap in.
- kthread_t * thd_by_tid (tid_t tid)
  Given a thread ID, locates the thread structure.
- void thd_add_to_runnable (kthread_t *t, int front_of_line)
  Enqueue a process in the runnable queue.
- int thd_remove_from_runnable (kthread_t *thd)
  Removes a thread from the runnable queue, if it's there.
- kthread_t * thd_create (int detach, void *(*routine)(void *param), void *param)
  Create a new thread.
- kthread_t * thd_create_ex (const kthread_attr_t *__RESTRICT attr, void *(*routine)(void *param), void *param)
  Create a new thread with the specified set of attributes.
- int thd_destroy (kthread_t *thd)
  Brutally kill the given thread.
- void thd_exit (void *rv) __noreturn
  Exit the current thread.
- void thd_schedule (int front_of_line, uint64_t now)
  Force a thread reschedule.

- void thd_schedule_next (kthread_t *thd)

  Force a given thread to the front of the queue.
- void thd_pass (void)

  Throw away the current thread's timeslice.
- void thd_sleep (int ms)

  Sleep for a given number of milliseconds.
- int thd_set_prio (kthread_t *thd, prio_t prio)

  Set a thread's priority value.
- kthread_t * thd_get_current (void)

  Retrieve the current thread's kthread struct.
- const char * thd_get_label (kthread_t *thd)

  Retrieve the thread's label.
- void thd_set_label (kthread_t *thd, const char *__RESTRICT label)

  Set the thread's label.
- const char * thd_get_pwd (kthread_t *thd)

  Retrieve the thread's current working directory.
- void thd_set_pwd (kthread_t *thd, const char *__RESTRICT pwd)

  Set the thread's current working directory.
- int * thd_get_errno (kthread_t *thd)

  Retrieve a pointer to the thread errno.
- struct _reent * thd_get_reent (kthread_t *thd)

  Retrieve a pointer to the thread reent struct.
- int thd_set_mode (int mode) __deprecated

  Change threading modes.
- int thd_get_mode (void) __deprecated

  Fetch the current threading mode.
- int thd_join (kthread_t *thd, void **value_ptr)

  Wait for a thread to exit.
- int thd_detach (kthread_t *thd)

  Detach a joinable thread.
- int thd_each (int(*cb)(kthread_t *thd, void *user_data), void *data)

  Iterate all threads and call the passed callback for each.
- int thd_pslist (int(*pf)(const char *fmt,...))

  Print a list of all threads using the given print function.
- int thd_pslist_queue (int(*pf)(const char *fmt,...))

  Print a list of all queued threads using the given print function.
- int thd_init (void)

  Initialize the threading system.
- void thd_shutdown (void)

  Shutdown the threading system.

9.86.1 Detailed Description

Threading support.

This file contains the interface to the threading system of KOS. Timer interrupts are used to reschedule threads within the system.

Author

Megan Potter

Lawrence Sebald

See also

arch/timer.h

kos/genwait.h

kos/mutex.h

kos/once.h

kos/recursive_lock.h

kos/rwsem.h

kos/sem.h

kos/tls.h

9.86.2 Macro Definition Documentation

KTHREAD_LABEL_SIZE

```
#define KTHREAD_LABEL_SIZE 256
```

Size of a kthread's label.

Maximum number of characters in a thread's label or name (including NULL terminator).

KTHREAD_PWD_SIZE

```
#define KTHREAD_PWD_SIZE 256
```

Size of a kthread's current directory.

Maximum number of characters in a thread's current working directory (including NULL terminator).

PRIO_DEFAULT

```
#define PRIO_DEFAULT 10
```

Default thread priority.

Threads are created by default with the priority specified.

PRIO_MAX

```
#define PRIO_MAX 4096
```

Maximal thread priority.

This macro defines the maximum value for a thread's priority. Note that the larger this number, the lower the priority of the thread.

STATE_FINISHED

```
#define STATE_FINISHED 0x0004
```

Finished execution.

STATE_READY

```
#define STATE_READY 0x0002
```

Ready to be scheduled.

STATE_RUNNING

```
#define STATE_RUNNING 0x0001
```

Process is "current".

STATE_WAIT

```
#define STATE_WAIT 0x0003
```

Blocked on a genwait.

STATE_ZOMBIE

```
#define STATE_ZOMBIE 0x0000
```

Waiting to die.

THD_DEFAULTS

```
#define THD_DEFAULTS 0
```

Defaults: no flags.

THD_DETACHED

```
#define THD_DETACHED 4
```

Thread is detached.

THD_MODE_COOP

```
#define THD_MODE_COOP 0
```

Cooperative mode.

Deprecated

THD_MODE_NONE

```
#define THD_MODE_NONE -1
```

Threads not running.

THD_MODE_PREEMPT

```
#define THD_MODE_PREEMPT 1
```

Preemptive threading mode.

THD_QUEUED

```
#define THD_QUEUED 2
```

Thread is in the run queue.

THD_USER

```
#define THD_USER 1
```

Thread runs in user mode.

9.86.3 Function Documentation

thd_add_to_runnable()

```
void thd_add_to_runnable (
            kthread_t * t,
            int front_of_line )
```

Enqueue a process in the runnable queue.

This function adds a thread to the runnable queue after the process group of the same priority if front_of_line is zero, otherwise queues it at the front of its priority group. Generally, you will not have to do this manually.

Parameters

t	The thread to queue.
front_of_line	Set to 1 to put this thread in front of other threads of the same priority, 0 to put it behind the other threads (normal behavior).

See also

thd_remove_from_runnable

thd_block_now()

```
int thd_block_now (
            irq_context_t * mycxt )
```

Block the current thread.

Blocks the calling thread and performs a reschedule as if a context switch timer had been executed. This is useful for, e.g., blocking on sync primitives. The param 'mycxt' should point to the calling thread's context block. This is implemented in arch-specific code.

The meaningfulness of the return value depends on whether the unblocker set a return value or not.

Parameters

mycxt	The IRQ context of the calling thread.

Returns

Whatever the unblocker deems necessary to return.

thd_by_tid()

```
kthread_t * thd_by_tid (
            tid_t tid )
```

Given a thread ID, locates the thread structure.

Parameters

tid	The thread ID to retrieve.

Returns

The thread on success, NULL on failure.

thd_choose_new()

```
irq_context_t * thd_choose_new (
            void  )
```

Find a new thread to swap in.

This function looks at the state of the system and returns a new thread context to swap in. This is called from thd_block_now() and from the preemptive context switcher. Note that thd_current might be NULL on entering this function, if the caller blocked itself.

It is assumed that by the time this returns, the irq_srt_addr and thd_current will be updated.

Returns

The IRQ context of the thread selected.

thd_create()

```
kthread_t * thd_create (
             int detach,
             void *(*)(void *param) routine,
             void * param )
```

Create a new thread.

This function creates a new kernel thread with default parameters to run the given routine. The thread will terminate and clean up resources when the routine completes if the thread is created detached, otherwise you must join the thread with thd_join() to clean up after it.

Parameters

detach	Set to 1 to create a detached thread. Set to 0 to create a joinable thread.
routine	The function to call in the new thread.
param	A parameter to pass to the function called.

Returns

The new thread on success, NULL on failure.

See also

thd_create_ex, thd_destroy

thd_create_ex()

```
kthread_t * thd_create_ex (
            const kthread_attr_t *__RESTRICT attr,
            void *(*)(void *param) routine,
            void * param )
```

Create a new thread with the specified set of attributes.

This function creates a new kernel thread with the specified set of parameters to run the given routine.

Parameters

attr	A set of thread attributes for the created thread. Passing NULL will initialize all attributes to their default values.
routine	The function to call in the new thread.
param	A parameter to pass to the function called.

Returns

The new thread on success, NULL on failure.

See also

thd_create, thd_destroy

thd_destroy()

```
int thd_destroy (
            kthread_t * thd )
```

Brutally kill the given thread.

This function kills the given thread, removing it from the execution chain, cleaning up thread-local data and other internal structures. In general, you shouldn't call this function at all.

Warning

You should never call this function on the current thread.

Parameters

thd	The thread to destroy.

Return values

0	On success.

See also

thd_create

thd_detach()

```
int thd_detach (
              kthread_t * thd )
```

Detach a joinable thread.

This function switches the specified thread's mode from THD_MODE_JOINABLE to THD_MODE_DETACHED. This will ensure that the thread cleans up all of its internal resources when it exits.

Parameters

thd	The joinable thread to detach.

Returns

0 on success or less than 0 if the thread is non-existant or already detached.

See also

thd_join()

thd_each()

```
int thd_each (
              int(*)(kthread_t *thd, void *user_data) cb,
              void * data )
```

Iterate all threads and call the passed callback for each.

Parameters

cb	The callback to call for each thread. If a nonzero value is returned, iteration ceases immediately.
data	User data to be passed to the callback

Return values

0	or the first nonzero value returned by `cb`.

See also

thd_pslist

thd_exit()

```
void thd_exit (
            void * rv )
```

Exit the current thread.

This function ends the execution of the current thread, removing it from all execution queues. This function will never return to the thread. Returning from the thread's function is equivalent to calling this function.

Parameters

rv	The return value of the thread.

thd_get_current()

```
kthread_t * thd_get_current (
            void  )
```

Retrieve the current thread's kthread struct.

Returns

The current thread's structure.

thd_get_errno()

```
int * thd_get_errno (
            kthread_t * thd )
```

Retrieve a pointer to the thread errno.

This function retrieves a pointer to the errno value for the thread. You should generally just use the errno variable to access this.

Parameters

thd	The thread to retrieve from.

Returns

A pointer to the thread's errno.

thd_get_label()

```
const char * thd_get_label (
             kthread_t * thd )
```

Retrieve the thread's label.

Parameters

thd	The thread to retrieve from.

Returns

The human-readable label of the thread.

See also

thd_set_label

thd_get_mode()

```
int thd_get_mode (
             void  )
```

Fetch the current threading mode.

With preemptive threading being the only mode.

Deprecated This is now deprecated.

Returns

The current mode of the threading system.

See also

thd_set_mode

thd_get_pwd()

```
const char * thd_get_pwd (
            kthread_t * thd )
```

Retrieve the thread's current working directory.

This function retrieves the working directory of a thread. Generally, you will want to use either fs_getwd() or one of the standard C functions for doing this, but this is here in case you need it when the thread isn't active for some reason.

Parameters

thd	The thread to retrieve from.

Returns

The thread's working directory.

See also

thd_set_pd

thd_get_reent()

```
struct _reent * thd_get_reent (
            kthread_t * thd )
```

Retrieve a pointer to the thread reent struct.

This function is used to retrieve some internal state that is used by newlib to provide a reentrant libc.

Parameters

thd	The thread to retrieve from.

Returns

The thread's reent struct.

thd_init()

```
int thd_init (
            void  )
```

Initialize the threading system.

This is normally done for you by default when KOS starts. This will also initialize all the various synchronization primitives.

Return values

-1	If threads are already initialized.
0	On success.

See also

thd_shutdown

thd_join()

```
int thd_join (
             kthread_t * thd,
             void ** value_ptr )
```

Wait for a thread to exit.

This function "joins" a joinable thread. This means effectively that the calling thread blocks until the speified thread completes execution. It is invalid to join a detached thread, only joinable threads may be joined.

Parameters

thd	The joinable thread to join.
value_ptr	A pointer to storage for the thread's return value, or NULL if you don't care about it.

Returns

0 on success, or less than 0 if the thread is non-existant or not joinable.

See also

thd_detach

thd_pass()

```
void thd_pass (
             void  )
```

Throw away the current thread's timeslice.

This function manually yields the current thread's timeslice to the system, forcing a reschedule to occur.

thd_pslist()

```
int thd_pslist (
             int(*)(const char *fmt,...)  pf )
```

Print a list of all threads using the given print function.

Parameters

pf	The printf-like function to print with.

Return values

0	On success.

See also

thd_pslist_queue

thd_pslist_queue()

```
int thd_pslist_queue (
            int(*)(const char *fmt,...)  pf )
```

Print a list of all queued threads using the given print function.

Parameters

pf	The printf-like function to print with.

Return values

0	On success.

See also

thd_pslist

thd_remove_from_runnable()

```
int thd_remove_from_runnable (
            kthread_t * thd )
```

Removes a thread from the runnable queue, if it's there.

This function removes a thread from the runnable queue, if it is currently in that queue. Generally, you shouldn't have to do this manually, as waiting on synchronization primitives and the like will do this for you if needed.

Parameters

thd	The thread to remove from the runnable queue.

Return values

0	On success, or if the thread isn't runnable.

See also

thd_add_to_runnable

thd_schedule()

```
void thd_schedule (
            int front_of_line,
            uint64_t now )
```

Force a thread reschedule.

This function is the thread scheduler, and is generally called from a timer interrupt. You will most likely never have a reason to call this function directly.

For most cases, you'll want to set front_of_line to zero, but read the comments in kernel/thread/thread.c for more info, especially if you need to guarantee low latencies. This function just updates irq_srt_addr and thd_current. Set 'now' to non-zero if you want to use a particular system time for checking timeouts.

Parameters

front_of_line	Set to 0, unless you have a good reason not to.
now	Set to 0, unless you have a good reason not to.

See also

thd_schedule_next

thd_schedule_next()

```
void thd_schedule_next (
            kthread_t * thd )
```

Force a given thread to the front of the queue.

This function promotes the given thread to be the next one that will be swapped in by the scheduler. This function is only callable inside an interrupt context (it simply returns otherwise).

thd_set_label()

```
void thd_set_label (
            kthread_t * thd,
            const char *__RESTRICT label )
```

Set the thread's label.

This function sets the label of a thread, which is simply a human-readable string that is used to identify the thread. These labels aren't used for anything internally, and you can give them any label you want. These are mainly seen in the printouts from thd_pslist() or thd_pslist_queue().

Parameters

thd	The thread to set the label of.
label	The string to set as the label.

See also

thd_get_label

thd_set_mode()

```
int thd_set_mode (
            int mode )
```

Change threading modes.

This function changes the current threading mode of the system. With preemptive threading being the only mode.

Deprecated This is now deprecated.

Parameters

mode	One of the THD_MODE values.

Returns

The old mode of the threading system.

See also

thd_get_mode

thd_set_prio()

```
int thd_set_prio (
             kthread_t * thd,
             prio_t prio )
```

Set a thread's priority value.

This function is used to change the priority value of a thread. If the thread is scheduled already, it will be rescheduled with the new priority value.

Parameters

thd	The thread to change the priority of.
prio	The priority value to assign to the thread.

Return values

0	On success.
-1	thd is NULL.
-2	prio requested was out of range.

thd_set_pwd()

```
void thd_set_pwd (
             kthread_t * thd,
             const char *__RESTRICT pwd )
```

Set the thread's current working directory.

This function will set the working directory of a thread. Generally, you will want to use either fs_chdir() or the standard C chdir() function to do this, but this is here in case you need to do it while the thread isn't active for some reason.

Parameters

thd	The thread to set the working directory of.
pwd	The directory to set as active.

See also

thd_get_pwd

thd_shutdown()

```
void thd_shutdown (
             void  )
```

Shutdown the threading system.

This is done for you by the normal shutdown procedure of KOS. This will also shutdown all the various synchronization primitives.

See also

thd_init

thd_sleep()

```
void thd_sleep (
              int ms )
```

Sleep for a given number of milliseconds.

This function puts the current thread to sleep for the specified amount of time. The thread will be removed from the runnable queue until the given number of milliseconds passes. That is to say that the thread will sleep for at least the given number of milliseconds. If another thread is running, it will likely sleep longer.

Parameters

ms	The number of milliseconds to sleep.

9.87 thread.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    include/kos/thread.h
00004    Copyright (C) 2000, 2001, 2002, 2003 Megan Potter
00005    Copyright (C) 2009, 2010, 2016 Lawrence Sebald
00006
00007 */
00008
00009 #ifndef __KOS_THREAD_H
00010 #define __KOS_THREAD_H
00011
00012 #include <sys/cdefs.h>
00013 __BEGIN_DECLS
00014
00015 #include <kos/cdefs.h>
00016 #include <kos/tls.h>
00017 #include <arch/irq.h>
00018 #include <sys/queue.h>
00019 #include <sys/reent.h>
00020 #include <stdint.h>
00021
00022 #include <stdint.h>
00023
00024 /** \file    kos/thread.h
00025     \brief   Threading support.
00026     \ingroup kthreads
00027
00028     This file contains the interface to the threading system of KOS. Timer
00029     interrupts are used to reschedule threads within the system.
00030
00031     \author Megan Potter
```

```
00032     \author Lawrence Sebald
00033
00034     \see    arch/timer.h
00035     \see    kos/genwait.h
00036     \see    kos/mutex.h
00037     \see    kos/once.h
00038     \see    kos/recursive_lock.h
00039     \see    kos/rwsem.h
00040     \see    kos/sem.h
00041     \see    kos/tls.h
00042 */
00043
00044 /** \defgroup kthreads KThreads
00045     \brief    KOS Native Threading API
00046     \ingroup  threading
00047
00048     The thread scheduler itself is a relatively simplistic priority scheduler.
00049     There is no provision for priorities to erode over time, so keep that in
00050     mind. That practically means that if you have 2 high priority threads that
00051     are always runnable and one low priority thread that is always runnable, the
00052     low priority thread will never actually run (since it will never get to the
00053     front of the run queue because of the high priority threads).
00054
00055     The scheduler supports two distinct types of threads: joinable and detached
00056     threads. A joinable thread is one that can return a value to the creating
00057     thread (or for that matter, any other thread that wishes to join it). A
00058     detached thread is one that is completely detached from the rest of the
00059     system and cannot return values by "normal" means. Detached threads
00060     automatically clean up all of the internal resources associated with the
00061     thread when it exits. Joinable threads, on the other hand, must keep some
00062     state available for the ability to return values. To make sure that all
00063     memory allocated by the thread's internal structures gets freed, you must
00064     either join with the thread (with thd_join()) or detach it (with
00065     thd_detach()). The old KOS threading system only had what would be
00066     considered detached threads.
00067
00068     \sa semaphore_t, mutex_t, kthread_once_t, kthread_key_t, rw_semaphore_t
00069 */
00070
00071 /** \brief   Maximal thread priority.
00072
00073     This macro defines the maximum value for a thread's priority. Note that the
00074     larger this number, the lower the priority of the thread.
00075 */
00076 #define PRIO_MAX 4096
00077
00078 /** \brief   Default thread priority.
00079
00080     Threads are created by default with the priority specified.
00081 */
00082 #define PRIO_DEFAULT 10
00083
00084 /** \brief   Size of a kthread's label
00085
00086     Maximum number of characters in a thread's label or name
00087     (including NULL terminator).
00088 */
00089 #define KTHREAD_LABEL_SIZE  256
00090
00091 /** \brief   Size of a kthread's current directory
00092
00093     Maximum number of characters in a thread's current working
00094     directory (including NULL terminator).
00095 */
00096 #define KTHREAD_PWD_SIZE    256
00097
00098 /* Pre-define list/queue types */
00099 struct kthread;
00100
00101 /* \cond */
00102 TAILQ_HEAD(ktqueue, kthread);
00103 LIST_HEAD(ktlist, kthread);
00104 /* \endcond */
00105
00106 /** \brief   Control Block Header
00107
00108     Header preceeding the static TLS data segments as defined by
00109     the SH-ELF TLS ABI (version 1). This is what the thread pointer
00110     (GBR) points to for compiler access to thread-local data.
00111 */
00112 typedef struct tcbhead {
```

```
00113     void *dtv;              /**< \brief Dynamic TLS vector (unused) */
00114     uintptr_t pointer_guard; /**< \brief Pointer guard (unused) */
00115 } tcbhead_t;
00116
00117 /** \brief   Structure describing one running thread.
00118
00119     Each thread has one of these structures assigned to it, which holds all the
00120     data associated with the thread. There are various functions to manipulate
00121     the data in here, so you shouldn't generally do so manually.
00122
00123     \headerfile kos/thread.h
00124 */
00125 typedef struct kthread {
00126     /** \brief  Thread list handle. Not a function. */
00127     LIST_ENTRY(kthread) t_list;
00128
00129     /** \brief  Run/Wait queue handle. Once again, not a function. */
00130     TAILQ_ENTRY(kthread) thdq;
00131
00132     /** \brief  Timer queue handle (if applicable). Also not a function. */
00133     TAILQ_ENTRY(kthread) timerq;
00134
00135     /** \brief  Kernel thread id. */
00136     tid_t tid;
00137
00138     /** \brief  Static priority: 0..PRIO_MAX (higher means lower priority). */
00139     prio_t prio;
00140
00141     /** \brief  Thread flags.
00142         \see    thd_flags   */
00143     uint32_t flags;
00144
00145     /** \brief  Process state.
00146         \see    thd_states  */
00147     int state;
00148
00149     /** \brief  Generic wait target, if waiting.
00150         \see    kos/genwait.h   */
00151     void *wait_obj;
00152
00153     /** \brief  Generic wait message, if waiting.
00154         \see    kos/genwait.h   */
00155     const char *wait_msg;
00156
00157     /** \brief  Wait timeout callback.
00158
00159         If the genwait times out while waiting, this function will be called.
00160         This allows hooks for things like fixing up semaphore count values, etc.
00161
00162         \param  obj         The object that we were waiting on.
00163     */
00164     void (*wait_callback)(void *obj);
00165
00166     /** \brief  Next scheduled time.
00167         This value is used for sleep and timed block operations. This value is
00168         in milliseconds since the start of timer_ms_gettime(). This should be
00169         enough for something like 2 million years of wait time. ;) */
00170     uint64_t wait_timeout;
00171
00172     /** \brief  Thread label.
00173         This value is used when printing out a user-readable process listing. */
00174     char label[KTHREAD_LABEL_SIZE];
00175
00176     /** \brief  Current file system path. */
00177     char pwd[KTHREAD_PWD_SIZE];
00178
00179     /** \brief  Register store -- used to save thread context. */
00180     irq_context_t context;
00181
00182     /** \brief  Thread private stack.
00183         This should be a pointer to the base of a stack page. */
00184     uint32_t *stack;
00185
00186     /** \brief  Size of the thread's stack, in bytes. */
00187     uint32_t stack_size;
00188
00189     /** \brief  Thread errno variable. */
00190     int thd_errno;
00191
00192     /** \brief  Our reent struct for newlib. */
00193     struct _reent thd_reent;
```

```
00194
00195     /** \brief  OS-level thread-local storage.
00196         \see    kos/tls.h   */
00197     struct kthread_tls_kv_list tls_list;
00198
00199     /** \brief Compiler-level thread-local storage. */
00200     tcbhead_t* tcbhead;
00201
00202     /** \brief  Return value of the thread function.
00203         This is only used in joinable threads.  */
00204     void *rv;
00205 } kthread_t;
00206
00207 /** \name     Thread flag values
00208     \brief    kthread_t::flags values
00209
00210     These are possible values for the flags field on the kthread_t structure.
00211     These can be ORed together.
00212
00213     @{
00214 */
00215 #define THD_DEFAULTS    0       /**< \brief Defaults: no flags */
00216 #define THD_USER        1       /**< \brief Thread runs in user mode */
00217 #define THD_QUEUED      2       /**< \brief Thread is in the run queue */
00218 #define THD_DETACHED    4       /**< \brief Thread is detached */
00219 /** @} */
00220
00221 /** \name     Thread states
00222     \brief    kthread_t::state values
00223
00224     Each thread in the system is in exactly one of this set of states.
00225
00226     @{
00227 */
00228 #define STATE_ZOMBIE    0x0000  /**< \brief Waiting to die */
00229 #define STATE_RUNNING   0x0001  /**< \brief Process is "current" */
00230 #define STATE_READY     0x0002  /**< \brief Ready to be scheduled */
00231 #define STATE_WAIT      0x0003  /**< \brief Blocked on a genwait */
00232 #define STATE_FINISHED  0x0004  /**< \brief Finished execution */
00233 /** @} */
00234
00235 /** \brief   Thread creation attributes.
00236
00237     This structure allows you to specify the various attributes for a thread to
00238     have when it is created. These can only be modified (in general) at thread
00239     creation time (with the exception of detaching a thread, which can be done
00240     later with thd_detach()).
00241
00242     Leaving any of the attributes in this structure 0 will set them to their
00243     default value.
00244
00245     \headerfile kos/thread.h
00246 */
00247 typedef struct kthread_attr {
00248     /** \brief  1 for a detached thread. */
00249     int create_detached;
00250
00251     /** \brief  Set the size of the stack to be created. */
00252     uint32_t stack_size;
00253
00254     /** \brief  Pre-allocate a stack for the thread.
00255         \note   If you use this attribute, you must also set stack_size. */
00256     void *stack_ptr;
00257
00258     /** \brief  Set the thread's priority. */
00259     prio_t prio;
00260
00261     /** \brief  Thread label. */
00262     const char *label;
00263 } kthread_attr_t;
00264
00265 /** \name   Threading system modes
00266     \brief  kthread mode values
00267
00268     The threading system will always be in one of the following modes. This
00269     represents either pre-emptive scheduling or an un-initialized state.
00270
00271     @{
00272 */
00273 #define THD_MODE_NONE       -1  /**< \brief Threads not running */
00274 #define THD_MODE_COOP       0   /**< \brief Cooperative mode \deprecated */
```

```
00275 #define THD_MODE_PREEMPT    1   /**< \brief Preemptive threading mode */
00276 /** @} */
00277
00278 /** \cond The currently executing thread -- Do not manipulate directly! */
00279 extern kthread_t *thd_current;
00280 /** \endcond */
00281
00282 /** \brief   Block the current thread.
00283
00284     Blocks the calling thread and performs a reschedule as if a context switch
00285     timer had been executed. This is useful for, e.g., blocking on sync
00286     primitives. The param 'mycxt' should point to the calling thread's context
00287     block. This is implemented in arch-specific code.
00288
00289     The meaningfulness of the return value depends on whether the unblocker set
00290     a return value or not.
00291
00292     \param  mycxt           The IRQ context of the calling thread.
00293
00294     \return                 Whatever the unblocker deems necessary to return.
00295 */
00296 int thd_block_now(irq_context_t *mycxt);
00297
00298 /** \brief   Find a new thread to swap in.
00299
00300     This function looks at the state of the system and returns a new thread
00301     context to swap in. This is called from thd_block_now() and from the
00302     preemptive context switcher. Note that thd_current might be NULL on entering
00303     this function, if the caller blocked itself.
00304
00305     It is assumed that by the time this returns, the irq_srt_addr and
00306     thd_current will be updated.
00307
00308     \return                 The IRQ context of the thread selected.
00309 */
00310 irq_context_t *thd_choose_new(void);
00311
00312 /** \brief       Given a thread ID, locates the thread structure.
00313     \relatesalso kthread_t
00314
00315     \param  tid             The thread ID to retrieve.
00316
00317     \return                 The thread on success, NULL on failure.
00318 */
00319 kthread_t *thd_by_tid(tid_t tid);
00320
00321 /** \brief       Enqueue a process in the runnable queue.
00322     \relatesalso kthread_t
00323
00324     This function adds a thread to the runnable queue after the process group of
00325     the same priority if front_of_line is zero, otherwise queues it at the front
00326     of its priority group. Generally, you will not have to do this manually.
00327
00328     \param  t               The thread to queue.
00329     \param  front_of_line   Set to 1 to put this thread in front of other
00330                             threads of the same priority, 0 to put it behind the
00331                             other threads (normal behavior).
00332
00333     \sa thd_remove_from_runnable
00334 */
00335 void thd_add_to_runnable(kthread_t *t, int front_of_line);
00336
00337 /** \brief       Removes a thread from the runnable queue, if it's there.
00338     \relatesalso kthread_t
00339
00340     This function removes a thread from the runnable queue, if it is currently
00341     in that queue. Generally, you shouldn't have to do this manually, as waiting
00342     on synchronization primitives and the like will do this for you if needed.
00343
00344     \param  thd             The thread to remove from the runnable queue.
00345
00346     \retval 0               On success, or if the thread isn't runnable.
00347
00348     \sa thd_add_to_runnable
00349 */
00350 int thd_remove_from_runnable(kthread_t *thd);
00351
00352 /** \brief       Create a new thread.
00353     \relatesalso kthread_t
00354
00355     This function creates a new kernel thread with default parameters to run the
```

```
00356     given routine. The thread will terminate and clean up resources when the
00357     routine completes if the thread is created detached, otherwise you must
00358     join the thread with thd_join() to clean up after it.
00359
00360     \param  detach          Set to 1 to create a detached thread. Set to 0 to
00361                             create a joinable thread.
00362     \param  routine         The function to call in the new thread.
00363     \param  param           A parameter to pass to the function called.
00364
00365     \return                 The new thread on success, NULL on failure.
00366
00367     \sa thd_create_ex, thd_destroy
00368 */
00369 kthread_t *thd_create(int detach, void *(*routine)(void *param), void *param);
00370
00371 /** \brief   Create a new thread with the specified set of attributes.
00372     \relatesalso kthread_t
00373
00374     This function creates a new kernel thread with the specified set of
00375     parameters to run the given routine.
00376
00377     \param  attr            A set of thread attributes for the created thread.
00378                             Passing NULL will initialize all attributes to their
00379                             default values.
00380     \param  routine         The function to call in the new thread.
00381     \param  param           A parameter to pass to the function called.
00382
00383     \return                 The new thread on success, NULL on failure.
00384
00385     \sa thd_create, thd_destroy
00386 */
00387 kthread_t *thd_create_ex(const kthread_attr_t *__RESTRICT attr,
00388                          void *(*routine)(void *param), void *param);
00389
00390 /** \brief       Brutally kill the given thread.
00391     \relatesalso kthread_t
00392
00393     This function kills the given thread, removing it from the execution chain,
00394     cleaning up thread-local data and other internal structures. In general, you
00395     shouldn't call this function at all.
00396
00397     \warning
00398     You should never call this function on the current thread.
00399
00400     \param  thd             The thread to destroy.
00401     \retval 0               On success.
00402
00403     \sa thd_create
00404 */
00405 int thd_destroy(kthread_t *thd);
00406
00407 /** \brief   Exit the current thread.
00408
00409     This function ends the execution of the current thread, removing it from all
00410     execution queues. This function will never return to the thread. Returning
00411     from the thread's function is equivalent to calling this function.
00412
00413     \param  rv              The return value of the thread.
00414 */
00415 void thd_exit(void *rv) __noreturn;
00416
00417 /** \brief   Force a thread reschedule.
00418
00419     This function is the thread scheduler, and is generally called from a timer
00420     interrupt. You will most likely never have a reason to call this function
00421     directly.
00422
00423     For most cases, you'll want to set front_of_line to zero, but read the
00424     comments in kernel/thread/thread.c for more info, especially if you need to
00425     guarantee low latencies. This function just updates irq_srt_addr and
00426     thd_current. Set 'now' to non-zero if you want to use a particular system
00427     time for checking timeouts.
00428
00429     \param  front_of_line   Set to 0, unless you have a good reason not to.
00430     \param  now             Set to 0, unless you have a good reason not to.
00431
00432     \sa thd_schedule_next
00433 */
00434 void thd_schedule(int front_of_line, uint64_t now);
00435
00436 /** \brief       Force a given thread to the front of the queue.
```

```
00437     \relatesalso kthread_t
00438
00439     This function promotes the given thread to be the next one that will be
00440     swapped in by the scheduler. This function is only callable inside an
00441     interrupt context (it simply returns otherwise).
00442 */
00443 void thd_schedule_next(kthread_t *thd);
00444
00445 /** \brief   Throw away the current thread's timeslice.
00446
00447     This function manually yields the current thread's timeslice to the system,
00448     forcing a reschedule to occur.
00449 */
00450 void thd_pass(void);
00451
00452 /** \brief   Sleep for a given number of milliseconds.
00453
00454     This function puts the current thread to sleep for the specified amount of
00455     time. The thread will be removed from the runnable queue until the given
00456     number of milliseconds passes. That is to say that the thread will sleep for
00457     at least the given number of milliseconds. If another thread is running, it
00458     will likely sleep longer.
00459
00460     \param  ms              The number of milliseconds to sleep.
00461 */
00462 void thd_sleep(int ms);
00463
00464 /** \brief       Set a thread's priority value.
00465     \relatesalso kthread_t
00466
00467     This function is used to change the priority value of a thread. If the
00468     thread is scheduled already, it will be rescheduled with the new priority
00469     value.
00470
00471     \param  thd             The thread to change the priority of.
00472     \param  prio            The priority value to assign to the thread.
00473
00474     \retval 0               On success.
00475     \retval -1              thd is NULL.
00476     \retval -2              prio requested was out of range.
00477 */
00478 int thd_set_prio(kthread_t *thd, prio_t prio);
00479
00480 /** \brief       Retrieve the current thread's kthread struct.
00481     \relatesalso kthread_t
00482
00483     \return                 The current thread's structure.
00484 */
00485 kthread_t *thd_get_current(void);
00486
00487 /** \brief       Retrieve the thread's label.
00488     \relatesalso kthread_t
00489
00490     \param  thd             The thread to retrieve from.
00491
00492     \return                 The human-readable label of the thread.
00493
00494     \sa thd_set_label
00495 */
00496 const char *thd_get_label(kthread_t *thd);
00497
00498 /** \brief       Set the thread's label.
00499     \relatesalso kthread_t
00500
00501     This function sets the label of a thread, which is simply a human-readable
00502     string that is used to identify the thread. These labels aren't used for
00503     anything internally, and you can give them any label you want. These are
00504     mainly seen in the printouts from thd_pslist() or thd_pslist_queue().
00505
00506     \param  thd             The thread to set the label of.
00507     \param  label           The string to set as the label.
00508
00509     \sa thd_get_label
00510 */
00511 void thd_set_label(kthread_t *thd, const char *__RESTRICT label);
00512
00513 /** \brief       Retrieve the thread's current working directory.
00514     \relatesalso kthread_t
00515
00516     This function retrieves the working directory of a thread. Generally, you
00517     will want to use either fs_getwd() or one of the standard C functions for
```

```
00518     doing this, but this is here in case you need it when the thread isn't
00519     active for some reason.
00520
00521     \param  thd             The thread to retrieve from.
00522
00523     \return                 The thread's working directory.
00524
00525     \sa thd_set_pd
00526 */
00527 const char *thd_get_pwd(kthread_t *thd);
00528
00529 /** \brief       Set the thread's current working directory.
00530     \relatesalso kthread_t
00531
00532     This function will set the working directory of a thread. Generally, you
00533     will want to use either fs_chdir() or the standard C chdir() function to
00534     do this, but this is here in case you need to do it while the thread isn't
00535     active for some reason.
00536
00537     \param  thd             The thread to set the working directory of.
00538     \param  pwd             The directory to set as active.
00539
00540     \sa thd_get_pwd
00541 */
00542 void thd_set_pwd(kthread_t *thd, const char *__RESTRICT pwd);
00543
00544 /** \brief       Retrieve a pointer to the thread errno.
00545     \relatesalso kthread_t
00546
00547     This function retrieves a pointer to the errno value for the thread. You
00548     should generally just use the errno variable to access this.
00549
00550     \param  thd             The thread to retrieve from.
00551
00552     \return                 A pointer to the thread's errno.
00553 */
00554 int *thd_get_errno(kthread_t *thd);
00555
00556 /** \brief       Retrieve a pointer to the thread reent struct.
00557     \relatesalso kthread_t
00558
00559     This function is used to retrieve some internal state that is used by
00560     newlib to provide a reentrant libc.
00561
00562     \param  thd             The thread to retrieve from.
00563
00564     \return                 The thread's reent struct.
00565 */
00566 struct _reent *thd_get_reent(kthread_t *thd);
00567
00568 /** \brief   Change threading modes.
00569
00570     This function changes the current threading mode of the system.
00571     With preemptive threading being the only mode.
00572
00573     \deprecated
00574     This is now deprecated.
00575
00576     \param  mode            One of the THD_MODE values.
00577     \return                 The old mode of the threading system.
00578
00579     \sa thd_get_mode
00580 */
00581 int thd_set_mode(int mode) __deprecated;
00582
00583 /** \brief   Fetch the current threading mode.
00584
00585     With preemptive threading being the only mode.
00586
00587     \deprecated
00588     This is now deprecated.
00589
00590     \return                 The current mode of the threading system.
00591
00592     \sa thd_set_mode
00593 */
00594 int thd_get_mode(void) __deprecated;
00595
00596 /** \brief       Wait for a thread to exit.
00597     \relatesalso kthread_t
00598
```

```
00599     This function "joins" a joinable thread. This means effectively that the
00600     calling thread blocks until the speified thread completes execution. It is
00601     invalid to join a detached thread, only joinable threads may be joined.
00602
00603     \param  thd             The joinable thread to join.
00604     \param  value_ptr       A pointer to storage for the thread's return value,
00605                             or NULL if you don't care about it.
00606
00607     \return                 0 on success, or less than 0 if the thread is
00608                             non-existant or not joinable.
00609
00610     \sa thd_detach
00611 */
00612 int thd_join(kthread_t *thd, void **value_ptr);
00613
00614 /** \brief       Detach a joinable thread.
00615     \relatesalso kthread_t
00616
00617     This function switches the specified thread's mode from THD_MODE_JOINABLE
00618     to THD_MODE_DETACHED. This will ensure that the thread cleans up all of its
00619     internal resources when it exits.
00620
00621     \param  thd             The joinable thread to detach.
00622
00623     \return                 0 on success or less than 0 if the thread is
00624                             non-existant or already detached.
00625     \sa    thd_join()
00626 */
00627 int thd_detach(kthread_t *thd);
00628
00629 /** \brief       Iterate all threads and call the passed callback for each
00630     \relatesalso kthread_t
00631
00632     \param cb               The callback to call for each thread.
00633                             If a nonzero value is returned, iteration
00634                             ceases immediately.
00635     \param data             User data to be passed to the callback
00636
00637     \retval                 0 or the first nonzero value returned by \p cb.
00638
00639     \sa thd_pslist
00640 */
00641 int thd_each(int (*cb)(kthread_t *thd, void *user_data), void *data);
00642
00643 /** \brief   Print a list of all threads using the given print function.
00644
00645     \param  pf              The printf-like function to print with.
00646
00647     \retval 0               On success.
00648
00649     \sa thd_pslist_queue
00650 */
00651 int thd_pslist(int (*pf)(const char *fmt, ...));
00652
00653 /** \brief   Print a list of all queued threads using the given print function.
00654
00655     \param  pf              The printf-like function to print with.
00656
00657     \retval 0               On success.
00658
00659     \sa thd_pslist
00660 */
00661 int thd_pslist_queue(int (*pf)(const char *fmt, ...));
00662
00663 /** \brief   Initialize the threading system.
00664
00665     This is normally done for you by default when KOS starts. This will also
00666     initialize all the various synchronization primitives.
00667
00668     \retval -1              If threads are already initialized.
00669     \retval 0               On success.
00670
00671     \sa thd_shutdown
00672 */
00673 int thd_init(void);
00674
00675 /** \brief   Shutdown the threading system.
00676
00677     This is done for you by the normal shutdown procedure of KOS. This will
00678     also shutdown all the various synchronization primitives.
00679
```

```
00680     \sa thd_init
00681 */
00682 void thd_shutdown(void);
00683
00684 __END_DECLS
00685
00686 #endif  /* __KOS_THREAD_H */
```

9.88 include/kos/time.h File Reference

```
#include <kos/cdefs.h>
```

Macros

- #define TIME_UTC 1
- #define _POSIX_TIMERS 1
- #define CLOCK_MONOTONIC (CLOCK_REALTIME + 1)
- #define CLOCK_PROCESS_CPUTIME_ID (CLOCK_REALTIME + 2)
- #define CLOCK_THREAD_CPUTIME_ID (CLOCK_REALTIME + 3)

Functions

- int timespec_get (struct timespec *ts, int base)

9.88.1 Macro Definition Documentation

_POSIX_TIMERS

```
#define _POSIX_TIMERS 1
```

CLOCK_MONOTONIC

```
#define CLOCK_MONOTONIC (CLOCK_REALTIME + 1)
```

CLOCK_PROCESS_CPUTIME_ID

```
#define CLOCK_PROCESS_CPUTIME_ID (CLOCK_REALTIME + 2)
```

CLOCK_THREAD_CPUTIME_ID

```
#define CLOCK_THREAD_CPUTIME_ID (CLOCK_REALTIME + 3)
```

TIME_UTC

```
#define TIME_UTC 1
```

9.88.2 Function Documentation

timespec_get()

```
int timespec_get (
            struct timespec * ts,
            int base )  [extern]
```

9.89 time.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/time.h
00004    Copyright (C) 2023 Lawrence Sebald
00005 */
00006
00007 /* This will probably go away at some point in the future, if/when Newlib gets
00008    an implementation of this function. But for now, it's here. */
00009
00010 #ifndef _TIME_H_
00011 #error "Do not include this file directly. Use <time.h> instead."
00012 #endif /* !_TIME_H_ */
00013
00014 #ifndef __KOS_TIME_H
00015 #define __KOS_TIME_H
00016
00017 #include <kos/cdefs.h>
00018
00019 __BEGIN_DECLS
00020
00021 #if !defined(__STRICT_ANSI__) || (__STDC_VERSION__ >= 201112L) || (__cplusplus >= 201703L)
00022
00023 /* Forward declaration. */
00024 struct timespec;
00025
00026 #define TIME_UTC 1
00027
00028 extern int timespec_get(struct timespec *ts, int base);
00029
00030 #endif /* !defined(__STRICT_ANSI__) || (__STDC_VERSION__ >= 201112L) || (__cplusplus >= 201703L) */
00031
00032 #if !defined(__STRICT_ANSI__) || (_POSIX_C_SOURCE >= 199309L)
00033
00034 #ifndef _POSIX_TIMERS
00035 #define _POSIX_TIMERS 1
00036 #endif
00037 #ifdef _POSIX_MONOTONIC_CLOCK
00038 #define _POSIX_MONOTONIC_CLOCK 1
00039 #endif
00040 #ifdef _POSIX_CPUTIME
00041 #define _POSIX_CPUTIME 1
00042 #endif
00043 #ifdef _POSIX_THREAD_CPU_TIME
00044 #define _POSIX_THREAD_CPUTIME 1
00045 #endif
00046
00047 #define CLOCK_MONOTONIC           (CLOCK_REALTIME + 1)
00048 #define CLOCK_PROCESS_CPUTIME_ID (CLOCK_REALTIME + 2)
00049 #define CLOCK_THREAD_CPUTIME_ID   (CLOCK_REALTIME + 3)
00050
00051 #endif /* !defined(__STRICT_ANSI__) || (_POSIX_C_SOURCE >= 199309L) */
00052
00053 __END_DECLS
00054
00055 #endif /* !__KOS_TIME_H */
```

9.90 include/kos/tls.h File Reference

Thread-local storage support.

```
#include <sys/cdefs.h>
#include <sys/queue.h>
```

Data Structures

- struct kthread_tls_kv_t

  Thread-local storage key-value pair.

Typedefs

- typedef int kthread_key_t

  Thread-local storage key type.

Functions

- int kthread_key_create (kthread_key_t *key, void(*destructor)(void *))

  Create a new thread-local storage key.
- void * kthread_getspecific (kthread_key_t key)

  Retrieve a value associated with a TLS key.
- int kthread_setspecific (kthread_key_t key, const void *value)

  Set thread specific data for a key.
- int kthread_key_delete (kthread_key_t key)

  Delete a TLS key.

9.90.1 Detailed Description

Thread-local storage support.

This file contains the definitions used to support key/value pairs of thread-local storage in KOS.

Author

Lawrence Sebald

9.90.2 Typedef Documentation

kthread_key_t

```
typedef int kthread_key_t
```

Thread-local storage key type.

9.90.3 Function Documentation

kthread_getspecific()

```
void * kthread_getspecific (
            kthread_key_t key )
```

Retrieve a value associated with a TLS key.

This function retrieves the thread-specific data associated with the given key.

Parameters

key	The key to look up data for.

Returns

The data associated with the key, or NULL if the key is not valid or no data has been set in the current thread.

kthread_key_create()

```
int kthread_key_create (
            kthread_key_t * key,
            void(*)(void *) destructor )
```

Create a new thread-local storage key.

This function creates a new thread-local storage key that shall be visible to all threads. Each thread is then responsible for associating any data with the key that it deems fit (by default a thread will have no data associated with a newly created key).

Parameters

key	The key to use.
destructor	A destructor for use with this key. If it is non-NULL, and a value associated with the key is non-NULL at thread exit, then the destructor will be called with the value as its argument.

Return values

-1	On failure, and sets errno to one of the following: EPERM if called inside an interrupt and another call is in progress, ENOMEM if out of memory.
0	On success.

kthread_key_delete()

```
int kthread_key_delete (
            kthread_key_t key )
```

Delete a TLS key.

This function deletes a TLS key, removing all threads' values for the given key. This function *does not* cause any destructors to be called.

Parameters

key	The key to delete.

Return values

-1	On failure, and sets errno to one of the following: EINVAL if the key is invalid, EPERM if unsafe to call free.
0	On success.

kthread_setspecific()

```
int kthread_setspecific (
            kthread_key_t key,
            const void * value )
```

Set thread specific data for a key.

This function sets the thread-specific data associated with the given key.

Parameters

key	The key to set data for.
value	The thread-specific value to use.

Return values

-1	On failure, and sets errno to one of the following: EINVAL if the key is not valid, ENOMEM if out of memory, or EPERM if called inside an interrupt and another call is in progress.
0	On success.

9.91 tls.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
```

```
00002
00003    include/kos/tls.h
00004    Copyright (C) 2009, 2010 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   kos/tls.h
00009     \brief  Thread-local storage support.
00010     \ingroup kthreads
00011
00012     This file contains the definitions used to support key/value pairs of
00013     thread-local storage in KOS.
00014
00015     \author Lawrence Sebald
00016 */
00017
00018 #ifndef __KOS_TLS_H
00019 #define __KOS_TLS_H
00020
00021 #include <sys/cdefs.h>
00022
00023 __BEGIN_DECLS
00024
00025 #include <sys/queue.h>
00026
00027 /** \brief  Thread-local storage key type. */
00028 typedef int kthread_key_t;
00029
00030 /** \brief  Thread-local storage key-value pair.
00031
00032     This is the structure that is actually used to store the specific value for
00033     a thread for a single TLS key.
00034
00035     You will not end up using these directly at all in programs, as they are
00036     only used internally.
00037 */
00038 typedef struct kthread_tls_kv {
00039     /** \brief  List handle -- NOT a function. */
00040     LIST_ENTRY(kthread_tls_kv) kv_list;
00041
00042     /** \brief  The key associated with this data. */
00043     kthread_key_t key;
00044
00045     /** \brief  The value of the data. */
00046     void *data;
00047
00048     /** \brief  Optional destructor for the value (set per key). */
00049     void (*destructor)(void *);
00050 } kthread_tls_kv_t;
00051
00052 /** \cond */
00053 /* TLS Key-Value pair list type. */
00054 LIST_HEAD(kthread_tls_kv_list, kthread_tls_kv);
00055 /** \endcond */
00056
00057 /** \cond */
00058 /* Retrieve the next key value (i.e, what key the next kthread_key_create will
00059    use). This function is not meant for external use (although it won't really
00060    hurt anything for you to call it). */
00061 kthread_key_t kthread_key_next(void);
00062 /** \endcond */
00063
00064 /** \brief  Create a new thread-local storage key.
00065
00066     This function creates a new thread-local storage key that shall be visible
00067     to all threads. Each thread is then responsible for associating any data
00068     with the key that it deems fit (by default a thread will have no data
00069     associated with a newly created key).
00070
00071     \param  key         The key to use.
00072     \param  destructor  A destructor for use with this key. If it is non-NULL,
00073                         and a value associated with the key is non-NULL at
00074                         thread exit, then the destructor will be called with the
00075                         value as its argument.
00076     \retval -1      On failure, and sets errno to one of the following: EPERM if
00077                     called inside an interrupt and another call is in progress,
00078                     ENOMEM if out of memory.
00079     \retval 0       On success.
00080 */
00081 int kthread_key_create(kthread_key_t *key, void (*destructor)(void *));
00082
```

```
00083 /** \brief  Retrieve a value associated with a TLS key.
00084
00085     This function retrieves the thread-specific data associated with the given
00086     key.
00087
00088     \param  key     The key to look up data for.
00089     \return The data associated with the key, or NULL if the key is not valid or
00090             no data has been set in the current thread.
00091 */
00092 void *kthread_getspecific(kthread_key_t key);
00093
00094 /** \brief  Set thread specific data for a key.
00095
00096     This function sets the thread-specific data associated with the given key.
00097
00098     \param  key     The key to set data for.
00099     \param  value   The thread-specific value to use.
00100     \retval -1      On failure, and sets errno to one of the following: EINVAL
00101                     if the key is not valid, ENOMEM if out of memory, or EPERM
00102                     if called inside an interrupt and another call is in
00103                     progress.
00104     \retval 0       On success.
00105 */
00106 int kthread_setspecific(kthread_key_t key, const void *value);
00107
00108 /** \brief  Delete a TLS key.
00109
00110     This function deletes a TLS key, removing all threads' values for the given
00111     key. This function <em>does not</em> cause any destructors to be called.
00112
00113     \param  key     The key to delete.
00114     \retval -1      On failure, and sets errno to one of the following: EINVAL
00115                     if the key is invalid, EPERM if unsafe to call free.
00116     \retval 0       On success.
00117 */
00118 int kthread_key_delete(kthread_key_t key);
00119
00120 /** \cond */
00121 /* Delete the destructor for a given key. This function is for internal use
00122    only! */
00123 void kthread_key_delete_destructor(kthread_key_t key);
00124
00125 /* Initialization and shutdown. Once again, internal use only. */
00126 int kthread_tls_init(void);
00127 void kthread_tls_shutdown(void);
00128 /** \endcond */
00129
00130 __END_DECLS
00131
00132 #endif /* __KOS_TLS_H */
```

9.92 include/libgen.h File Reference

Definitions for pattern matching functions.

```
#include <sys/cdefs.h>
```

Functions

- char * basename (char *path)

  Get the last component of a pathname.
- char * dirname (char *path)

  Get the parent directory of a file pathname.

9.92.1 Detailed Description

Definitions for pattern matching functions.

This file contains the definitions for the functions basename() and dirname() as specified by the POSIX standard. How the POSIX people came up with this filename for this stuff, I don't know.

The GNU C library defines a slightly different version of these functions in <string.h> instead of here. Please note that these functions, unlike the GNU versions that are in <string.h> on Linux do modify their input values, as POSIX allows.

Author

Lawrence Sebald

9.92.2 Function Documentation

basename()

```
char * basename (
            char * path )
```

Get the last component of a pathname.

This function retrieves the basename from a given path. The basename of a path is the non-directory component thereof, minus any trailing '/' characters. This function does not attempt to perform any sort of path resolution.

Note

This function may modify its input and the returned value may be a pointer to part of its input.

Parameters

path	The path to extract the basename from.

Returns

A pointer to the basename of the given path.

dirname()

```
char * dirname (
            char * path )
```

Get the parent directory of a file pathname.

This function retrieves the name of the parent directory of the file pathname specified. This function does not attempt to perform any sort of path resolution to check for the existence of the specified diretory or to resolve any symbolic links.

Note

This function may modify its input and the returned value may be a pointer to part of its input.

Parameters

path	The path to extract the parent directory from.

Returns

A pointer to the parent directory of the given path.

9.93 libgen.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    libgen.h
00004    Copyright (C) 2020 Lawrence Sebald
00005 */
00006
00007 /** \file   libgen.h
00008     \brief  Definitions for pattern matching functions.
00009
00010     This file contains the definitions for the functions basename() and
00011     dirname() as specified by the POSIX standard. How the POSIX people came
00012     up with this filename for this stuff, I don't know.
00013
00014     The GNU C library defines a slightly different version of these functions in
00015     <string.h> instead of here. Please note that these functions, unlike the
00016     GNU versions that are in <string.h> on Linux do modify their input values,
00017     as POSIX allows.
00018
00019     \author Lawrence Sebald
00020 */
00021
00022 #ifndef __LIBGEN_H
00023 #define __LIBGEN_H
00024
00025 #include <sys/cdefs.h>
00026 __BEGIN_DECLS
00027
00028 /** \brief  Get the last component of a pathname.
00029
00030     This function retrieves the basename from a given path. The basename of a
00031     path is the non-directory component thereof, minus any trailing '/'
00032     characters. This function does not attempt to perform any sort of path
00033     resolution.
00034
00035     \note   This function may modify its input and the returned value may be a
00036             pointer to part of its input.
00037
00038     \param  path            The path to extract the basename from.
00039     \return                 A pointer to the basename of the given path.
00040 */
00041 char *basename(char *path);
00042
00043 /** \brief  Get the parent directory of a file pathname.
00044
00045     This function retrieves the name of the parent directory of the file
00046     pathname specified. This function does not attempt to perform any sort of
00047     path resolution to check for the existence of the specified diretory or to
00048     resolve any symbolic links.
00049
00050     \note   This function may modify its input and the returned value may be a
00051             pointer to part of its input.
00052
00053     \param  path            The path to extract the parent directory from.
00054     \return                 A pointer to the parent directory of the given path.
```

```
00055 */
00056 char *dirname(char *path);
00057
00058 __END_DECLS
00059
00060 #endif /* !__LIBGEN_H */
```

9.94 include/malloc.h File Reference

Standard C Malloc functionality.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Data Structures

- struct mallinfo

  ANSI C functions.

Macros

- #define M_MXFAST 1
- #define DEFAULT_MXFAST 64
- #define M_TRIM_THRESHOLD -1
- #define DEFAULT_TRIM_THRESHOLD (256∗1024)
- #define M_TOP_PAD -2
- #define DEFAULT_TOP_PAD 0
- #define M_MMAP_THRESHOLD -3
- #define DEFAULT_MMAP_THRESHOLD (256∗1024)
- #define M_MMAP_MAX -4
- #define DEFAULT_MMAP_MAX 65536

Functions

- void ∗ malloc (size_t size)

  Allocate a block of memory.
- void ∗ calloc (size_t nmemb, size_t size)

  Allocate a block of memory and initialize it to 0.
- void free (void ∗ptr)

  Release a previously allocated block of memory.
- void ∗ realloc (void ∗ptr, size_t size)

  Change the size of a previously allocated block of memory.
- void ∗ memalign (size_t alignment, size_t size)

  Allocate a block of memory aligned to a specified block size.
- void ∗ valloc (size_t size)

  Allocate a block of memory aligned to the system page size.
- void ∗ aligned_alloc (size_t alignment, size_t size)

Allocate memory aligned to a specified block size.

- struct mallinfo mallinfo ()

  Sets tunable parameters for malloc related options.
- int mallopt (int, int)
- void malloc_stats (void)

  Debug function.
- int malloc_irq_safe (void)

  Determine if it is safe to call malloc() in an IRQ context.
- int mem_check_block (void *p)

  Only available with KM_DBG.
- int mem_check_all (void)

  Only available with KM_DBG.

9.94.1 Detailed Description

Standard C Malloc functionality.

This implements standard C heap allocation, deallocation, and stats.

Author

Megan Potter

Lawrence Sebald

9.94.2 Macro Definition Documentation

DEFAULT_MMAP_MAX

```
#define DEFAULT_MMAP_MAX 65536
```

DEFAULT_MMAP_THRESHOLD

```
#define DEFAULT_MMAP_THRESHOLD (256*1024)
```

DEFAULT_MXFAST

```
#define DEFAULT_MXFAST 64
```

DEFAULT_TOP_PAD

```
#define DEFAULT_TOP_PAD 0
```

DEFAULT_TRIM_THRESHOLD

```
#define DEFAULT_TRIM_THRESHOLD (256*1024)
```

M_MMAP_MAX

```
#define M_MMAP_MAX -4
```

M_MMAP_THRESHOLD

```
#define M_MMAP_THRESHOLD -3
```

M_MXFAST

```
#define M_MXFAST 1
```

M_TOP_PAD

```
#define M_TOP_PAD -2
```

M_TRIM_THRESHOLD

```
#define M_TRIM_THRESHOLD -1
```

9.94.3 Function Documentation

aligned_alloc()

```
void * aligned_alloc (
            size_t alignment,
            size_t size )
```

Allocate memory aligned to a specified block size.

This function is the standard-compliant C11 method for allocating aligned blocks of memory. It works mostly the same as memalign(), although the size must be a multiple of the alignment.

Parameters

alignment	Required alignment of the memory block.
size	Number of bytes of memory to allocate.

Returns

A pointer to the newly allocated memory block.

calloc()

```
void * calloc (
            size_t nmemb,
            size_t size )
```

Allocate a block of memory and initialize it to 0.

This function allocates a block of memory of size * nmemb bytes, initializing it to all zero bytes in the process. In other words, this function allocates an array of nmemb elements that are each size bytes in length. Just as with malloc(), you are responsible for calling free() on the block of memory when you are done with it.

Parameters

nmemb	Number of elements to allocate space for.
size	Size of each element in the array.

Returns

A pointer to the newly allocated block of memory, or NULL on failure.

See also

free

malloc

free()

```
void free (
            void * ptr )
```

Release a previously allocated block of memory.

This function frees memory that has been previously allocated by way of any of the allocation functions in this file (malloc(), calloc(), memalign(), valloc(), or aligned_alloc()), releasing the memory to be potentially used for any future allocations.

Parameters

ptr	A pointer to the block of memory to be freed.

Note

Passing a NULL pointer to this function has no effect.

Attempting to free the same block of memory twice exhibits undefined behavior (i.e, it may crash, it may do something else evil, etc).

Attempting to free a block of memory that was not allocated by way of the normal allocation functions exhibits undefined behavior.

mallinfo()

```
struct mallinfo mallinfo ( )
```

Sets tunable parameters for malloc related options.

malloc()

```
void * malloc (
            size_t size )
```

Allocate a block of memory.

This function allocates a block of memory of the specified number of bytes on the heap. This memory must later be freed by way of the free() function. The memory *is not* freed simply because a pointer to it goes out of scope.

Parameters

size	Number of bytes of memory to allocate.

Returns

A pointer to the newly allocated block of memory, or NULL on failure.

See also

free

calloc

Note

The block of memory returned is completely uninitialized and may contain random data.

malloc_irq_safe()

```
int malloc_irq_safe (
            void  )
```

Determine if it is safe to call malloc() in an IRQ context.

This function checks the value of the internal spinlock that is used for malloc() to ensure that a call to it will not freeze the running process. This is only really useful in an IRQ context to ensure that a call to malloc() (or some other memory allocation function) won't cause a deadlock.

Return values

1	If it is safe to call malloc() in the current IRQ.
0	Otherwise.

malloc_stats()

```
void malloc_stats (
            void  )
```

Debug function.

mallopt()

```
int mallopt (
            int ,
            int  )
```

mem_check_all()

```
int mem_check_all (
            void  )
```

Only available with KM_DBG.

mem_check_block()

```
int mem_check_block (
            void * p )
```

Only available with KM_DBG.

memalign()

```
void * memalign (
            size_t alignment,
            size_t size )
```

Allocate a block of memory aligned to a specified block size.

This function allocates a block of memory such that the lowest address in the block is aligned to a multiple of alignment bytes. This is useful, for instance, for things like DMA that require aligned blocks of memory to work.

Over-reliance on memalign will most certainly cause memory fragmentation, so you should only use it when it is actually necessary.

Parameters

alignment	The alignment requested for the block of memory. This must be a power-of-two.
size	The number of bytes of memory to allocate.

Returns

A pointer to the newly allocated block of memory on success, or NULL on failure.

Note

All memory allocation functions will have their blocks aligned on 8-byte boundaries. There is no reason to call this function if you need less than 16-byte alignment.

realloc()

```
void * realloc (
            void * ptr,
            size_t size )
```

Change the size of a previously allocated block of memory.

This function changes the size of the previously allocated block of memory from its current size to the specified size. This may involve reallocating and copying the data from the original pointer to a new location in memory.

If this function returns non-NULL, the old pointer is considered invalid and should not be used (unless, of course, the returned pointer is the same as the old pointer). No action is needed to clean up the old pointer, as this function will have freed it if necessary.

If this function returns NULL, the old pointer is still valid and has not had its size changed.

Parameters

ptr	A pointer to the block of memory to resize. It must have been previously allocated by way of one of the memory allocation functions in this file (or must be NULL).
size	The requested size of the new block of memory.

Returns

A pointer to the newly allocted/resized block of memory on success, or NULL on failure.

Note

If ptr is NULL on a call to this function, the function acts the same as malloc(size).

If ptr is non-NULL and size is zero, this function does not∗ free the block of memory – it simply returns a block that is of minimal size. By the standard, this pointer must not be dereferenced.

valloc()

```
void * valloc (
            size_t size )
```

Allocate a block of memory aligned to the system page size.

This function allocates a block of memory such that the lowest address in the block is aligned to the system page size (typically 4096 bytes). It basically ends up doing return memalign(PAGESIZE, size).

Parameters

size	The number of bytes of memory to allocate.

Returns

A pointer to the newly allocated block of memory on success, or NULL on failure.

See also

PAGESIZE

<arch/arch.h>

9.95 malloc.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    malloc.h
```

```
00004    Copyright (C) 2003 Megan Potter
00005    Copyright (C) 2015 Lawrence Sebald
00006
00007 */
00008
00009 /** \file   malloc.h
00010     \brief  Standard C Malloc functionality
00011
00012     This implements standard C heap allocation, deallocation, and stats.
00013
00014     \author Megan Potter
00015     \author Lawrence Sebald
00016 */
00017
00018 #ifndef __MALLOC_H
00019 #define __MALLOC_H
00020
00021 #include <sys/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 #include <arch/types.h>
00025
00026 /* Unlike previous versions, we totally decouple the implementation from
00027    the declarations. */
00028
00029 /** \brief ANSI C functions */
00030 struct mallinfo {
00031     /** \brief non-mmapped space allocated from system */
00032     int arena;
00033     /** \brief number of free chunks */
00034     int ordblks;
00035     /** \brief number of fastbin blocks */
00036     int smblks;
00037     /** \brief number of mmapped regions */
00038     int hblks;
00039     /** \brief space in mmapped regions */
00040     int hblkhd;
00041     /** \brief maximum total allocated space */
00042     int usmblks;
00043     /** \brief space available in freed fastbin blocks */
00044     int fsmblks;
00045     /** \brief total allocated space */
00046     int uordblks;
00047     /** \brief total free space */
00048     int fordblks;
00049     /** \brief top-most, releasable (via malloc_trim) space */
00050     int keepcost;
00051 };
00052
00053 /** \brief  Allocate a block of memory.
00054
00055     This function allocates a block of memory of the specified number of bytes
00056     on the heap. This memory must later be freed by way of the free() function.
00057     The memory *is not* freed simply because a pointer to it goes out of scope.
00058
00059     \param  size            Number of bytes of memory to allocate.
00060     \return                 A pointer to the newly allocated block of memory, or
00061                             NULL on failure.
00062
00063     \see                    free
00064     \see                    calloc
00065     \note                   The block of memory returned is completely
00066                             uninitialized and may contain random data.
00067 */
00068 void *malloc(size_t size);
00069
00070 /** \brief  Allocate a block of memory and initialize it to 0.
00071
00072     This function allocates a block of memory of size * nmemb bytes,
00073     initializing it to all zero bytes in the process. In other words, this
00074     function allocates an array of nmemb elements that are each size bytes in
00075     length. Just as with malloc(), you are responsible for calling free() on the
00076     block of memory when you are done with it.
00077
00078     \param  nmemb           Number of elements to allocate space for.
00079     \param  size            Size of each element in the array.
00080     \return                 A pointer to the newly allocated block of memory, or
00081                             NULL on failure.
00082
00083     \see                    free
00084     \see                    malloc
```

```
00085 */
00086 void *calloc(size_t nmemb, size_t size);
00087
00088 /** \brief  Release a previously allocated block of memory.
00089
00090     This function frees memory that has been previously allocated by way of any
00091     of the allocation functions in this file (malloc(), calloc(), memalign(),
00092     valloc(), or aligned_alloc()), releasing the memory to be potentially used
00093     for any future allocations.
00094
00095     \param  ptr             A pointer to the block of memory to be freed.
00096
00097     \note                   Passing a NULL pointer to this function has no
00098                             effect.
00099     \note                   Attempting to free the same block of memory twice
00100                             exhibits undefined behavior (i.e, it may crash, it
00101                             may do something else evil, etc).
00102     \note                   Attempting to free a block of memory that was not
00103                             allocated by way of the normal allocation functions
00104                             exhibits undefined behavior.
00105 */
00106 void free(void *ptr);
00107
00108 /** \brief  Change the size of a previously allocated block of memory.
00109
00110     This function changes the size of the previously allocated block of memory
00111     from its current size to the specified size. This may involve reallocating
00112     and copying the data from the original pointer to a new location in memory.
00113
00114     If this function returns non-NULL, the old pointer is considered invalid and
00115     should not be used (unless, of course, the returned pointer is the same as
00116     the old pointer). No action is needed to clean up the old pointer, as this
00117     function will have freed it if necessary.
00118
00119     If this function returns NULL, the old pointer is still valid and has not
00120     had its size changed.
00121
00122     \param  ptr             A pointer to the block of memory to resize. It must
00123                             have been previously allocated by way of one of the
00124                             memory allocation functions in this file (or must
00125                             be NULL).
00126     \param  size            The requested size of the new block of memory.
00127     \return                 A pointer to the newly allocted/resized block of
00128                             memory on success, or NULL on failure.
00129
00130     \note                   If ptr is NULL on a call to this function, the
00131                             function acts the same as malloc(size).
00132     \note                   If ptr is non-NULL and size is zero, this function
00133                             *does not* free the block of memory -- it simply
00134                             returns a block that is of minimal size. By the
00135                             standard, this pointer must not be dereferenced.
00136 */
00137 void *realloc(void *ptr, size_t size);
00138
00139 /** \brief  Allocate a block of memory aligned to a specified block size.
00140
00141     This function allocates a block of memory such that the lowest address in
00142     the block is aligned to a multiple of alignment bytes. This is useful, for
00143     instance, for things like DMA that require aligned blocks of memory to work.
00144
00145     Over-reliance on memalign will most certainly cause memory fragmentation, so
00146     you should only use it when it is actually necessary.
00147
00148     \param  alignment       The alignment requested for the block of memory.
00149                             This must be a power-of-two.
00150     \param  size            The number of bytes of memory to allocate.
00151     \return                 A pointer to the newly allocated block of memory on
00152                             success, or NULL on failure.
00153
00154     \note                   All memory allocation functions will have their
00155                             blocks aligned on 8-byte boundaries. There is no
00156                             reason to call this function if you need less than
00157                             16-byte alignment.
00158 */
00159 void *memalign(size_t alignment, size_t size);
00160
00161 /** \brief  Allocate a block of memory aligned to the system page size.
00162
00163     This function allocates a block of memory such that the lowest address in
00164     the block is aligned to the system page size (typically 4096 bytes). It
00165     basically ends up doing return memalign(PAGESIZE, size).
```

```
00166
00167     \param  size            The number of bytes of memory to allocate.
00168     \return                 A pointer to the newly allocated block of memory on
00169                             success, or NULL on failure.
00170
00171     \see                    PAGESIZE
00172     \see                    <arch/arch.h>
00173 */
00174 void *valloc(size_t size);
00175
00176 #if !defined(__STRICT_ANSI__) || (__STDC_VERSION__ >= 201112L)
00177
00178 /** \brief  Allocate memory aligned to a specified block size.
00179
00180     This function is the standard-compliant C11 method for allocating aligned
00181     blocks of memory. It works mostly the same as memalign(), although the
00182     size must be a multiple of the alignment.
00183
00184     \param      alignment   Required alignment of the memory block.
00185     \param      size        Number of bytes of memory to allocate.
00186     \return                 A pointer to the newly allocated memory block.
00187 */
00188 void *aligned_alloc(size_t alignment, size_t size);
00189
00190 #endif /* !defined(__STRICT_ANSI__) || (__STDC_VERSION__ >= 201112L) */
00191
00192 /** \brief Sets tunable parameters for malloc related options.
00193 */
00194 struct mallinfo mallinfo();
00195
00196 /* mallopt defines */
00197 #define M_MXFAST 1
00198 #define DEFAULT_MXFAST 64
00199
00200 #define M_TRIM_THRESHOLD -1
00201 #define DEFAULT_TRIM_THRESHOLD (256*1024)
00202
00203 #define M_TOP_PAD -2
00204 #define DEFAULT_TOP_PAD 0
00205
00206 #define M_MMAP_THRESHOLD -3
00207 #define DEFAULT_MMAP_THRESHOLD (256*1024)
00208
00209 #define M_MMAP_MAX -4
00210 #define DEFAULT_MMAP_MAX 65536
00211 int  mallopt(int, int);
00212
00213 /** \brief Debug function
00214 */
00215 void malloc_stats(void);
00216
00217 /** \brief  Determine if it is safe to call malloc() in an IRQ context.
00218
00219     This function checks the value of the internal spinlock that is used for
00220     malloc() to ensure that a call to it will not freeze the running process.
00221     This is only really useful in an IRQ context to ensure that a call to
00222     malloc() (or some other memory allocation function) won't cause a deadlock.
00223
00224     \retval     1           If it is safe to call malloc() in the current IRQ.
00225     \retval     0           Otherwise.
00226 */
00227 int malloc_irq_safe(void);
00228
00229 /** \brief Only available with KM_DBG
00230 */
00231 int mem_check_block(void *p);
00232
00233 /** \brief Only available with KM_DBG
00234  */
00235 int mem_check_all(void);
00236
00237 __END_DECLS
00238
00239 #endif  /* __MALLOC_H */
```

9.96 include/netdb.h File Reference

Network address database functionality.

```
#include <sys/cdefs.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <inttypes.h>
```

Data Structures

- struct hostent

  Network host entry.
- struct addrinfo

  Network address information structure.

Macros

- #define h_addr h_addr_list[0]

  Primary network address.
- #define HOST_NOT_FOUND 1

  Hostname not found.
- #define TRY_AGAIN 2

  Try the request again.
- #define NO_RECOVERY 3

  A non-recoverable error.
- #define NO_DATA 4

  Host found, but no data.
- #define EAI_AGAIN 1

  Try the request again.
- #define EAI_BADFLAGS 2

  Invalid hint flags.
- #define EAI_FAIL 3

  A non-recoverable error.
- #define EAI_FAMILY 4

  Invalid address family.
- #define EAI_MEMORY 5

  Memory allocation error.
- #define EAI_NONAME 6

  Hostname not found.
- #define EAI_SERVICE 7

  Invalid service value.
- #define EAI_SOCKTYPE 8

  Invalid socket type.
- #define EAI_SYSTEM 9

  System error, check errno.
- #define EAI_OVERFLOW 10

  Argument buffer overflow.
- #define AI_PASSIVE 0x00000001

Address intended for bind().

- #define AI_CANONNAME 0x00000002

  Request canonical name.
- #define AI_NUMERICHOST 0x00000004

  Inhibit host resolution.
- #define AI_NUMERICSERV 0x00000008

  Inhibit service resolution.
- #define AI_V4MAPPED 0x00000010

  Return v4-mapped IPv6 addrs.
- #define AI_ALL 0x00000020

  Query for both IPv4 and IPv6.
- #define AI_ADDRCONFIG 0x00000040

  Only query for IPv4/IPv6 addrs the system has a valid addr.

Functions

- void freeaddrinfo (struct addrinfo *ai)

  Free an address information structure returned by getaddrinfo().
- int getaddrinfo (const char *nodename, const char *servname, const struct addrinfo *hints, struct addrinfo **res)

  Get information about a specified addresss.
- struct hostent * gethostbyname (const char *name)

  Look up a host by its name.
- struct hostent * gethostbyname2 (const char *name, int af)

  Look up a host by its name and address family.

Variables

- int h_errno

  Error value for gethostbyname().

9.96.1 Detailed Description

Network address database functionality.

This file contains functions related to network address lookups, usually performed through DNS.

Author

Lawrence Sebald

9.96.2 Macro Definition Documentation

h_addr

```
#define h_addr h_addr_list[0]
```

Primary network address.

9.96.3 Function Documentation

freeaddrinfo()

```
void freeaddrinfo (
            struct addrinfo * ai )
```

Free an address information structure returned by getaddrinfo().

This function cleans up any memory associated with the specified struct addrinfo, which was returned previously by a call to getaddrinfo().

Parameters

ai	The struct addrinfo to clean up.

getaddrinfo()

```
int getaddrinfo (
            const char * nodename,
            const char * servname,
            const struct addrinfo * hints,
            struct addrinfo ** res )
```

Get information about a specified addresss.

This function translates the name of a host and service into a set of socket addresses and related information to be used in creating a socket. This includes potentially looking up the host information in the network address database (and thus in DNS possibly as well).

Parameters

nodename	The host to look up.
servname	The service to look up.
hints	Hints used in aiding lookup.
res	The resulting address information.

Returns

0 on success, non-zero error code on failure.

See also

Errors for the getaddrinfo() function

gethostbyname()

```
struct hostent * gethostbyname (
            const char * name )
```

Look up a host by its name.

This function queries the network address database (possibly recursively) for the network address of the specified hostname. This will first search any local databases before querying remote databases (such as a DNS server) for the host specified.

Parameters

name	The hostname to look up.

Returns

A pointer to a host entry on success or NULL on failure. h_errno is set on failure to indicate the error that occurred.

Note

This function is non-reentrant. getaddrinfo() should (in general) be used instead of this function.

gethostbyname2()

```
struct hostent * gethostbyname2 (
            const char * name,
            int af )
```

Look up a host by its name and address family.

This function queries the network address database (possibly recursively) for the network address of the specified hostname. This will first search any local databases before querying remote databases (such as a DNS server) for the host specified.

This function allows you to specify the address family that you wish the returned hostent to contain. This function is a GNU extension and has not been specified by any POSIX specification.

Parameters

name	The hostname to look up.
af	The address family to use for lookups (AF_INET or AF_INET6).

Returns

A pointer to a host entry on success or NULL on failure. h_errno is set on failure to indicate the error that occurred.

Note

This function is non-reentrant. getaddrinfo() should (in general) be used instead of this function.

9.96.4 Variable Documentation

h_errno

```
int h_errno  [extern]
```

Error value for gethostbyname().

See also

Error values for the h_errno variable

9.97 netdb.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    netdb.h
00004    Copyright (C) 2014 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   netdb.h
00009     \brief  Network address database functionality.
00010
00011     This file contains functions related to network address lookups, usually
00012     performed through DNS.
00013
00014     \author Lawrence Sebald
00015 */
00016
00017 #ifndef __NETDB_H
00018 #define __NETDB_H
00019
00020 #include <sys/cdefs.h>
00021
00022 __BEGIN_DECLS
00023
00024 #include <netinet/in.h>
00025 #include <sys/socket.h>
00026 #include <inttypes.h>
00027
00028 /** \brief  Network host entry.
00029
00030     This structure describes a network host entry in the address database. When
00031     looking up an address with the gethostbyname() function, one of these will
00032     be returned with information about the host.
00033
00034     \headerfile netdb.h
00035 */
00036 struct hostent {
00037     char *h_name;                   /**< \brief Official name of the host. */
00038     char **h_aliases;               /**< \brief Alternative host names. */
00039     int h_addrtype;                 /**< \brief Address type. */
00040     int h_length;                   /**< \brief Length of address, in bytes. */
00041     char **h_addr_list;             /**< \brief Network addresses of host. */
00042 #define h_addr h_addr_list[0]       /**< \brief Primary network address. */
00043 };
00044
00045 /** \brief  Network address information structure.
00046
00047     This structure describes information on an address in the database. This
00048     structure is used by functions such as getaddrinfo() to return information
```

```
00049     about the specified host.
00050
00051     \headerfile netdb.h
00052 */
00053 struct addrinfo {
00054     int ai_flags;                   /**< \brief Input flags.
00055                                         \see   addrinfo_flags */
00056     int ai_family;                  /**< \brief Socket address family. */
00057     int ai_socktype;                /**< \brief Socket type. */
00058     int ai_protocol;                /**< \brief Socket protocol. */
00059     socklen_t ai_addrlen;           /**< \brief Address length. */
00060     struct sockaddr *ai_addr;       /**< \brief Address structure. */
00061     char *ai_canonname;             /**< \brief Canonical name. */
00062     struct addrinfo *ai_next;       /**< \brief Next address entry (if any). */
00063 };
00064
00065 /** \brief  Error value for gethostbyname().
00066     \see    herrno_vals */
00067 extern int h_errno;
00068
00069 /** \defgroup herrno_vals               Error values for the h_errno variable
00070
00071     These are the possible values for h_errno, indicating errors returns from
00072     the gethostbyname() function.
00073
00074     @{
00075 */
00076 #define HOST_NOT_FOUND      1       /**< \brief Hostname not found. */
00077 #define TRY_AGAIN           2       /**< \brief Try the request again. */
00078 #define NO_RECOVERY         3       /**< \brief A non-recoverable error. */
00079 #define NO_DATA             4       /**< \brief Host found, but no data. */
00080 /** @} */
00081
00082 /** \defgroup addrinfo_errors           Errors for the getaddrinfo() function
00083
00084     These are the possible error return values from the getaddrinfo() function.
00085
00086     @{
00087 */
00088 #define EAI_AGAIN           1       /**< \brief Try the request again. */
00089 #define EAI_BADFLAGS        2       /**< \brief Invalid hint flags. */
00090 #define EAI_FAIL            3       /**< \brief A non-recoverable error. */
00091 #define EAI_FAMILY          4       /**< \brief Invalid address family. */
00092 #define EAI_MEMORY          5       /**< \brief Memory allocation error. */
00093 #define EAI_NONAME          6       /**< \brief Hostname not found. */
00094 #define EAI_SERVICE         7       /**< \brief Invalid service value. */
00095 #define EAI_SOCKTYPE        8       /**< \brief Invalid socket type. */
00096 #define EAI_SYSTEM          9       /**< \brief System error, check errno. */
00097 #define EAI_OVERFLOW        10      /**< \brief Argument buffer overflow. */
00098 /** @} */
00099
00100 /** \defgroup addrinfo_flags            Flags for ai_flags in struct addrinfo
00101
00102     These are the flags that can be set in the ai_flags field of struct
00103     addrinfo. These values can be bitwise ORed together.
00104
00105     Currently only AI_PASSIVE is actually supported by the getaddrinfo()
00106     function.
00107
00108     @{
00109 */
00110 #define AI_PASSIVE      0x00000001  /**< \brief Address intended for bind(). */
00111 #define AI_CANONNAME    0x00000002  /**< \brief Request canonical name. */
00112 #define AI_NUMERICHOST  0x00000004  /**< \brief Inhibit host resolution. */
00113 #define AI_NUMERICSERV  0x00000008  /**< \brief Inhibit service resolution. */
00114 #define AI_V4MAPPED     0x00000010  /**< \brief Return v4-mapped IPv6 addrs. */
00115 #define AI_ALL          0x00000020  /**< \brief Query for both IPv4 and IPv6. */
00116 #define AI_ADDRCONFIG   0x00000040  /**< \brief Only query for IPv4/IPv6 addrs
00117                                             the system has a valid addr. */
00118 /** @} */
00119
00120 /** \brief  Free an address information structure returned by getaddrinfo().
00121
00122     This function cleans up any memory associated with the specified
00123     struct addrinfo, which was returned previously by a call to getaddrinfo().
00124
00125     \param  ai              The struct addrinfo to clean up.
00126 */
00127 void freeaddrinfo(struct addrinfo *ai);
00128
00129 /** \brief  Get information about a specified addresss.
```

```
00130
00131     This function translates the name of a host and service into a set of socket
00132     addresses and related information to be used in creating a socket. This
00133     includes potentially looking up the host information in the network address
00134     database (and thus in DNS possibly as well).
00135
00136     \param  nodename        The host to look up.
00137     \param  servname        The service to look up.
00138     \param  hints           Hints used in aiding lookup.
00139     \param  res             The resulting address information.
00140     \return                 0 on success, non-zero error code on failure.
00141     \see    addrinfo_errors
00142 */
00143 int getaddrinfo(const char *nodename, const char *servname,
00144                 const struct addrinfo *hints, struct addrinfo **res);
00145
00146 /** \brief  Look up a host by its name.
00147
00148     This function queries the network address database (possibly recursively)
00149     for the network address of the specified hostname. This will first search
00150     any local databases before querying remote databases (such as a DNS server)
00151     for the host specified.
00152
00153     \param  name            The hostname to look up.
00154     \return                 A pointer to a host entry on success or NULL on
00155                             failure. h_errno is set on failure to indicate the
00156                             error that occurred.
00157
00158     \note                   This function is non-reentrant. getaddrinfo() should
00159                             (in general) be used instead of this function.
00160 */
00161 struct hostent *gethostbyname(const char *name);
00162
00163 /** \brief  Look up a host by its name and address family.
00164
00165     This function queries the network address database (possibly recursively)
00166     for the network address of the specified hostname. This will first search
00167     any local databases before querying remote databases (such as a DNS server)
00168     for the host specified.
00169
00170     This function allows you to specify the address family that you wish the
00171     returned hostent to contain. This function is a GNU extension and has not
00172     been specified by any POSIX specification.
00173
00174     \param  name            The hostname to look up.
00175     \param  af              The address family to use for lookups (AF_INET or
00176                             AF_INET6).
00177     \return                 A pointer to a host entry on success or NULL on
00178                             failure. h_errno is set on failure to indicate the
00179                             error that occurred.
00180
00181     \note                   This function is non-reentrant. getaddrinfo() should
00182                             (in general) be used instead of this function.
00183 */
00184 struct hostent *gethostbyname2(const char *name, int af);
00185
00186 __END_DECLS
00187
00188 #endif /* !__NETDB_H */
```

9.98 include/netinet/in.h File Reference

Definitions for the Internet address family.

```
#include <sys/cdefs.h>
#include <inttypes.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```

Data Structures

- struct in_addr

  Structure used to store an IPv4 address.
- struct in6_addr

  Structure used to store an IPv6 address.
- struct sockaddr_in

  Structure used to store an IPv4 address for a socket.
- struct sockaddr_in6

  Structure used to store an IPv6 address for a socket.

Macros

- #define s6_addr __s6_addr.__s6_addr8
- #define INADDR_ANY 0x00000000

  Local IPv4 host address.
- #define INADDR_BROADCAST 0xFFFFFFFF

  IPv4 broadcast address.
- #define INADDR_NONE 0xFFFFFFFF

  IPv4 error address.
- #define IN6ADDR_ANY_INIT

  Initialize an IPv6 local host address.
- #define IN6ADDR_LOOPBACK_INIT

  Initialize an IPv6 loopback address.
- #define INET_ADDRSTRLEN 16

  Length of a string form of a maximal IPv4 address.
- #define INET6_ADDRSTRLEN 46

  Length of a string form of a maximal IPv6 address.
- #define IPPROTO_IP 0

  Internet Protocol Version 4.
- #define IPPROTO_ICMP 1

  Internet Control Message Protocol.
- #define IPPROTO_TCP 6

  Transmission Control Protocol.
- #define IPPROTO_UDP 17

  User Datagram Protocol.
- #define IPPROTO_IPV6 41

  Internet Protocol Version 6.
- #define IPPROTO_UDPLITE 136

  Lightweight User Datagram Protocol.
- #define IP_TTL 24

  TTL for unicast (get/set)
- #define IPV6_JOIN_GROUP 17

  Join a multicast group (set)
- #define IPV6_LEAVE_GROUP 18

  Leave a multicast group (set)
- #define IPV6_MULTICAST_HOPS 19

Hop limit for multicast (get/set)

- #define IPV6_MULTICAST_IF 20

  Multicast interface (get/set)
- #define IPV6_MULTICAST_LOOP 21

  Multicasts loopback (get/set)
- #define IPV6_UNICAST_HOPS 22

  Hop limit for unicast (get/set)
- #define IPV6_V6ONLY 23

  IPv6 only – no IPv4 (get/set)
- #define IN6_IS_ADDR_UNSPECIFIED(a)

  Test if an IPv6 Address is unspecified.
- #define IN6_IS_ADDR_LOOPBACK(a)

  Test if an IPv6 Address is a loopback address.
- #define IN6_IS_ADDR_V4MAPPED(a)

  Test if an IPv6 Address is an IPv4 mapped address.
- #define IN6_IS_ADDR_V4COMPAT(a)

  Test if an IPv6 Address is an IPv4 compatibility address.
- #define IN6_IS_ADDR_LINKLOCAL(a)

  Test if an IPv6 Address is a link-local address.
- #define IN6_IS_ADDR_SITELOCAL(a)

  Test if an IPv6 Address is a site-local address.
- #define IN6_IS_ADDR_MULTICAST(a) ((a)->__s6_addr.__s6_addr8[0] == 0xFF)

  Test if an IPv6 Address is a multicast address.
- #define IN6_IS_ADDR_MC_NODELOCAL(a)

  Test if an IPv6 Address is a node-local multicast address.
- #define IN6_IS_ADDR_MC_LINKLOCAL(a)

  Test if an IPv6 Address is a link-local multicast address.
- #define IN6_IS_ADDR_MC_SITELOCAL(a)

  Test if an IPv6 Address is a site-local multicast address.
- #define IN6_IS_ADDR_MC_ORGLOCAL(a)

  Test if an IPv6 Address is an organization-local multicast address.
- #define IN6_IS_ADDR_MC_GLOBAL(a)

  Test if an IPv6 Address is a global multicast address.

Typedefs

- typedef uint16_t in_port_t

  16-bit type used to store a value for an internet port.
- typedef uint32_t in_addr_t

  32-bit value used to store an IPv4 address.

Variables

- const struct in6_addr in6addr_any

  IPv6 local host address.
- const struct in6_addr in6addr_loopback

  IPv6 loopback address.

9.98.1 Detailed Description

Definitions for the Internet address family.

This file contains the standard definitions (as directed by the POSIX 2008 standard) for internet-related functionality in the AF_INET address family. This does is not guaranteed to have everything that one might have in a fully-standard compliant implementation of the POSIX standard.

Author

Lawrence Sebald

9.98.2 Macro Definition Documentation

IN6_IS_ADDR_LINKLOCAL

```
#define IN6_IS_ADDR_LINKLOCAL(
              a )
```

Value:

```
    (((a)->__s6_addr.__s6_addr8[0] == 0xFE) && \
     (((a)->__s6_addr.__s6_addr8[1] & 0xC0) == 0x80))
```

Test if an IPv6 Address is a link-local address.

This macro tests whether an IPv6 address (struct in6_addr *) is a link-local address.

Parameters

a	The address to test (struct in6_addr *)

Returns

Nonzero if the address is link-local, 0 otherwise.

IN6_IS_ADDR_LOOPBACK

```
#define IN6_IS_ADDR_LOOPBACK(
              a )
```

Value:

```
    ((a)->__s6_addr.__s6_addr32[0] == 0 && \
     (a)->__s6_addr.__s6_addr32[1] == 0 && \
     (a)->__s6_addr.__s6_addr32[2] == 0 && \
     (a)->__s6_addr.__s6_addr16[6] == 0 && \
     (a)->__s6_addr.__s6_addr8[14] == 0 && \
     (a)->__s6_addr.__s6_addr8[15] == 1)
```

Test if an IPv6 Address is a loopback address.

This macro tests whether an IPv6 address (struct in6_addr *) is a loopback address.

Parameters

a	The address to test (struct in6_addr *)

Returns

Nonzero if the address is a loopback, 0 otherwise.

IN6_IS_ADDR_MC_GLOBAL

```
#define IN6_IS_ADDR_MC_GLOBAL(
              a )
```

Value:

```
(IN6_IS_ADDR_MULTICAST(a) && \
 (((a)->__s6_addr.__s6_addr8[1] & 0x0F) == 0x0E))
```

Test if an IPv6 Address is a global multicast address.

This macro tests whether an IPv6 address (struct in6_addr *) is a global multicast address.

Parameters

a	The address to test (struct in6_addr *)

Returns

Nonzero if the address is a global multicast address, 0 otherwise.

IN6_IS_ADDR_MC_LINKLOCAL

```
#define IN6_IS_ADDR_MC_LINKLOCAL(
              a )
```

Value:

```
(IN6_IS_ADDR_MULTICAST(a) && \
 (((a)->__s6_addr.__s6_addr8[1] & 0x0F) == 0x02))
```

Test if an IPv6 Address is a link-local multicast address.

This macro tests whether an IPv6 address (struct in6_addr *) is a link-local multicast address.

Parameters

a	The address to test (struct in6_addr *)

Returns

Nonzero if the address is a link-local multicast address, 0 otherwise.

IN6_IS_ADDR_MC_NODELOCAL

```
#define IN6_IS_ADDR_MC_NODELOCAL(
              a )
```

Value:

```
(IN6_IS_ADDR_MULTICAST(a) && \
 (((a)->__s6_addr.__s6_addr8[1] & 0x0F) == 0x01))
```

Test if an IPv6 Address is a node-local multicast address.

This macro tests whether an IPv6 address (struct in6_addr *) is a node-local multicast address.

Parameters

a	The address to test (struct in6_addr *)

Returns

Nonzero if the address is a node-local multicast address, 0 otherwise.

IN6_IS_ADDR_MC_ORGLOCAL

```
#define IN6_IS_ADDR_MC_ORGLOCAL(
              a )
```

Value:

```
(IN6_IS_ADDR_MULTICAST(a) && \
 (((a)->__s6_addr.__s6_addr8[1] & 0x0F) == 0x08))
```

Test if an IPv6 Address is an organization-local multicast address.

This macro tests whether an IPv6 address (struct in6_addr *) is an organization-local multicast address.

Parameters

a	The address to test (struct in6_addr *)

Returns

Nonzero if the address is an organization-local multicast address, 0 otherwise.

IN6_IS_ADDR_MC_SITELOCAL

```
#define IN6_IS_ADDR_MC_SITELOCAL(
              a )
```

Value:

```
    (IN6_IS_ADDR_MULTICAST(a) && \
     (((a)->__s6_addr.__s6_addr8[1] & 0x0F) == 0x05))
```

Test if an IPv6 Address is a site-local multicast address.

This macro tests whether an IPv6 address (struct in6_addr *) is a site-local multicast address.

Parameters

a	The address to test (struct in6_addr *)

Returns

Nonzero if the address is a site-local multicast address, 0 otherwise.

IN6_IS_ADDR_MULTICAST

```
#define IN6_IS_ADDR_MULTICAST(
              a )  ((a)->__s6_addr.__s6_addr8[0] == 0xFF)
```

Test if an IPv6 Address is a multicast address.

This macro tests whether an IPv6 address (struct in6_addr *) is a multicast address.

Parameters

a	The address to test (struct in6_addr *)

Returns

Nonzero if the address is multicast, 0 otherwise.

IN6_IS_ADDR_SITELOCAL

```
#define IN6_IS_ADDR_SITELOCAL(
              a )
```

Value:

```
    (((a)->__s6_addr.__s6_addr8[0] == 0xFE) && \
     (((a)->__s6_addr.__s6_addr8[1] & 0xC0) == 0xC0))
```

Test if an IPv6 Address is a site-local address.

This macro tests whether an IPv6 address (struct in6_addr *) is a site-local address.

Parameters

a	The address to test (struct in6_addr ∗)

Returns

Nonzero if the address is site-local, 0 otherwise.

IN6_IS_ADDR_UNSPECIFIED

```
#define IN6_IS_ADDR_UNSPECIFIED(
              a )
```

Value:

```
    ((a)->__s6_addr.__s6_addr32[0] == 0 && \
     (a)->__s6_addr.__s6_addr32[1] == 0 && \
     (a)->__s6_addr.__s6_addr32[2] == 0 && \
     (a)->__s6_addr.__s6_addr32[3] == 0)
```

Test if an IPv6 Address is unspecified.

This macro tests whether an IPv6 address (struct in6_addr ∗) is an unspecified address.

Parameters

a	The address to test (struct in6_addr ∗)

Returns

Nonzero if the address is unspecified, 0 otherwise.

IN6_IS_ADDR_V4COMPAT

```
#define IN6_IS_ADDR_V4COMPAT(
              a )
```

Value:

```
    ((a)->__s6_addr.__s6_addr32[0] == 0 && \
     (a)->__s6_addr.__s6_addr32[1] == 0 && \
     (a)->__s6_addr.__s6_addr32[2] == 0 && \
     (a)->__s6_addr.__s6_addr32[3] != 0 && \
     (a)->__s6_addr.__s6_addr8[15] != 1)
```

Test if an IPv6 Address is an IPv4 compatibility address.

This macro tests whether an IPv6 address (struct in6_addr ∗) is an IPv4 compatibility address.

Parameters

a	The address to test (struct in6_addr ∗)

Returns

Nonzero if the address is IPv4 compat, 0 otherwise.

IN6_IS_ADDR_V4MAPPED

```
#define IN6_IS_ADDR_V4MAPPED(
              a )
```

Value:

```
    ((a)->__s6_addr.__s6_addr32[0] == 0 &&      \
     (a)->__s6_addr.__s6_addr32[1] == 0 &&      \
     (a)->__s6_addr.__s6_addr16[4] == 0 &&      \
     (a)->__s6_addr.__s6_addr16[5] == 0xFFFF)
```

Test if an IPv6 Address is an IPv4 mapped address.

This macro tests whether an IPv6 address (struct in6_addr ∗) is an IPv4 mapped address.

Parameters

a	The address to test (struct in6_addr ∗)

Returns

Nonzero if the address is IPv4 mapped, 0 otherwise.

IN6ADDR_ANY_INIT

```
#define IN6ADDR_ANY_INIT
```

Value:

```
                                {{{ 0, 0, 0, 0, 0, 0, 0, 0, \
0, 0, 0, 0, 0, 0, 0, 0 }}}
```

Initialize an IPv6 local host address.

This macro can be used to initialize a struct in6_addr to any local address. It functions similarly to INADDR_ANY for IPv4.

IN6ADDR_LOOPBACK_INIT

```
#define IN6ADDR_LOOPBACK_INIT
```

Value:

```
{{{ 0, 0, 0, 0, 0, 0, 0, 0, \
0, 0, 0, 0, 0, 0, 0, 1 }}}
```

Initialize an IPv6 loopback address.

This macro can be used to initialize a struct in6_addr to the loopback address.

INADDR_ANY

```
#define INADDR_ANY 0x00000000
```

Local IPv4 host address.

This address can be used by many things if you prefer to not specify the local address, and would rather it be detected automatically.

INADDR_BROADCAST

```
#define INADDR_BROADCAST 0xFFFFFFFF
```

IPv4 broadcast address.

This address is the normal IPv4 broadcast address (255.255.255.255).

INADDR_NONE

```
#define INADDR_NONE 0xFFFFFFFF
```

IPv4 error address.

This address is non-standard, but is available on many systems. It is used to detect failure from some functions that normally return addresses (such as the inet_addr function).

INET6_ADDRSTRLEN

```
#define INET6_ADDRSTRLEN 46
```

Length of a string form of a maximal IPv6 address.

INET_ADDRSTRLEN

```
#define INET_ADDRSTRLEN 16
```

Length of a string form of a maximal IPv4 address.

IPPROTO_ICMP

```
#define IPPROTO_ICMP 1
```

Internet Control Message Protocol.

IPPROTO_IP

```
#define IPPROTO_IP 0
```

Internet Protocol Version 4.

IPPROTO_IPV6

```
#define IPPROTO_IPV6 41
```

Internet Protocol Version 6.

IPPROTO_TCP

```
#define IPPROTO_TCP 6
```

Transmission Control Protocol.

IPPROTO_UDP

```
#define IPPROTO_UDP 17
```

User Datagram Protocol.

IPPROTO_UDPLITE

```
#define IPPROTO_UDPLITE 136
```

Lightweight User Datagram Protocol.

s6_addr

```
#define s6_addr __s6_addr.__s6_addr8
```

9.98.3 Typedef Documentation

in_addr_t

```
typedef uint32_t in_addr_t
```

32-bit value used to store an IPv4 address.

in_port_t

```
typedef uint16_t in_port_t
```

16-bit type used to store a value for an internet port.

9.98.4 Variable Documentation

in6addr_any

```
const struct in6_addr in6addr_any  [extern]
```

IPv6 local host address.

This constant variable contains the IPv6 local host address.

in6addr_loopback

```
const struct in6_addr in6addr_loopback  [extern]
```

IPv6 loopback address.

This constant variable contains the IPv6 loopback address.

9.99 in.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    netinet/in.h
00004    Copyright (C) 2006, 2007, 2010 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   netinet/in.h
00009     \brief  Definitions for the Internet address family.
00010
00011     This file contains the standard definitions (as directed by the POSIX 2008
00012     standard) for internet-related functionality in the AF_INET address family.
00013     This does is not guaranteed to have everything that one might have in a
00014     fully-standard compliant implementation of the POSIX standard.
00015
00016     \author Lawrence Sebald
00017 */
00018
00019 #ifndef __NETINET_IN_H
00020 #define __NETINET_IN_H
00021
00022 #include <sys/cdefs.h>
00023
00024 __BEGIN_DECLS
00025
00026 /* Bring in <inttypes.h> to grab the uint16_t and uint32_t types (for in_port_t
00027    and in_addr_t below), along with uint8_t. Bring in <sys/socket.h> for the
00028    sa_family_t type, as required. IEEE Std 1003.1-2008 specifically states that
00029    <netinet/in.h> can make visible all symbols from both <inttypes.h> and
00030    <sys/socket.h>. */
00031 #include <inttypes.h>
00032 #include <sys/socket.h>
00033
00034 /** \brief  16-bit type used to store a value for an internet port. */
00035 typedef uint16_t in_port_t;
00036
00037 /** \brief  32-bit value used to store an IPv4 address. */
00038 typedef uint32_t in_addr_t;
00039
00040 /** \brief  Structure used to store an IPv4 address.
00041     \headerfile netinet/in.h
00042 */
00043 struct in_addr {
00044     in_addr_t s_addr;
00045 };
00046
00047 /** \brief  Structure used to store an IPv6 address.
00048     \headerfile netinet/in.h
00049 */
00050 struct in6_addr {
00051     union {
00052         uint8_t  __s6_addr8[16];
00053         uint16_t __s6_addr16[8];
00054         uint32_t __s6_addr32[4];
00055         uint64_t __s6_addr64[2];
00056     } __s6_addr;
00057 #define s6_addr __s6_addr.__s6_addr8
00058 };
00059
00060 /* Bring in <arpa/inet.h> to make ntohl/ntohs/htonl/htons visible, as per IEEE
00061    Std 1003.1-2008 (the standard specifically states that <netinet/in.h> may
00062    make all symbols from <arpa/inet.h> visible. The <arpa/inet.h> header
00063    actually needs the stuff above, so that's why we include it here. */
00064 #include <arpa/inet.h>
00065
00066 /** \brief  Structure used to store an IPv4 address for a socket.
00067
00068     This structure is the standard way to set up addresses for sockets in the
00069     AF_INET address family. Generally you will not send one of these directly
00070     to a function, but rather will cast it to a struct sockaddr. Also, this
00071     structure contains the old sin_zero member which is no longer required by
00072     the standard (for compatibility with applications that expect it).
00073
00074     \headerfile netinet/in.h
00075 */
00076 struct sockaddr_in {
```

```
00077     /** \brief  Family for the socket. Must be AF_INET. */
00078     sa_family_t    sin_family;
00079
00080     /** \brief  Port for the socket. Must be in network byte order. */
00081     in_port_t      sin_port;
00082
00083     /** \brief  Address for the socket. Must be in network byte order. */
00084     struct in_addr sin_addr;
00085
00086     /** \brief  Empty space, ignored for all intents and purposes. */
00087     unsigned char  sin_zero[8];
00088 };
00089
00090 /** \brief  Structure used to store an IPv6 address for a socket.
00091
00092     This structure is the standard way to set up addresses for sockets in the
00093     AF_INET6 address family. Generally you will not send one of these directly
00094     to a function, but rather will cast it to a struct sockaddr.
00095
00096     \headerfile netinet/in.h
00097  */
00098 struct sockaddr_in6 {
00099     /** \brief  Family for the socket. Must be AF_INET6. */
00100     sa_family_t     sin6_family;
00101
00102     /** \brief  Port for the socket. Must be in network byte order. */
00103     in_port_t       sin6_port;
00104
00105     /** \brief  Traffic class and flow information. */
00106     uint32_t        sin6_flowinfo;
00107
00108     /** \brief  Address for the socket. Must be in network byte order. */
00109     struct in6_addr sin6_addr;
00110
00111     /** \brief  Set of interfaces for a scope. */
00112     uint32_t        sin6_scope_id;
00113 };
00114
00115 /** \brief  Local IPv4 host address.
00116
00117     This address can be used by many things if you prefer to not specify the
00118     local address, and would rather it be detected automatically.
00119 */
00120 #define INADDR_ANY       0x00000000
00121
00122 /** \brief  IPv4 broadcast address.
00123
00124     This address is the normal IPv4 broadcast address (255.255.255.255).
00125 */
00126 #define INADDR_BROADCAST 0xFFFFFFFF
00127
00128 /** \brief  IPv4 error address.
00129
00130     This address is non-standard, but is available on many systems. It is used
00131     to detect failure from some functions that normally return addresses (such
00132     as the inet_addr function).
00133 */
00134 #define INADDR_NONE      0xFFFFFFFF
00135
00136 /** \brief  Initialize an IPv6 local host address.
00137
00138     This macro can be used to initialize a struct in6_addr to any local address.
00139     It functions similarly to INADDR_ANY for IPv4.
00140 */
00141 #define IN6ADDR_ANY_INIT {{{ 0, 0, 0, 0, 0, 0, 0, 0, \
00142                              0, 0, 0, 0, 0, 0, 0, 0 }}}
00143
00144 /** \brief  Initialize an IPv6 loopback address.
00145
00146     This macro can be used to initialize a struct in6_addr to the loopback
00147     address.
00148 */
00149 #define IN6ADDR_LOOPBACK_INIT {{{ 0, 0, 0, 0, 0, 0, 0, 0, \
00150                                   0, 0, 0, 0, 0, 0, 0, 1 }}}
00151
00152 /** \brief  IPv6 local host address.
00153
00154     This constant variable contains the IPv6 local host address.
00155 */
00156 extern const struct in6_addr in6addr_any;
00157
```

```
00158 /** \brief  IPv6 loopback address.
00159
00160     This constant variable contains the IPv6 loopback address.
00161 */
00162 extern const struct in6_addr in6addr_loopback;
00163
00164 /** \brief  Length of a string form of a maximal IPv4 address. */
00165 #define INET_ADDRSTRLEN 16
00166
00167 /** \brief  Length of a string form of a maximal IPv6 address. */
00168 #define INET6_ADDRSTRLEN 46
00169
00170 /** \brief  Internet Protocol Version 4. */
00171 #define IPPROTO_IP      0
00172
00173 /** \brief  Internet Control Message Protocol. */
00174 #define IPPROTO_ICMP    1
00175
00176 /** \brief  Transmission Control Protocol. */
00177 #define IPPROTO_TCP     6
00178
00179 /** \brief  User Datagram Protocol. */
00180 #define IPPROTO_UDP     17
00181
00182 /** \brief  Internet Protocol Version 6. */
00183 #define IPPROTO_IPV6    41
00184
00185 /** \brief  Lightweight User Datagram Protocol. */
00186 #define IPPROTO_UDPLITE 136
00187
00188 /** \defgroup ipv4_opts                 IPv4 protocol level options
00189
00190     These are the various socket-level optoins that can be accessed with the
00191     setsockopt() and getsockopt() fnctions for the IPPROTO_IP level value.
00192
00193     As there isn't really a full standard list of these defined in the SUS
00194     (apparently), only ones that we support are listed here.
00195
00196     \see                so_opts
00197     \see                ipv6_opts
00198     \see                udp_opts
00199     \see                udplite_opts
00200     \see                tcp_opts
00201
00202     @{
00203 */
00204
00205 #define IP_TTL              24  /**< \brief TTL for unicast (get/set) */
00206
00207 /** @} */
00208
00209 /** \defgroup ipv6_opts                 IPv6 protocol level options
00210
00211     These are the various socket-level options that can be accessed with the
00212     setsockopt() and getsockopt() functions for the IPPROTO_IPV6 level value.
00213
00214     Not all of these are currently supported, but they are listed for
00215     completeness.
00216
00217     \see                so_opts
00218     \see                ipv4_opts
00219     \see                udp_opts
00220     \see                udplite_opts
00221     \see                tcp_opts
00222
00223     @{
00224 */
00225
00226 #define IPV6_JOIN_GROUP     17  /**< \brief Join a multicast group (set) */
00227 #define IPV6_LEAVE_GROUP    18  /**< \brief Leave a multicast group (set) */
00228 #define IPV6_MULTICAST_HOPS 19  /**< \brief Hop limit for multicast (get/set) */
00229 #define IPV6_MULTICAST_IF   20  /**< \brief Multicast interface (get/set) */
00230 #define IPV6_MULTICAST_LOOP 21  /**< \brief Multicasts loopback (get/set) */
00231 #define IPV6_UNICAST_HOPS   22  /**< \brief Hop limit for unicast (get/set) */
00232 #define IPV6_V6ONLY         23  /**< \brief IPv6 only -- no IPv4 (get/set) */
00233
00234 /** @} */
00235
00236 /** \brief  Test if an IPv6 Address is unspecified.
00237
00238     This macro tests whether an IPv6 address (struct in6_addr *) is an
```

```
00239     unspecified address.
00240
00241     \param  a           The address to test (struct in6_addr *)
00242     \return             Nonzero if the address is unspecified, 0 otherwise.
00243 */
00244 #define IN6_IS_ADDR_UNSPECIFIED(a)  \
00245     ((a)->__s6_addr.__s6_addr32[0] == 0 && \
00246      (a)->__s6_addr.__s6_addr32[1] == 0 && \
00247      (a)->__s6_addr.__s6_addr32[2] == 0 && \
00248      (a)->__s6_addr.__s6_addr32[3] == 0)
00249
00250 /** \brief  Test if an IPv6 Address is a loopback address.
00251
00252     This macro tests whether an IPv6 address (struct in6_addr *) is a
00253     loopback address.
00254
00255     \param  a           The address to test (struct in6_addr *)
00256     \return             Nonzero if the address is a loopback, 0 otherwise.
00257 */
00258 #define IN6_IS_ADDR_LOOPBACK(a)  \
00259     ((a)->__s6_addr.__s6_addr32[0] == 0 && \
00260      (a)->__s6_addr.__s6_addr32[1] == 0 && \
00261      (a)->__s6_addr.__s6_addr32[2] == 0 && \
00262      (a)->__s6_addr.__s6_addr16[6] == 0 && \
00263      (a)->__s6_addr.__s6_addr8[14] == 0 && \
00264      (a)->__s6_addr.__s6_addr8[15] == 1)
00265
00266 /** \brief  Test if an IPv6 Address is an IPv4 mapped address.
00267
00268     This macro tests whether an IPv6 address (struct in6_addr *) is an IPv4
00269     mapped address.
00270
00271     \param  a           The address to test (struct in6_addr *)
00272     \return             Nonzero if the address is IPv4 mapped, 0 otherwise.
00273 */
00274 #define IN6_IS_ADDR_V4MAPPED(a)  \
00275     ((a)->__s6_addr.__s6_addr32[0] == 0 &&      \
00276      (a)->__s6_addr.__s6_addr32[1] == 0 &&      \
00277      (a)->__s6_addr.__s6_addr16[4] == 0 &&      \
00278      (a)->__s6_addr.__s6_addr16[5] == 0xFFFF)
00279
00280 /** \brief  Test if an IPv6 Address is an IPv4 compatibility address.
00281
00282     This macro tests whether an IPv6 address (struct in6_addr *) is an IPv4
00283     compatibility address.
00284
00285     \param  a           The address to test (struct in6_addr *)
00286     \return             Nonzero if the address is IPv4 compat, 0 otherwise.
00287 */
00288 #define IN6_IS_ADDR_V4COMPAT(a)  \
00289     ((a)->__s6_addr.__s6_addr32[0] == 0 && \
00290      (a)->__s6_addr.__s6_addr32[1] == 0 && \
00291      (a)->__s6_addr.__s6_addr32[2] == 0 && \
00292      (a)->__s6_addr.__s6_addr32[3] != 0 && \
00293      (a)->__s6_addr.__s6_addr8[15] != 1)
00294
00295 /** \brief  Test if an IPv6 Address is a link-local address.
00296
00297     This macro tests whether an IPv6 address (struct in6_addr *) is a link-local
00298     address.
00299
00300     \param  a           The address to test (struct in6_addr *)
00301     \return             Nonzero if the address is link-local, 0 otherwise.
00302 */
00303 #define IN6_IS_ADDR_LINKLOCAL(a)  \
00304     (((a)->__s6_addr.__s6_addr8[0] == 0xFE) && \
00305      (((a)->__s6_addr.__s6_addr8[1] & 0xC0) == 0x80))
00306
00307 /** \brief  Test if an IPv6 Address is a site-local address.
00308
00309     This macro tests whether an IPv6 address (struct in6_addr *) is a site-local
00310     address.
00311
00312     \param  a           The address to test (struct in6_addr *)
00313     \return             Nonzero if the address is site-local, 0 otherwise.
00314 */
00315 #define IN6_IS_ADDR_SITELOCAL(a)  \
00316     (((a)->__s6_addr.__s6_addr8[0] == 0xFE) && \
00317      (((a)->__s6_addr.__s6_addr8[1] & 0xC0) == 0xC0))
00318
00319 /** \brief  Test if an IPv6 Address is a multicast address.
```

```
00320
00321     This macro tests whether an IPv6 address (struct in6_addr *) is a multicast
00322     address.
00323
00324     \param  a           The address to test (struct in6_addr *)
00325     \return             Nonzero if the address is multicast, 0 otherwise.
00326 */
00327 #define IN6_IS_ADDR_MULTICAST(a)  \
00328     ((a)->__s6_addr.__s6_addr8[0] == 0xFF)
00329
00330 /** \brief  Test if an IPv6 Address is a node-local multicast address.
00331
00332     This macro tests whether an IPv6 address (struct in6_addr *) is a node-local
00333     multicast address.
00334
00335     \param  a           The address to test (struct in6_addr *)
00336     \return             Nonzero if the address is a node-local multicast
00337                         address, 0 otherwise.
00338 */
00339 #define IN6_IS_ADDR_MC_NODELOCAL(a)  \
00340     (IN6_IS_ADDR_MULTICAST(a) && \
00341      (((a)->__s6_addr.__s6_addr8[1] & 0x0F) == 0x01))
00342
00343 /** \brief  Test if an IPv6 Address is a link-local multicast address.
00344
00345     This macro tests whether an IPv6 address (struct in6_addr *) is a link-local
00346     multicast address.
00347
00348     \param  a           The address to test (struct in6_addr *)
00349     \return             Nonzero if the address is a link-local multicast
00350                         address, 0 otherwise.
00351 */
00352 #define IN6_IS_ADDR_MC_LINKLOCAL(a)  \
00353     (IN6_IS_ADDR_MULTICAST(a) && \
00354      (((a)->__s6_addr.__s6_addr8[1] & 0x0F) == 0x02))
00355
00356 /** \brief  Test if an IPv6 Address is a site-local multicast address.
00357
00358     This macro tests whether an IPv6 address (struct in6_addr *) is a site-local
00359     multicast address.
00360
00361     \param  a           The address to test (struct in6_addr *)
00362     \return             Nonzero if the address is a site-local multicast
00363                         address, 0 otherwise.
00364 */
00365 #define IN6_IS_ADDR_MC_SITELOCAL(a)  \
00366     (IN6_IS_ADDR_MULTICAST(a) && \
00367      (((a)->__s6_addr.__s6_addr8[1] & 0x0F) == 0x05))
00368
00369 /** \brief  Test if an IPv6 Address is an organization-local multicast address.
00370
00371     This macro tests whether an IPv6 address (struct in6_addr *) is an
00372     organization-local multicast address.
00373
00374     \param  a           The address to test (struct in6_addr *)
00375     \return             Nonzero if the address is an organization-local
00376                         multicast address, 0 otherwise.
00377 */
00378 #define IN6_IS_ADDR_MC_ORGLOCAL(a)  \
00379     (IN6_IS_ADDR_MULTICAST(a) && \
00380      (((a)->__s6_addr.__s6_addr8[1] & 0x0F) == 0x08))
00381
00382 /** \brief  Test if an IPv6 Address is a global multicast address.
00383
00384     This macro tests whether an IPv6 address (struct in6_addr *) is a global
00385     multicast address.
00386
00387     \param  a           The address to test (struct in6_addr *)
00388     \return             Nonzero if the address is a global multicast address,
00389                         0 otherwise.
00390 */
00391 #define IN6_IS_ADDR_MC_GLOBAL(a)  \
00392     (IN6_IS_ADDR_MULTICAST(a) && \
00393      (((a)->__s6_addr.__s6_addr8[1] & 0x0F) == 0x0E))
00394
00395 __END_DECLS
00396
00397 #endif /* __NETINET_IN_H */
```

9.100 include/netinet/tcp.h File Reference

Definitions for the Transmission Control Protocol.

```
#include <sys/cdefs.h>
```

Macros

- #define TCP_NODELAY 1

  Don't delay to coalesce.

9.100.1 Detailed Description

Definitions for the Transmission Control Protocol.

This file contains the standard definitions (as directed by the Open Group Base Specifications Issue 7, 2018 Edition aka POSIX 2017) for functionality of the Transmission Control Protocol. This does is not guaranteed to have everything that one might have in a fully-standard compliant implementation of the POSIX standard.

Author

Lawrence Sebald

9.101 tcp.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    netinet/tcp.h
00004    Copyright (C) 2023 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   netinet/tcp.h
00009     \brief  Definitions for the Transmission Control Protocol.
00010
00011     This file contains the standard definitions (as directed by the Open Group
00012     Base Specifications Issue 7, 2018 Edition aka POSIX 2017) for  functionality
00013     of the Transmission Control Protocol.
00014     This does is not guaranteed to have everything that one might have in a
00015     fully-standard compliant implementation of the POSIX standard.
00016
00017     \author Lawrence Sebald
00018 */
00019
00020 #ifndef __NETINET_TCP_H
00021 #define __NETINET_TCP_H
00022
00023 #include <sys/cdefs.h>
00024
00025 __BEGIN_DECLS
00026
00027 /** \defgroup tcp_opts                  TCP protocol level options
00028
00029     These are the various socket-level optoins that can be accessed with the
00030     setsockopt() and getsockopt() fnctions for the IPPROTO_TCP level value.
00031
00032     All options listed here are at least guaranteed to be accepted by
00033     setsockopt() and getsockopt() for IPPROTO_TCP, however they are not
```

```
00034     guaranteed to be implemented in any meaningful way.
00035
00036     \see                so_opts
00037     \see                ipv4_opts
00038     \see                ipv6_opts
00039     \see                udp_opts
00040     \see                udplite_opts
00041
00042     @{
00043 */
00044
00045 #define TCP_NODELAY             1 /**< \brief Don't delay to coalesce. */
00046
00047 /** @} */
00048
00049 __END_DECLS
00050
00051 #endif /* !__NETINET_TCP_H */
```

9.102 include/netinet/udp.h File Reference

Definitions for the User Datagram Protocol.

```
#include <sys/cdefs.h>
```

Macros

- #define UDP_NOCHECKSUM 25

  Don't calculate UDP checksums.

9.102.1 Detailed Description

Definitions for the User Datagram Protocol.

This file contains definitions related to the User Datagram Protocol (UDP). UDP is a connectionless datagram delivery protocol, which provides optional datagram integrity validation.

UDP is described in RFC 768.

Author

Lawrence Sebald

9.103 udp.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    netinet/udp.h
00004    Copyright (C) 2014, 2023 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   netinet/udp.h
00009     \brief  Definitions for the User Datagram Protocol.
00010
00011     This file contains definitions related to the User Datagram Protocol (UDP).
00012     UDP is a connectionless datagram delivery protocol, which provides optional
00013     datagram integrity validation.
00014
00015     UDP is described in RFC 768.
00016
00017     \author Lawrence Sebald
00018 */
00019
00020 #ifndef __NETINET_UDP_H
00021 #define __NETINET_UDP_H
00022
00023 #include <sys/cdefs.h>
00024
00025 __BEGIN_DECLS
00026
00027 /** \defgroup udp_opts                  UDP protocol level options
00028
00029     These are the various socket-level options that can be accessed with the
00030     setsockopt() and getsockopt() functions for the IPPROTO_UDP level value.
00031
00032     \see                so_opts
00033     \see                ipv4_opts
00034     \see                ipv6_opts
00035     \see                udplite_opts
00036     \see                tcp_opts
00037
00038     @{
00039 */
00040
00041 #define UDP_NOCHECKSUM          25  /**< \brief Don't calculate UDP checksums */
00042
00043 /** @} */
00044
00045 __END_DECLS
00046
00047 #endif /* !__NETINET_UDP_H */
```

9.104 include/netinet/udplite.h File Reference

Definitions for UDP-Lite.

```
#include <sys/cdefs.h>
```

Macros

- #define UDPLITE_SEND_CSCOV 26

  Sending checksum coverage.
- #define UDPLITE_RECV_CSCOV 27

  Receiving checksum coverage.

9.104.1 Detailed Description

Definitions for UDP-Lite.

This file contains definitions related to UDP-Lite, a version of UDP that allows for partial checksum coverage (rather than requiring that either all or none of the packet is covered as does UDP).

UDP-Lite is described in RFC 3828.

Author

Lawrence Sebald

9.105 udplite.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    netinet/udplite.h
00004    Copyright (C) 2014, 2023 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   netinet/udplite.h
00009     \brief  Definitions for UDP-Lite.
00010
00011     This file contains definitions related to UDP-Lite, a version of UDP that
00012     allows for partial checksum coverage (rather than requiring that either all
00013     or none of the packet is covered as does UDP).
00014
00015     UDP-Lite is described in RFC 3828.
00016
00017     \author Lawrence Sebald
00018 */
00019
00020 #ifndef __NETINET_UDPLITE_H
00021 #define __NETINET_UDPLITE_H
00022
00023 #include <sys/cdefs.h>
00024
00025 __BEGIN_DECLS
00026
00027 /** \defgroup udplite_opts              UDP-Lite protocol level options
00028
00029     These are the various socket-level options that can be accessed with the
00030     setsockopt() and getsockopt() functions for the IPPROTO_UDPLITE level value.
00031
00032     \see                so_opts
00033     \see                ipv4_opts
00034     \see                ipv6_opts
00035     \see                udp_opts
00036     \see                tcp_opts
00037
00038     @{
00039 */
00040
00041 #define UDPLITE_SEND_CSCOV      26 /**< \brief Sending checksum coverage. */
00042 #define UDPLITE_RECV_CSCOV      27 /**< \brief Receiving checksum coverage. */
00043
00044 /** @} */
00045
00046 __END_DECLS
00047
00048 #endif /* !__NETINET_UDPLITE_H */
```

9.106 include/poll.h File Reference

Definitions for the poll() function.

```
#include <sys/cdefs.h>
#include <sys/types.h>
```

Data Structures

- struct pollfd

  Structure representing a single file descriptor used by poll().

Macros

- #define POLLRDNORM (1 << 0)

  Normal data may be read.
- #define POLLRDBAND (1 << 1)

  Priority data may be read.
- #define POLLPRI (1 << 2)

  High-priority data may be read.
- #define POLLOUT (1 << 3)

  Normal data may be written.
- #define POLLWRNORM POLLOUT

  Normal data may be written.
- #define POLLWRBAND (1 << 4)

  Priority data may be written.
- #define POLLERR (1 << 5)

  Error has occurred (revents only)
- #define POLLHUP (1 << 6)

  Peer disconnected (revents only)
- #define POLLNVAL (1 << 7)

  Invalid fd (revents only)
- #define POLLIN (POLLRDNORM | POLLRDBAND)

  Data other than high-priority data may be read.

Typedefs

- typedef __uint32_t nfds_t

  Type representing a number of file descriptors.

Functions

- int poll (struct pollfd fds[ ], nfds_t nfds, int timeout)

  Poll a group of file descriptors for activity.

9.106.1 Detailed Description

Definitions for the poll() function.

This file contains the definitions needed for using the poll() function, as directed by the POSIX 2008 standard (aka The Open Group Base Specifications Issue 7). Currently the functionality defined herein only works for sockets, and that is likely how it will stay for some time.

The poll() function works quite similarly to the select() function that it is quite likely that you'd be more familiar with.

Author

Lawrence Sebald

9.106.2 Typedef Documentation

nfds_t

```
typedef __uint32_t nfds_t
```

Type representing a number of file descriptors.

9.106.3 Function Documentation

poll()

```
int poll (
            struct pollfd fds[],
            nfds_t nfds,
            int timeout )
```

Poll a group of file descriptors for activity.

This function will poll a group of file descriptors to check for the events specified on them. The function shall block for the specified period of time (in milliseconds) waiting for an event to occur. The function shall return as soon as at least one fd matches the events specified (or one of the error conditions), or when timeout expires.

Parameters

fds	The file descriptors to check, and what events to look for on each.
nfds	Number of elements in fds.
timeout	Maximum amount of time to block, in milliseconds. Pass 0 to ensure the function does not block and -1 to block for an "infinite" amount of time, until an event occurs.

Returns

-1 on error (sets errno as appropriate), or the number of file descriptors that matched the event flags before the function returns.

See also

Events for the poll() function

9.107 poll.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    poll.h
00004    Copyright (C) 2012 Lawrence Sebald
00005 */
00006
00007 /** \file   poll.h
00008     \brief  Definitions for the poll() function.
00009
00010     This file contains the definitions needed for using the poll() function, as
00011     directed by the POSIX 2008 standard (aka The Open Group Base Specifications
00012     Issue 7). Currently the functionality defined herein only works for sockets,
00013     and that is likely how it will stay for some time.
00014
00015     The poll() function works quite similarly to the select() function that it
00016     is quite likely that you'd be more familiar with.
00017
00018     \author Lawrence Sebald
00019 */
00020
00021 #ifndef __POLL_H
00022 #define __POLL_H
00023
00024 #include <sys/cdefs.h>
00025 #include <sys/types.h>
00026
00027 __BEGIN_DECLS
00028
00029 /** \brief  Type representing a number of file descriptors. */
00030 typedef __uint32_t nfds_t;
00031
00032 /** \brief  Structure representing a single file descriptor used by poll().
00033     \headerfile poll.h
00034 */
00035 struct pollfd {
00036     int fd;                 /**< \brief The file descriptor in question. */
00037     short events;           /**< \brief Events to poll for on input. */
00038     short revents;          /**< \brief Events signalled for output. */
00039 };
00040
00041 /** \defgroup poll_events               Events for the poll() function
00042
00043     These are the events that can be set in the events or revents fields of the
00044     struct pollfd.
00045
00046     @{
00047 */
00048 #define POLLRDNORM  (1 << 0)    /**< \brief Normal data may be read */
00049 #define POLLRDBAND  (1 << 1)    /**< \brief Priority data may be read */
00050 #define POLLPRI     (1 << 2)    /**< \brief High-priority data may be read */
00051 #define POLLOUT     (1 << 3)    /**< \brief Normal data may be written */
00052 #define POLLWRNORM  POLLOUT     /**< \brief Normal data may be written */
00053 #define POLLWRBAND  (1 << 4)    /**< \brief Priority data may be written */
00054 #define POLLERR     (1 << 5)    /**< \brief Error has occurred (revents only) */
00055 #define POLLHUP     (1 << 6)    /**< \brief Peer disconnected (revents only) */
00056 #define POLLNVAL    (1 << 7)    /**< \brief Invalid fd (revents only) */
00057
00058 /** \brief  Data other than high-priority data may be read */
00059 #define POLLIN      (POLLRDNORM | POLLRDBAND)
00060 /** @} */
00061
```

```
00062 /** \brief  Poll a group of file descriptors for activity.
00063
00064     This function will poll a group of file descriptors to check for the events
00065     specified on them. The function shall block for the specified period of time
00066     (in milliseconds) waiting for an event to occur. The function shall return
00067     as soon as at least one fd matches the events specified (or one of the error
00068     conditions), or when timeout expires.
00069
00070     \param  fds         The file descriptors to check, and what events to look
00071                         for on each.
00072     \param  nfds        Number of elements in fds.
00073     \param  timeout     Maximum amount of time to block, in milliseconds. Pass
00074                         0 to ensure the function does not block and -1 to block
00075                         for an "infinite" amount of time, until an event occurs.
00076     \return             -1 on error (sets errno as appropriate), or the number
00077                         of file descriptors that matched the event flags before
00078                         the function returns.
00079     \sa     poll_events
00080 */
00081 int poll(struct pollfd fds[], nfds_t nfds, int timeout);
00082
00083 __END_DECLS
00084
00085 #endif /* !POLL_H */
```

9.108 include/pthread.h File Reference

POSIX-compatibile (sorta) threading support.

9.108.1 Detailed Description

POSIX-compatibile (sorta) threading support.

This file was imported (with a few changes) from Newlib. If you really want to know about the functions in here, you should probably consult the Single Unix Specification and the POSIX specification. Here's a link to that: http↩://pubs.opengroup.org/onlinepubs/007904875/basedefs/pthread.h.html

The rest of this file will remain undocumented, as it isn't really a part of KOS proper... Also, doxygen tends to mangle this whole thing anyway...

9.109 pthread.h

Go to the documentation of this file.

```
00001 /*  pthread.h
00002  *
00003  *  Written by Joel Sherrill <joel@OARcorp.com>.
00004  *
00005  *  COPYRIGHT (c) 1989-2000.
00006  *  On-Line Applications Research Corporation (OAR).
00007  *
00008  *  Permission to use, copy, modify, and distribute this software for any
00009  *  purpose without fee is hereby granted, provided that this entire notice
00010  *  is included in all copies of any software which is or includes a copy
00011  *  or modification of this software.
00012  *
00013  *  THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED
00014  *  WARRANTY.  IN PARTICULAR,  THE AUTHOR MAKES NO REPRESENTATION
00015  *  OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS
00016  *  SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.
00017  *
00018  */
00019
00020 // Need a local copy of this because the default one is buggy.
```

```
00021
00022 /** \file   pthread.h
00023     \brief  POSIX-compatibile (sorta) threading support.
00024     \ingroup threading
00025
00026     This file was imported (with a few changes) from Newlib. If you really want
00027     to know about the functions in here, you should probably consult the Single
00028     Unix Specification and the POSIX specification. Here's a link to that:
00029     http://pubs.opengroup.org/onlinepubs/007904875/basedefs/pthread.h.html
00030
00031     The rest of this file will remain undocumented, as it isn't really a part of
00032     KOS proper... Also, doxygen tends to mangle this whole thing anyway...
00033 */
00034
00035 /** \cond */
00036
00037 #ifndef __PTHREAD_h
00038 #define __PTHREAD_h
00039
00040 #ifdef __cplusplus
00041 extern "C" {
00042 #endif
00043
00044 #include <unistd.h>
00045 #include <sys/_pthread.h>
00046
00047 #if defined(_POSIX_THREADS)
00048
00049 #include <sys/types.h>
00050 #include <time.h>
00051 #include <sys/sched.h>
00052
00053     /* Register Fork Handlers, P1003.1c/Draft 10, P1003.1c/Draft 10, p. 27
00054
00055         If an OS does not support processes, then it falls under this provision
00056         and may not provide pthread_atfork():
00057
00058         "Either the implementation shall support the pthread_atfork() function
00059          as described above or the pthread_atfork() funciton shall not be
00060          provided."
00061
00062         NOTE: RTEMS does not provide pthread_atfork().  */
00063
00064 #if !defined(__rtems__)
00065 // #warning "Add pthread_atfork() prototype"
00066 #endif
00067
00068     /* Mutex Initialization Attributes, P1003.1c/Draft 10, p. 81 */
00069
00070     int pthread_mutexattr_init(pthread_mutexattr_t *attr);
00071     int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
00072     int pthread_mutexattr_getpshared(const pthread_mutexattr_t *attr, int  *pshared);
00073     int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr, int pshared);
00074
00075     /* Initializing and Destroying a Mutex, P1003.1c/Draft 10, p. 87 */
00076
00077     int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
00078     int pthread_mutex_destroy(pthread_mutex_t *mutex);
00079
00080     /* This is used to statically initialize a pthread_mutex_t. Example:
00081
00082         pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
00083      */
00084
00085 #define PTHREAD_MUTEX_INITIALIZER  MUTEX_INITIALIZER
00086
00087     /*  Locking and Unlocking a Mutex, P1003.1c/Draft 10, p. 93
00088         NOTE: P1003.4b/D8 adds pthread_mutex_timedlock(), p. 29 */
00089
00090     int pthread_mutex_lock(pthread_mutex_t *mutex);
00091     int pthread_mutex_trylock(pthread_mutex_t *mutex);
00092     int pthread_mutex_unlock(pthread_mutex_t *mutex);
00093
00094 #if defined(_POSIX_TIMEOUTS)
00095
00096     int pthread_mutex_timedlock(pthread_mutex_t *mutex, const struct timespec *timeout);
00097
00098 #endif /* _POSIX_TIMEOUTS */
00099
00100     /* Condition Variable Initialization Attributes, P1003.1c/Draft 10, p. 96 */
00101
```

```
00102     int pthread_condattr_init(pthread_condattr_t *attr);
00103     int pthread_condattr_destroy(pthread_condattr_t *attr);
00104     int pthread_condattr_getpshared(const pthread_condattr_t *attr, int *pshared);
00105     int pthread_condattr_setpshared(pthread_condattr_t *attr, int pshared);
00106 
00107     /* Initializing and Destroying a Condition Variable, P1003.1c/Draft 10, p. 87 */
00108 
00109     int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr);
00110     int pthread_cond_destroy(pthread_cond_t *cond);
00111 
00112     /* This is used to statically initialize a pthread_cond_t. Example:
00113 
00114        pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
00115      */
00116 
00117 #define PTHREAD_COND_INITIALIZER  COND_INITIALIZER
00118 
00119     /* Broadcasting and Signaling a Condition, P1003.1c/Draft 10, p. 101 */
00120 
00121     int pthread_cond_signal(pthread_cond_t *cond);
00122     int pthread_cond_broadcast(pthread_cond_t *cond);
00123 
00124     /* Waiting on a Condition, P1003.1c/Draft 10, p. 105 */
00125 
00126     int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
00127 
00128     int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex,
00129                const struct timespec *abstime);
00130 
00131 #if defined(_POSIX_THREAD_PRIORITY_SCHEDULING)
00132 
00133     /* Thread Creation Scheduling Attributes, P1003.1c/Draft 10, p. 120 */
00134 
00135     int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
00136     int pthread_attr_getscope(const pthread_attr_t *attr, int *contentionscope);
00137     int pthread_attr_setinheritsched(pthread_attr_t *attr, int inheritsched);
00138     int pthread_attr_getinheritsched(const pthread_attr_t *attr, int *inheritsched);
00139     int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
00140     int pthread_attr_getschedpolicy(const pthread_attr_t *attr, int *policy);
00141 
00142 #endif /* defined(_POSIX_THREAD_PRIORITY_SCHEDULING) */
00143 
00144     int pthread_attr_setschedparam(pthread_attr_t *attr, const struct sched_param *param);
00145     int pthread_attr_getschedparam(const pthread_attr_t *attr, struct sched_param *param);
00146 
00147 #if defined(_POSIX_THREAD_PRIORITY_SCHEDULING)
00148 
00149     /* Dynamic Thread Scheduling Parameters Access, P1003.1c/Draft 10, p. 124 */
00150 
00151     int pthread_getschedparam(pthread_t thread, int *policy, struct sched_param *param);
00152     int pthread_setschedparam(pthread_t thread, int policy, struct sched_param *param);
00153 
00154 #endif /* defined(_POSIX_THREAD_PRIORITY_SCHEDULING) */
00155 
00156 #if defined(_POSIX_THREAD_PRIO_INHERIT) || defined(_POSIX_THREAD_PRIO_PROTECT)
00157 
00158     /* Mutex Initialization Scheduling Attributes, P1003.1c/Draft 10, p. 128 */
00159 
00160     int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr, int protocol);
00161     int pthread_mutexattr_getprotocol(const pthread_mutexattr_t *attr, int *protocol);
00162     int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr, int prioceiling);
00163     int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t *attr, int *prioceiling);
00164 
00165 #endif /* _POSIX_THREAD_PRIO_INHERIT || _POSIX_THREAD_PRIO_PROTECT */
00166 
00167 #if defined(_POSIX_THREAD_PRIO_PROTECT)
00168 
00169     /* Change the Priority Ceiling of a Mutex, P1003.1c/Draft 10, p. 131 */
00170 
00171     int pthread_mutex_setprioceiling(pthread_mutex_t *mutex, int prioceiling, int *old_ceiling);
00172     int pthread_mutex_getprioceiling(pthread_mutex_t *mutex, int *prioceiling);
00173 
00174 #endif /* _POSIX_THREAD_PRIO_PROTECT */
00175 
00176     /* Thread Creation Attributes, P1003.1c/Draft 10, p, 140 */
00177 
00178     int pthread_attr_init(pthread_attr_t *attr);
00179     int pthread_attr_destroy(pthread_attr_t *attr);
00180     int pthread_attr_getstacksize(const pthread_attr_t *attr, size_t *stacksize);
00181     int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
00182     int pthread_attr_getstackaddr(const pthread_attr_t *attr, void **stackaddr);
```

```
00183     int pthread_attr_setstackaddr(pthread_attr_t  *attr, void *stackaddr);
00184     int pthread_attr_getdetachstate(const pthread_attr_t *attr, int *detachstate);
00185     int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
00186 
00187     /* Thread Creation, P1003.1c/Draft 10, p. 144 */
00188 
00189     int pthread_create(pthread_t *thread, const pthread_attr_t  *attr,
00190                void * (*start_routine)(void *), void *arg);
00191 
00192     /* Wait for Thread Termination, P1003.1c/Draft 10, p. 147 */
00193 
00194     int pthread_join(pthread_t thread, void **value_ptr);
00195 
00196     /* Detaching a Thread, P1003.1c/Draft 10, p. 149 */
00197 
00198     int pthread_detach(pthread_t thread);
00199 
00200     /* Thread Termination, p1003.1c/Draft 10, p. 150 */
00201 
00202     void pthread_exit(void *value_ptr);
00203 
00204     /* Get Calling Thread's ID, p1003.1c/Draft 10, p. XXX */
00205 
00206     pthread_t pthread_self(void);
00207 
00208     /* Compare Thread IDs, p1003.1c/Draft 10, p. 153 */
00209 
00210     int pthread_equal(pthread_t t1, pthread_t t2);
00211 
00212     /* Dynamic Package Initialization */
00213 
00214     /* This is used to statically initialize a pthread_once_t. Example:
00215 
00216         pthread_once_t once = PTHREAD_ONCE_INIT;
00217 
00218         NOTE:  This is named inconsistently -- it should be INITIALIZER.  */
00219 
00220 #define PTHREAD_ONCE_INIT  { 1, 0 }  /* is initialized and not run */
00221 
00222     int pthread_once(pthread_once_t *once_control, void (*init_routine)(void));
00223 
00224     /* Thread-Specific Data Key Create, P1003.1c/Draft 10, p. 163 */
00225 
00226     int pthread_key_create(pthread_key_t *key, void (*destructor)(void *));
00227 
00228     /* Thread-Specific Data Management, P1003.1c/Draft 10, p. 165 */
00229 
00230     int pthread_setspecific(pthread_key_t key, const void *value);
00231     void *pthread_getspecific(pthread_key_t key);
00232 
00233     /* Thread-Specific Data Key Deletion, P1003.1c/Draft 10, p. 167 */
00234 
00235     int pthread_key_delete(pthread_key_t key);
00236 
00237     /* Execution of a Thread, P1003.1c/Draft 10, p. 181 */
00238 
00239 #define PTHREAD_CANCEL_ENABLE  0
00240 #define PTHREAD_CANCEL_DISABLE 1
00241 
00242 #define PTHREAD_CANCEL_DEFERRED 0
00243 #define PTHREAD_CANCEL_ASYNCHRONOUS 1
00244 
00245 #define PTHREAD_CANCELED ((void *) -1)
00246 
00247     int pthread_cancel(pthread_t thread);
00248 
00249     /* Setting Cancelability State, P1003.1c/Draft 10, p. 183 */
00250 
00251     int pthread_setcancelstate(int state, int *oldstate);
00252     int pthread_setcanceltype(int type, int *oldtype);
00253     void pthread_testcancel(void);
00254 
00255     /* Establishing Cancellation Handlers, P1003.1c/Draft 10, p. 184 */
00256 
00257     void pthread_cleanup_push(void (*routine)(void *), void *arg);
00258     void pthread_cleanup_pop(int execute);
00259 
00260 #if defined(_POSIX_THREAD_CPUTIME)
00261 
00262     /* Accessing a Thread CPU-time Clock, P1003.4b/D8, p. 58 */
00263 
```

```
00264     int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
00265
00266     /* CPU-time Clock Thread Creation Attribute, P1003.4b/D8, p. 59 */
00267
00268     int pthread_attr_setcputime(pthread_attr_t *attr, int clock_allowed);
00269
00270     int pthread_attr_getcputime(pthread_attr_t *attr, int *clock_allowed);
00271
00272 #endif /* defined(_POSIX_THREAD_CPUTIME) */
00273
00274 #endif /* defined(_POSIX_THREADS) */
00275
00276 #ifdef __cplusplus
00277 }
00278 #endif
00279
00280 #endif
00281 /** \endcond */
00282 /* end of include file */
```

9.110 include/sys/_pthread.h File Reference

Basic sys/_pthread.h file for newlib.

Macros

- #define _POSIX_THREADS

  POSIX threads supported (sorta)
- #define _POSIX_TIMEOUTS

  POSIX timeouts supported (sorta)
- #define _POSIX_TIMERS

  POSIX timers supported (not really)

9.110.1 Detailed Description

Basic sys/_pthread.h file for newlib.

This file specifies a few things to make sure pthreads stuff compiles.

9.110.2 Macro Definition Documentation

_POSIX_THREADS

```
#define _POSIX_THREADS
```

POSIX threads supported (sorta)

_POSIX_TIMEOUTS

```
#define _POSIX_TIMEOUTS
```

POSIX timeouts supported (sorta)

_POSIX_TIMERS

```
#define _POSIX_TIMERS
```

POSIX timers supported (not really)

9.111 _pthread.h

Go to the documentation of this file.

```
00001 /** \file   sys/_pthread.h
00002     \brief  Basic sys/_pthread.h file for newlib.
00003
00004     This file specifies a few things to make sure pthreads stuff compiles.
00005 */
00006
00007 #ifndef __SYS__PTHREAD_H
00008 #define __SYS__PTHREAD_H
00009
00010 // Make sure pthreads compile ok.
00011 /** \brief  POSIX threads supported (sorta) */
00012 #define _POSIX_THREADS
00013
00014 /** \brief  POSIX timeouts supported (sorta) */
00015 #define _POSIX_TIMEOUTS
00016
00017 /** \brief  POSIX timers supported (not really) */
00018 #define _POSIX_TIMERS
00019
00020 #endif  /* __SYS__PTHREAD_H */
```

9.112 include/sys/_types.h File Reference

Internal typedefs.

```
#include <kos/cdefs.h>
#include <sys/lock.h>
#include <newlib.h>
#include <arch/types.h>
#include <sys/_pthread.h>
```

Data Structures

- struct _mbstate_t

  Conversion state information.

Macros

- #define _TIME_T_ long
- #define _CLOCKID_T_ unsigned long
- #define _TIMER_T_ unsigned long
- #define _CLOCK_T_ unsigned long /* clock() */
- #define FD_SETSIZE 1024

  Size of an fd_set.
- #define LITTLE_ENDIAN 1234

  Little Endian test macro.
- #define BIG_ENDIAN 4321

  Big Endian test macro.
- #define PDP_ENDIAN 3412

  PDP Endian test macro.
- #define AT_EACCESS 1

  Check access using effective user and group ID.
- #define AT_SYMLINK_NOFOLLOW 2

  Do not follow symlinks.
- #define AT_SYMLINK_FOLLOW 4

  Follow symbolic links.
- #define AT_REMOVEDIR 8

  Remove directory instead of file.
- #define IOV_MAX 1024

  Maximum length of an iovec, in elements.

Typedefs

- typedef long _off_t

  File offset type.
- typedef _off_t __off_t
- typedef short __dev_t

  Device ID type.
- typedef unsigned short __uid_t

  User ID type.
- typedef unsigned short __gid_t

  Group ID type.
- typedef long long _off64_t

  64-bit file offset type.
- typedef long _fpos_t

  File position type.
- typedef long _ssize_t
- typedef __newlib_recursive_lock_t _flock_t

  File lock type.
- typedef void * _iconv_t

  Iconv descriptor type.
- typedef long __blkcnt_t
- typedef long __blksize_t

- typedef long __daddr_t
- typedef unsigned long long __fsblkcnt_t
- typedef unsigned long __fsfilcnt_t
- typedef unsigned long __id_t
- typedef unsigned long __ino_t
- typedef int __pid_t
- typedef long __key_t
- typedef unsigned long __mode_t
- typedef unsigned short __nlink_t
- typedef long __suseconds_t
- typedef unsigned long __useconds_t
- typedef _TIME_T_ __time_t
- typedef _CLOCKID_T_ __clockid_t
- typedef _TIMER_T_ __timer_t
- typedef _CLOCK_T_ __clock_t
- typedef char * __va_list

9.112.1 Detailed Description

Internal typedefs.

This file contains internal typedefs required by libc. You probably shouldn't use any of these in your code. Most of these are copied from newlib's sys/_types.h.

9.112.2 Macro Definition Documentation

_CLOCK_T_

```
#define _CLOCK_T_ unsigned long /* clock() */
```

_CLOCKID_T_

```
#define _CLOCKID_T_ unsigned long
```

_TIME_T_

```
#define _TIME_T_ long
```

_TIMER_T_

```
#define _TIMER_T_ unsigned long
```

AT_EACCESS

```
#define AT_EACCESS 1
```

Check access using effective user and group ID.

AT_REMOVEDIR

```
#define AT_REMOVEDIR 8
```

Remove directory instead of file.

AT_SYMLINK_FOLLOW

```
#define AT_SYMLINK_FOLLOW 4
```

Follow symbolic links.

AT_SYMLINK_NOFOLLOW

```
#define AT_SYMLINK_NOFOLLOW 2
```

Do not follow symlinks.

BIG_ENDIAN

```
#define BIG_ENDIAN 4321
```

Big Endian test macro.

FD_SETSIZE

```
#define FD_SETSIZE 1024
```

Size of an fd_set.

IOV_MAX

```
#define IOV_MAX 1024
```

Maximum length of an iovec, in elements.

LITTLE_ENDIAN

```
#define LITTLE_ENDIAN 1234
```

Little Endian test macro.

PDP_ENDIAN

```
#define PDP_ENDIAN 3412
```

PDP Endian test macro.

9.112.3 Typedef Documentation

__blkcnt_t

```
typedef long __blkcnt_t
```

__blksize_t

```
typedef long __blksize_t
```

__clock_t

```
typedef _CLOCK_T_ __clock_t
```

__clockid_t

```
typedef _CLOCKID_T_ __clockid_t
```

__daddr_t

```
typedef long __daddr_t
```

__dev_t

```
typedef short __dev_t
```

Device ID type.

__fsblkcnt_t

```
typedef unsigned long long __fsblkcnt_t
```

__fsfilcnt_t

```
typedef unsigned long __fsfilcnt_t
```

__gid_t

```
typedef unsigned short __gid_t
```

Group ID type.

__id_t

```
typedef unsigned long __id_t
```

__ino_t

```
typedef unsigned long __ino_t
```

__key_t

```
typedef long __key_t
```

__mode_t

```
typedef unsigned long __mode_t
```

__nlink_t

```
typedef unsigned short __nlink_t
```

__off_t

```
typedef _off_t __off_t
```

__pid_t

```
typedef int __pid_t
```

__suseconds_t

```
typedef long __suseconds_t
```

__time_t

```
typedef _TIME_T_ __time_t
```

__timer_t

```
typedef _TIMER_T_ __timer_t
```

__uid_t

```
typedef unsigned short __uid_t
```

User ID type.

__useconds_t

```
typedef unsigned long __useconds_t
```

__va_list

```
typedef char* __va_list
```

_flock_t

```
typedef __newlib_recursive_lock_t _flock_t
```

File lock type.

_fpos_t

```
typedef long _fpos_t
```

File position type.

_iconv_t

```
typedef void* _iconv_t
```

Iconv descriptor type.

_off64_t

```
typedef long long _off64_t
```

64-bit file offset type.

_off_t

```
typedef long _off_t
```

File offset type.

_ssize_t

```
typedef long _ssize_t
```

9.113 _types.h

Go to the documentation of this file.

```
00001 /** \file   sys/_types.h
00002     \brief  Internal typedefs.
00003
00004     This file contains internal typedefs required by libc. You probably
00005     shouldn't use any of these in your code. Most of these are copied from
00006     newlib's %sys/_types.h.
00007 */
00008
00009 #ifndef _SYS__TYPES_H
00010 #define _SYS__TYPES_H
00011
00012 #include <kos/cdefs.h>
00013 __BEGIN_DECLS
00014
00015 #include <sys/lock.h>
00016 #include <newlib.h>
00017
00018 // This part copied from newlib's sys/_types.h.
00019 #ifndef __off_t_defined
00020 /** \brief  File offset type. */
00021 typedef long _off_t;
00022 typedef _off_t __off_t;
00023 #endif
00024
00025 #ifndef __dev_t_defined
00026 /** \brief  Device ID type. */
00027 typedef short __dev_t;
00028 #endif
00029
00030 #ifndef __uid_t_defined
00031 /** \brief  User ID type. */
00032 typedef unsigned short __uid_t;
```

```
00033 #endif
00034 #ifndef __gid_t_defined
00035 /** \brief  Group ID type. */
00036 typedef unsigned short __gid_t;
00037 #endif
00038
00039 #ifndef __off64_t_defined
00040 /** \brief  64-bit file offset type. */
00041 __extension__ typedef long long _off64_t;
00042 #endif
00043
00044 #ifndef __fpos_t_defined
00045 /** \brief  File position type. */
00046 typedef long _fpos_t;
00047 #endif
00048
00049 #ifdef __LARGE64_FILES
00050 #ifndef __fpos64_t_defined
00051 /** \brief  64-bit file position type. */
00052 typedef _off64_t _fpos64_t;
00053 #endif
00054 #endif
00055
00056 #if defined(__INT_MAX__) && __INT_MAX__ == 2147483647
00057 /** \brief  Signed size type. */
00058 typedef int _ssize_t;
00059 #else
00060 typedef long _ssize_t;
00061 #endif
00062
00063 /** \cond */
00064 #define __need_wint_t
00065 #include <stddef.h>
00066 /** \endcond */
00067
00068 #ifndef __mbstate_t_defined
00069 /** \brief  Conversion state information.
00070     \headerfile sys/_types.h
00071 */
00072 typedef struct {
00073     int __count;
00074     union {
00075         wint_t __wch;
00076         unsigned char __wchb[4];
00077     } __value;            /* Value so far.  */
00078 } _mbstate_t;
00079 #endif
00080
00081 #ifndef __flock_t_defined
00082 /** \brief  File lock type. */
00083 typedef __newlib_recursive_lock_t _flock_t;
00084 #endif
00085
00086 #ifndef __iconv_t_defined
00087 /** \brief  Iconv descriptor type. */
00088 typedef void *_iconv_t;
00089 #endif
00090
00091 #ifndef __blkcnt_t_defined
00092 typedef long __blkcnt_t;
00093 #endif
00094
00095 #ifndef __blksize_t_defined
00096 typedef long __blksize_t;
00097 #endif
00098
00099 #ifndef __daddr_t_defined
00100 typedef long __daddr_t;
00101 #endif
00102
00103 #ifndef __fsblkcnt_t_defined
00104 typedef unsigned long long __fsblkcnt_t;
00105 #endif
00106
00107 #ifndef __fsfilcnt_t_defined
00108 typedef unsigned long __fsfilcnt_t;
00109 #endif
00110
00111 #ifndef __id_t_defined
00112 typedef unsigned long __id_t;
00113 #endif
```

```
00114
00115 #ifndef __ino_t_defined
00116 typedef unsigned long __ino_t;
00117 #endif
00118
00119 #ifndef __pid_t_defined
00120 typedef int __pid_t;
00121 #endif
00122
00123 #ifndef __key_t_defined
00124 typedef long __key_t;
00125 #endif
00126
00127 #ifndef __mode_t_defined
00128 typedef unsigned long __mode_t;
00129 #endif
00130
00131 typedef unsigned short __nlink_t;
00132 typedef long        __suseconds_t;  /* microseconds (signed) */
00133 typedef unsigned long   __useconds_t;   /* microseconds (unsigned) */
00134
00135 #if __NEWLIB__ >= 3
00136 #define _TIME_T_ long long
00137 #else
00138 #define _TIME_T_ long
00139 #endif
00140 typedef _TIME_T_    __time_t;
00141
00142 #ifndef __clockid_t_defined
00143 #define _CLOCKID_T_     unsigned long
00144 #endif
00145
00146 typedef _CLOCKID_T_ __clockid_t;
00147
00148 #define _TIMER_T_   unsigned long
00149 typedef _TIMER_T_   __timer_t;
00150
00151 #ifndef __clock_t_defined
00152 #define _CLOCK_T_   unsigned long   /* clock() */
00153 #endif
00154
00155 typedef _CLOCK_T_   __clock_t;
00156
00157 // This part inserted to fix newlib brokenness.
00158 /** \brief  Size of an fd_set. */
00159 #define FD_SETSIZE      1024
00160
00161 /* The architecture should define the macro BYTE_ORDER in <arch/types.h> to
00162    equal one of these macros for code that looks for these BSD-style macros. */
00163 /** \brief  Little Endian test macro */
00164 #define LITTLE_ENDIAN   1234
00165
00166 /** \brief  Big Endian test macro */
00167 #define BIG_ENDIAN      4321
00168
00169 /** \brief  PDP Endian test macro */
00170 #define PDP_ENDIAN      3412
00171
00172 /* Sigh... for some reason, Newlib only bothers defining these on Cygwin...
00173    We're only actually concerned with AT_SYMLINK_NOFOLLOW currently. These
00174    should all be defined in <fcntl.h>, by the way. */
00175 #ifndef AT_EACCESS
00176 /** \brief  Check access using effective user and group ID */
00177 #define AT_EACCESS              1
00178 #endif
00179
00180 #ifndef AT_SYMLINK_NOFOLLOW
00181 /** \brief  Do not follow symlinks */
00182 #define AT_SYMLINK_NOFOLLOW     2
00183 #endif
00184
00185 #ifndef AT_SYMLINK_FOLLOW
00186 /** \brief  Follow symbolic links */
00187 #define AT_SYMLINK_FOLLOW       4
00188 #endif
00189
00190 #ifndef AT_REMOVEDIR
00191 /** \brief  Remove directory instead of file */
00192 #define AT_REMOVEDIR            8
00193 #endif
00194
```

```
00195 #ifndef IOV_MAX
00196 /** \brief  Maximum length of an iovec, in elements. */
00197 #define IOV_MAX                 1024
00198 #endif
00199
00200 /* This is for old KOS source compatability. */
00201 #include <arch/types.h>
00202
00203 /* Include stuff to make pthreads work as well. */
00204 #include <sys/_pthread.h>
00205
00206 #if __GNUC_MINOR__ > 95 || __GNUC__ >= 3
00207 typedef __builtin_va_list   __va_list;
00208 #else
00209 typedef char *          __va_list;
00210 #endif
00211
00212 __END_DECLS
00213
00214 #endif  /* _SYS__TYPES_H */
00215
00216 /* Grab our C11 time stuff if we got here from <time.h>. */
00217 #ifdef _TIME_H_
00218 #include <kos/time.h>
00219 #endif
00220
00221 #ifdef _STDLIB_H_
00222 #include <kos/stdlib.h>
00223 #endif
```

9.114 include/sys/dirent.h File Reference

Directory entry functionality.

```
#include <kos/cdefs.h>
#include <unistd.h>
#include <arch/types.h>
#include <kos/fs.h>
```

Data Structures

- struct dirent

  POSIX directory entry structure.
- struct DIR

  Type representing a directory stream.

Functions

- DIR * opendir (const char *name)

  Open a directory based on the specified name.
- int closedir (DIR *dir)

  Closes a directory that was previously opened.
- struct dirent * readdir (DIR *dir)

  Read an entry from a directory stream.
- int dirfd (DIR *dirp)

  Retrieve the file descriptor of an opened directory stream.

- void rewinddir (DIR *dir)

  Rewind a directory stream to the start of the directory.
- int scandir (const char *dir, struct dirent ***namelist, int(*filter)(const struct dirent *), int(*compar)(const struct dirent **, const struct dirent **))

  Not implemented.
- void seekdir (DIR *dir, off_t offset)

  Not implemented.
- off_t telldir (DIR *dir)

  Not implemented.

9.114.1 Detailed Description

Directory entry functionality.

This partially implements the standard POSIX dirent.h functionality.

Author

Megan Potter

9.114.2 Function Documentation

closedir()

```
int closedir (
            DIR * dir )
```

Closes a directory that was previously opened.

This function is used to close a directory stream that was previously opened with the opendir() function. You must do this to clean up any resources associated with the directory stream.

Parameters

dir	The directory stream to close.

Returns

0 on success. -1 on error, setting errno as appropriate.

dirfd()

```
int dirfd (
            DIR * dirp )
```

Retrieve the file descriptor of an opened directory stream.

This function retrieves the file descriptor of a directory stream that was previously opened with opendir().

Parameters

dirp	The directory stream to retrieve the descriptor of.

Returns

The file descriptor from the directory stream on success or -1 on failure (sets errno as appropriate).

Note

Do not close() the returned file descriptor. It will be closed when closedir() is called on the directory stream.

opendir()

```
DIR * opendir (
            const char * name )
```

Open a directory based on the specified name.

The directory specified is opened if it exists. A directory stream object is returned for accessing the entries of the directory.

Parameters

name	The name of the directory to open.

Returns

A directory stream object to be used with readdir() on success, NULL on failure. Sets errno as appropriate.

Note

As with other functions for opening files on the VFS, relative paths are permitted for the name parameter of this function.

See also

closedir

readdir

readdir()

```
struct dirent * readdir (
            DIR * dir )
```

Read an entry from a directory stream.

This function reads the next entry from the directory stream provided, returning the directory entry associated with the next object in the directory.

Parameters

dir	The directory stream to read from.

Returns

A pointer to the next directory entry in the directory or NULL if there are no other entries in the directory. If an error is incurred, NULL will be returned and errno set to indicate the error.

Note

Do not free the returned dirent!

rewinddir()

```
void rewinddir (
            DIR * dir )
```

Rewind a directory stream to the start of the directory.

This function rewinds the directory stream so that the next call to the readdir() function will return the first entry in the directory.

Parameters

dir	The directory stream to rewind.

Note

Some filesystems do not support this call. Notably, none of the dcload filesystems support it. Error values will be returned in errno (so set errno to 0, then check after calling the function to see if there was a problem anywhere).

scandir()

```
int scandir (
            const char * dir,
            struct dirent *** namelist,
            int(*)(const struct dirent *) filter,
            int(*)(const struct dirent **, const struct dirent **) compar )
```

Not implemented.

seekdir()

```
void seekdir (
            DIR * dir,
            off_t offset )
```

Not implemented.

telldir()

```
off_t telldir (
            DIR * dir )
```

Not implemented.

9.115 dirent.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dirent.h
00004    Copyright (C) 2003 Megan Potter
00005
00006 */
00007
00008 /** \file   dirent.h
00009     \brief  Directory entry functionality.
00010
00011     This partially implements the standard POSIX dirent.h functionality.
00012
00013     \author Megan Potter
00014 */
00015
00016 #ifndef __SYS_DIRENT_H
00017 #define __SYS_DIRENT_H
00018
00019 #include <kos/cdefs.h>
00020
00021 __BEGIN_DECLS
00022
00023 #include <unistd.h>
00024 #include <arch/types.h>
00025 #include <kos/fs.h>
00026
00027 /** \brief  POSIX directory entry structure.
00028
00029     This structure contains information about a single entry in a directory in
00030     the VFS.
00031
00032     \headerfile sys/dirent.h
00033  */
00034 struct dirent {
00035     int     d_ino;              /**< \brief File unique identifier. */
00036     off_t   d_off;              /**< \brief File offset */
00037     uint16  d_reclen;           /**< \brief Record length */
00038     uint8   d_type;             /**< \brief File type */
00039     char    d_name[256];        /**< \brief Filename */
00040 };
00041
00042 /** \brief  Type representing a directory stream.
00043
00044     This type represents a directory stream and is used by the directory reading
00045     functions to trace their position in the directory.
00046
00047     The values in this function are all private and subject to change. Do not
00048     attempt to use any of them directly.
```

```
00049
00050     \headerfile sys/dirent.h
00051 */
00052 typedef struct {
00053     file_t          fd;         /**< \brief File descriptor for the directory */
00054     struct dirent   d_ent;      /**< \brief Current directory entry */
00055 } DIR;
00056
00057 // Standard UNIX dir functions. Not all of these are fully functional
00058 // right now due to lack of support in KOS.
00059
00060 /** \brief  Open a directory based on the specified name.
00061
00062     The directory specified is opened if it exists. A directory stream object is
00063     returned for accessing the entries of the directory.
00064
00065     \param  name        The name of the directory to open.
00066     \return             A directory stream object to be used with readdir() on
00067                         success, NULL on failure. Sets errno as appropriate.
00068     \note               As with other functions for opening files on the VFS,
00069                         relative paths are permitted for the name parameter of
00070                         this function.
00071     \see    closedir
00072     \see    readdir
00073 */
00074 DIR *opendir(const char *name);
00075
00076 /** \brief  Closes a directory that was previously opened.
00077
00078     This function is used to close a directory stream that was previously opened
00079     with the opendir() function. You must do this to clean up any resources
00080     associated with the directory stream.
00081
00082     \param  dir         The directory stream to close.
00083     \return             0 on success. -1 on error, setting errno as appropriate.
00084 */
00085 int closedir(DIR *dir);
00086
00087 /** \brief  Read an entry from a directory stream.
00088
00089     This function reads the next entry from the directory stream provided,
00090     returning the directory entry associated with the next object in the
00091     directory.
00092
00093     \param  dir         The directory stream to read from.
00094     \return             A pointer to the next directory entry in the directory
00095                         or NULL if there are no other entries in the directory.
00096                         If an error is incurred, NULL will be returned and errno
00097                         set to indicate the error.
00098
00099     \note               Do not free the returned dirent!
00100 */
00101 struct dirent *readdir(DIR *dir);
00102
00103 /** \brief  Retrieve the file descriptor of an opened directory stream.
00104
00105     This function retrieves the file descriptor of a directory stream that was
00106     previously opened with opendir().
00107
00108     \param  dirp        The directory stream to retrieve the descriptor of.
00109     \return             The file descriptor from the directory stream on success
00110                         or -1 on failure (sets errno as appropriate).
00111
00112     \note               Do not close() the returned file descriptor. It will be
00113                         closed when closedir() is called on the directory
00114                         stream.
00115 */
00116 int dirfd(DIR *dirp);
00117
00118 /** \brief  Rewind a directory stream to the start of the directory.
00119
00120     This function rewinds the directory stream so that the next call to the
00121     readdir() function will return the first entry in the directory.
00122
00123     \param  dir         The directory stream to rewind.
00124
00125     \note               Some filesystems do not support this call. Notably, none
00126                         of the dcload filesystems support it. Error values will
00127                         be returned in errno (so set errno to 0, then check
00128                         after calling the function to see if there was a problem
00129                         anywhere).
```

```
00130 */
00131 void rewinddir(DIR *dir);
00132
00133 /** \brief Not implemented */
00134 int scandir(const char *dir, struct dirent ***namelist,
00135             int(*filter)(const struct dirent *),
00136             int(*compar)(const struct dirent **, const struct dirent **));
00137
00138 /** \brief Not implemented */
00139 void seekdir(DIR *dir, off_t offset);
00140
00141 /** \brief Not implemented */
00142 off_t telldir(DIR *dir);
00143
00144 __END_DECLS
00145
00146 #endif
```

9.116 include/sys/lock.h File Reference

Data Structures

- struct __newlib_recursive_lock_t

Macros

- #define __NEWLIB_RECURSIVE_LOCK_INIT { (void *)0, 0, 0 }
- #define __NEWLIB_LOCK_INIT 0
- #define __LOCK_INIT(class, lock) class _LOCK_T lock = __NEWLIB_LOCK_INIT;
- #define __LOCK_INIT_RECURSIVE(class, lock) class _LOCK_RECURSIVE_T lock = __NEWLIB_RECURSIVE_LOCK_INIT;
- #define __lock_init(lock) __newlib_lock_init(&(lock))
- #define __lock_init_recursive(lock) __newlib_lock_init_recursive(&(lock))
- #define __lock_close(lock) __newlib_lock_close(&(lock))
- #define __lock_close_recursive(lock) __newlib_lock_close_recursive(&(lock))
- #define __lock_acquire(lock) __newlib_lock_acquire(&(lock))
- #define __lock_acquire_recursive(lock) __newlib_lock_acquire_recursive(&(lock))
- #define __lock_try_acquire(lock) __newlib_lock_try_acquire(&(lock))
- #define __lock_try_acquire_recursive(lock) __newlib_lock_try_acquire_recursive(&(lock))
- #define __lock_release(lock) __newlib_lock_release(&(lock))
- #define __lock_release_recursive(lock) __newlib_lock_release_recursive(&(lock))

Typedefs

- typedef volatile int __newlib_lock_t
- typedef unsigned long int _COND_T
- typedef __newlib_lock_t _LOCK_T
- typedef __newlib_recursive_lock_t _LOCK_RECURSIVE_T

Functions

- void __newlib_lock_init (__newlib_lock_t *)
- void __newlib_lock_close (__newlib_lock_t *)
- void __newlib_lock_acquire (__newlib_lock_t *)
- void __newlib_lock_try_acquire (__newlib_lock_t *)
- void __newlib_lock_release (__newlib_lock_t *)
- void __newlib_lock_init_recursive (__newlib_recursive_lock_t *)
- void __newlib_lock_close_recursive (__newlib_recursive_lock_t *)
- void __newlib_lock_acquire_recursive (__newlib_recursive_lock_t *)
- void __newlib_lock_try_acquire_recursive (__newlib_recursive_lock_t *)
- void __newlib_lock_release_recursive (__newlib_recursive_lock_t *)

9.116.1 Macro Definition Documentation

__lock_acquire

```
#define __lock_acquire(
              lock ) __newlib_lock_acquire(&(lock))
```

__lock_acquire_recursive

```
#define __lock_acquire_recursive(
              lock ) __newlib_lock_acquire_recursive(&(lock))
```

__lock_close

```
#define __lock_close(
              lock ) __newlib_lock_close(&(lock))
```

__lock_close_recursive

```
#define __lock_close_recursive(
              lock ) __newlib_lock_close_recursive(&(lock))
```

__LOCK_INIT

```
#define __LOCK_INIT(
              class,
              lock ) class _LOCK_T lock = __NEWLIB_LOCK_INIT;
```

__lock_init

```
#define __lock_init(
              lock ) __newlib_lock_init(&(lock))
```

__LOCK_INIT_RECURSIVE

```
#define __LOCK_INIT_RECURSIVE(
              class,
              lock ) class _LOCK_RECURSIVE_T lock = __NEWLIB_RECURSIVE_LOCK_INIT;
```

__lock_init_recursive

```
#define __lock_init_recursive(
              lock ) __newlib_lock_init_recursive(&(lock))
```

__lock_release

```
#define __lock_release(
              lock ) __newlib_lock_release(&(lock))
```

__lock_release_recursive

```
#define __lock_release_recursive(
              lock ) __newlib_lock_release_recursive(&(lock))
```

__lock_try_acquire

```
#define __lock_try_acquire(
              lock ) __newlib_lock_try_acquire(&(lock))
```

__lock_try_acquire_recursive

```
#define __lock_try_acquire_recursive(
              lock ) __newlib_lock_try_acquire_recursive(&(lock))
```

__NEWLIB_LOCK_INIT

```
#define __NEWLIB_LOCK_INIT 0
```

__NEWLIB_RECURSIVE_LOCK_INIT

```
#define __NEWLIB_RECURSIVE_LOCK_INIT { (void *)0, 0, 0 }
```

9.116.2 Typedef Documentation

__newlib_lock_t

```
typedef volatile int __newlib_lock_t
```

_COND_T

```
typedef unsigned long int _COND_T
```

_LOCK_RECURSIVE_T

```
typedef __newlib_recursive_lock_t _LOCK_RECURSIVE_T
```

_LOCK_T

```
typedef __newlib_lock_t _LOCK_T
```

9.116.3 Function Documentation

__newlib_lock_acquire()

```
void __newlib_lock_acquire (
            __newlib_lock_t *  )
```

__newlib_lock_acquire_recursive()

```
void __newlib_lock_acquire_recursive (
            __newlib_recursive_lock_t *  )
```

__newlib_lock_close()

```
void __newlib_lock_close (
            __newlib_lock_t *  )
```

__newlib_lock_close_recursive()

```
void __newlib_lock_close_recursive (
            __newlib_recursive_lock_t *  )
```

__newlib_lock_init()

```
void __newlib_lock_init (
            __newlib_lock_t *  )
```

__newlib_lock_init_recursive()

```
void __newlib_lock_init_recursive (
            __newlib_recursive_lock_t *  )
```

__newlib_lock_release()

```
void __newlib_lock_release (
            __newlib_lock_t *  )
```

__newlib_lock_release_recursive()

```
void __newlib_lock_release_recursive (
            __newlib_recursive_lock_t *  )
```

__newlib_lock_try_acquire()

```
void __newlib_lock_try_acquire (
            __newlib_lock_t *  )
```

__newlib_lock_try_acquire_recursive()

```
void __newlib_lock_try_acquire_recursive (
            __newlib_recursive_lock_t *  )
```

9.117 lock.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    lock_common.h
00004    Copyright (C)2004 Megan Potter
00005
00006    This file is patched into the dc-chain newlib's <sys/lock.h>, by newlib.mk
00007    in \utils\dc-chain\scripts.
00008 */
00009
00010 #ifndef __SYS_LOCK_H__
00011 #define __SYS_LOCK_H__
00012
00013 typedef struct {
00014     void* owner;
00015     int nest;
00016     volatile int lock;
00017 } __newlib_recursive_lock_t;
00018
00019 #define __NEWLIB_RECURSIVE_LOCK_INIT { (void *)0, 0, 0 }
00020
00021 typedef volatile int __newlib_lock_t;
00022 #define __NEWLIB_LOCK_INIT 0
00023
00024 typedef unsigned long int _COND_T;
00025 typedef __newlib_lock_t _LOCK_T;
00026 typedef __newlib_recursive_lock_t _LOCK_RECURSIVE_T;
00027
00028 #define __LOCK_INIT(class,lock) class _LOCK_T lock = __NEWLIB_LOCK_INIT;
00029 #define __LOCK_INIT_RECURSIVE(class,lock) class _LOCK_RECURSIVE_T lock = __NEWLIB_RECURSIVE_LOCK_INIT;
00030 #define __lock_init(lock) __newlib_lock_init(&(lock))
00031 #define __lock_init_recursive(lock) __newlib_lock_init_recursive(&(lock))
00032 #define __lock_close(lock) __newlib_lock_close(&(lock))
00033 #define __lock_close_recursive(lock) __newlib_lock_close_recursive(&(lock))
00034 #define __lock_acquire(lock) __newlib_lock_acquire(&(lock))
00035 #define __lock_acquire_recursive(lock) __newlib_lock_acquire_recursive(&(lock))
00036 #define __lock_try_acquire(lock) __newlib_lock_try_acquire(&(lock))
00037 #define __lock_try_acquire_recursive(lock) __newlib_lock_try_acquire_recursive(&(lock))
00038 #define __lock_release(lock) __newlib_lock_release(&(lock))
00039 #define __lock_release_recursive(lock) __newlib_lock_release_recursive(&(lock))
00040
00041 void __newlib_lock_init(__newlib_lock_t*);
00042 void __newlib_lock_close(__newlib_lock_t*);
00043 void __newlib_lock_acquire(__newlib_lock_t*);
00044 void __newlib_lock_try_acquire(__newlib_lock_t*);
00045 void __newlib_lock_release(__newlib_lock_t*);
00046
00047 void __newlib_lock_init_recursive(__newlib_recursive_lock_t*);
00048 void __newlib_lock_close_recursive(__newlib_recursive_lock_t*);
00049 void __newlib_lock_acquire_recursive(__newlib_recursive_lock_t*);
00050 void __newlib_lock_try_acquire_recursive(__newlib_recursive_lock_t*);
00051 void __newlib_lock_release_recursive(__newlib_recursive_lock_t*);
00052
00053
00054 #endif // __NEWLIB_LOCK_COMMON_H
00055
```

9.118 include/sys/sched.h File Reference

Basic sys/sched.h file for newlib.

```
#include <kos/cdefs.h>
#include <kos/thread.h>
#include <kos/sem.h>
#include <kos/cond.h>
#include <kos/mutex.h>
#include <kos/tls.h>
#include <kos/once.h>
```

Data Structures

- struct sched_param

  Scheduling Parameters, P1003.1b-1993, p. 249.
- struct pthread_mutexattr_t

  POSIX mutex attributes.
- struct pthread_condattr_t

  POSIX condition variable attributes.
- struct pthread_attr_t

  POSIX thread attributes.

Macros

- #define SCHED_OTHER 0

  Other scheduling.
- #define SCHED_FIFO 1

  FIFO scheduling.
- #define SCHED_RR 2

  Round-robin scheduling.

Typedefs

- typedef kthread_t * pthread_t

  POSIX thread type.
- typedef mutex_t pthread_mutex_t

  POSIX mutex type.
- typedef condvar_t pthread_cond_t

  POSIX condition type.
- typedef kthread_once_t pthread_once_t

  POSIX once control.
- typedef kthread_key_t pthread_key_t

  POSIX thread data key.

9.118.1 Detailed Description

Basic sys/sched.h file for newlib.

This file specifies a few things to make sure pthreads stuff compiles.

9.118.2 Macro Definition Documentation

SCHED_FIFO

```
#define SCHED_FIFO 1
```

FIFO scheduling.

SCHED_OTHER

```
#define SCHED_OTHER 0
```

Other scheduling.

SCHED_RR

```
#define SCHED_RR 2
```

Round-robin scheduling.

9.118.3 Typedef Documentation

pthread_cond_t

```
typedef condvar_t pthread_cond_t
```

POSIX condition type.

pthread_key_t

```
typedef kthread_key_t pthread_key_t
```

POSIX thread data key.

pthread_mutex_t

```
typedef mutex_t pthread_mutex_t
```

POSIX mutex type.

pthread_once_t

```
typedef kthread_once_t pthread_once_t
```

POSIX once control.

pthread_t

```
typedef kthread_t* pthread_t
```

POSIX thread type.

9.119 sched.h

Go to the documentation of this file.

```
00001 /** \file   sys/sched.h
00002     \brief  Basic sys/sched.h file for newlib.
00003
00004     This file specifies a few things to make sure pthreads stuff compiles.
00005 */
00006
00007 #ifndef __SYS_SCHED_H
00008 #define __SYS_SCHED_H
00009
00010 #include <kos/cdefs.h>
00011 __BEGIN_DECLS
00012
00013 // These are copied from Newlib to make stuff compile as expected.
00014
00015 #define SCHED_OTHER    0    /**< \brief Other scheduling */
00016 #define SCHED_FIFO     1    /**< \brief FIFO scheduling */
00017 #define SCHED_RR       2    /**< \brief Round-robin scheduling */
00018
00019 /** \brief  Scheduling Parameters, P1003.1b-1993, p. 249.
00020     \note   Fields whose name begins with "ss_" added by P1003.4b/D8, p. 33.
00021     \headerfile sys/sched.h
00022 */
00023 struct sched_param {
00024     int sched_priority;           /**< \brief Process execution scheduling priority */
00025 };
00026
00027 // And all this maps pthread types to KOS types for pthread.h.
00028 #include <kos/thread.h>
00029 #include <kos/sem.h>
00030 #include <kos/cond.h>
00031 #include <kos/mutex.h>
00032 #include <kos/tls.h>
00033 #include <kos/once.h>
00034
00035 // Missing structs we don't care about in this impl.
00036 /** \brief  POSIX mutex attributes.
00037
00038     Not implemented in KOS.
00039
00040     \headerfile sys/sched.h
00041 */
00042 typedef struct {
00043     // Empty
00044 } pthread_mutexattr_t;
00045
00046 /** \brief  POSIX condition variable attributes.
00047
00048     Not implemented in KOS.
00049
00050     \headerfile sys/sched.h
00051 */
00052 typedef struct {
00053     // Empty
00054 } pthread_condattr_t;
00055
00056 /** \brief  POSIX thread attributes.
00057
00058     Not implemented in KOS.
00059
00060     \headerfile sys/sched.h
00061 */
00062 typedef struct {
00063     // Empty
00064 } pthread_attr_t;
00065
00066 // Map over KOS types. The mutex/condvar maps have to be pointers
00067 // because we allow _INIT #defines to work.
00068 typedef kthread_t * pthread_t;      /**< \brief POSIX thread type */
00069 typedef mutex_t pthread_mutex_t;    /**< \brief POSIX mutex type */
00070 typedef condvar_t pthread_cond_t;   /**< \brief POSIX condition type */
00071
00072 // These, on the other hand, map right over.
00073 typedef kthread_once_t pthread_once_t;  /**< \brief POSIX once control */
00074 typedef kthread_key_t pthread_key_t;    /**< \brief POSIX thread data key */
00075
00076 __END_DECLS
```

```
00077
00078 #endif  /* __SYS_SCHED_H */
```

9.120 include/sys/select.h File Reference

Definitions for the select() function.

```
#include <sys/cdefs.h>
#include <sys/types.h>
#include <newlib.h>
#include <time.h>
#include <sys/time.h>
```

Data Structures

- struct fd_set

Macros

- #define _SYS_TYPES_FD_SET
- #define FD_SETSIZE 1024
- #define NFDBITS 32
- #define FD_SET(n, p) ((p)->fds_bits[(n) / NFDBITS] |= (1 << ((n) % NFDBITS)))
- #define FD_CLR(n, p) ((p)->fds_bits[(n) / NFDBITS] &= ~(1 << ((n) % NFDBITS)))
- #define FD_ISSET(n, p) ((p)->fds_bits[(n) / NFDBITS] & (1 << ((n) % NFDBITS)))
- #define FD_ZERO(p)

Functions

- int select (int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds, struct timeval *timeout)

  Wait for activity on a group of file descriptors.

9.120.1 Detailed Description

Definitions for the select() function.

This file contains the definitions needed for using the select() function, as directed by the POSIX 2008 standard (aka The Open Group Base Specifications Issue 7). Currently the functionality defined herein only really works for sockets, and that is likely how it will stay for some time.

Author

Lawrence Sebald

9.120.2 Macro Definition Documentation

_SYS_TYPES_FD_SET

```
#define _SYS_TYPES_FD_SET
```

FD_CLR

```
#define FD_CLR(
              n,
              p ) ((p)->fds_bits[(n) / NFDBITS] &= ~(1 << ((n) % NFDBITS)))
```

FD_ISSET

```
#define FD_ISSET(
              n,
              p ) ((p)->fds_bits[(n) / NFDBITS] & (1 << ((n) % NFDBITS)))
```

FD_SET

```
#define FD_SET(
              n,
              p ) ((p)->fds_bits[(n) / NFDBITS] |= (1 << ((n) % NFDBITS)))
```

FD_SETSIZE

```
#define FD_SETSIZE 1024
```

FD_ZERO

```
#define FD_ZERO(
              p )
```

Value:

```
    do { \
        int __i; \
        for(__i = 0; __i < FD_SETSIZE / NFDBITS; ++__i) { \
            (p)->fds_bits[__i] = 0; \
        } \
    } while(0)
```

NFDBITS

```
#define NFDBITS 32
```

9.120.3 Function Documentation

select()

```
int select (
              int nfds,
              fd_set * readfds,
              fd_set * writefds,
              fd_set * errorfds,
              struct timeval * timeout )
```

Wait for activity on a group of file descriptors.

This function will check the specfied group of file descriptors for activity and wait for activity (up to the timeout specified) if there is not any pending events already.

Parameters

nfds	The maximum fd specified in any of the sets, plus 1.
readfds	File descriptors to check for the ability to read without blocking.
writefds	File descriptors to check for the ability to write without blocking.
errorfds	File descriptors to check for error/exceptional conditions.
timeout	Maximum amount of time to block. Passing a 0 timeout will make the funciton not block, Passing NULL here will make the function block indefinitely.

Returns

-1 on error (sets errno as appropriate), or the number of bits set in the fd sets on success (this may be 0 if the timeout expires).

9.121 select.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    sys/select.h
00004    Copyright (C) 2012 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   select.h
00009     \brief  Definitions for the select() function.
00010
00011     This file contains the definitions needed for using the select() function,
00012     as directed by the POSIX 2008 standard (aka The Open Group Base
00013     Specifications Issue 7). Currently the functionality defined herein only
00014     really works for sockets, and that is likely how it will stay for some time.
00015
00016     \author Lawrence Sebald
00017 */
00018
00019 #ifndef __SYS_SELECT_H
00020 #define __SYS_SELECT_H
00021
00022 #include <sys/cdefs.h>
00023 #include <sys/types.h>
00024
```

```
00025 __BEGIN_DECLS
00026
00027 #include <newlib.h>
00028
00029 #if __NEWLIB__ > 2 || (__NEWLIB__ == 2 && __NEWLIB_MINOR__ > 2)
00030 #include <sys/_timeval.h>
00031 #else
00032 #include <time.h>
00033 #include <sys/time.h>
00034 #endif
00035
00036 /* Newlib used to define fd_set and friends in <sys/types.h>, but at some point
00037    that stopped being the case... This should tell us whether we need to define
00038    it here or not... */
00039 #ifndef _SYS_TYPES_FD_SET
00040
00041 #define _SYS_TYPES_FD_SET
00042
00043 #ifndef FD_SETSIZE
00044 /* This matches fs.h. */
00045 #define FD_SETSIZE 1024
00046 #endif
00047
00048 #define NFDBITS 32
00049
00050 typedef struct fd_set {
00051     unsigned long fds_bits[FD_SETSIZE / NFDBITS];
00052 } fd_set;
00053
00054 #define FD_SET(n, p)     ((p)->fds_bits[(n) / NFDBITS] |= (1 « ((n) % NFDBITS)))
00055 #define FD_CLR(n, p)     ((p)->fds_bits[(n) / NFDBITS] &= ~(1 « ((n) % NFDBITS)))
00056 #define FD_ISSET(n, p)   ((p)->fds_bits[(n) / NFDBITS] & (1 « ((n) % NFDBITS)))
00057 #define FD_ZERO(p)       \
00058     do { \
00059         int __i; \
00060         for(__i = 0; __i < FD_SETSIZE / NFDBITS; ++__i) { \
00061             (p)->fds_bits[__i] = 0; \
00062         } \
00063     } while(0)
00064
00065 #endif /* !_SYS_TYPES_FD_SET */
00066
00067 /** \brief  Wait for activity on a group of file descriptors.
00068
00069     This function will check the specfied group of file descriptors for activity
00070     and wait for activity (up to the timeout specified) if there is not any
00071     pending events already.
00072
00073     \param  nfds        The maximum fd specified in any of the sets, plus 1.
00074     \param  readfds     File descriptors to check for the ability to read
00075                         without blocking.
00076     \param  writefds    File descriptors to check for the ability to write
00077                         without blocking.
00078     \param  errorfds    File descriptors to check for error/exceptional
00079                         conditions.
00080     \param  timeout     Maximum amount of time to block. Passing a 0 timeout
00081                         will make the funciton not block, Passing NULL here will
00082                         make the function block indefinitely.
00083     \return             -1 on error (sets errno as appropriate), or the number
00084                         of bits set in the fd sets on success (this may be 0 if
00085                         the timeout expires).
00086 */
00087 int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds,
00088            struct timeval *timeout);
00089
00090 __END_DECLS
00091
00092 #endif /* !__SYS_SELECT_H */
```

9.122 include/sys/socket.h File Reference

Main sockets header.

```
#include <sys/cdefs.h>
#include <sys/types.h>
```

```
#include <sys/uio.h>
```

Data Structures

- struct sockaddr

  Socket address structure.
- struct sockaddr_storage

  Socket address structure of appropriate size to hold any supported socket type's addresses.

Macros

- #define _SS_MAXSIZE 128

  Size of the struct sockaddr_storage. The size here is chosen for compatibility with anything that may expect the struct sockaddr_storage to be of size 128. Technically, since there are no current plans to support AF_UNIX sockets, this could be smaller, but most implementations make it 128 bytes.
- #define _SS_ALIGNSIZE (sizeof(__uint64_t))

  Desired alignment of struct sockaddr_storage.
- #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))

  First padding size used within struct sockaddr_storage.
- #define _SS_PAD2SIZE

  Second padding size used within struct sockaddr_storage.
- #define SOCK_DGRAM 1

  Datagram socket type.
- #define SOCK_STREAM 2

  Stream socket type.
- #define SOL_SOCKET 1

  Socket-level option setting.
- #define SO_ACCEPTCONN 1

  Socket is accepting connections (get)
- #define SO_BROADCAST 2

  Support broadcasting (get/set)
- #define SO_DEBUG 3

  Record debugging info (get/set)
- #define SO_DONTROUTE 4

  Do not route packets (get/set)
- #define SO_ERROR 5

  Retrieve error status (get)
- #define SO_KEEPALIVE 6

  Send keepalive messages (get/set)
- #define SO_LINGER 7

  Socket lingers on close (get/set)
- #define SO_OOBINLINE 8

  OOB data is inline (get/set)
- #define SO_RCVBUF 9

  Receive buffer size (get/set)

- #define SO_RCVLOWAT 10

  Receive low-water mark (get/set)
- #define SO_RCVTIMEO 11

  Receive timeout value (get/set)
- #define SO_REUSEADDR 12

  Reuse local addresses (get/set)
- #define SO_SNDBUF 13

  Send buffer size (get/set)
- #define SO_SNDLOWAT 14

  Send low-water mark (get/set)
- #define SO_SNDTIMEO 15

  Send timeout value (get/set)
- #define SO_TYPE 16

  Socket type (get)
- #define MSG_CTRUNC 0x01

  Control data truncated (U)
- #define MSG_DONTROUTE 0x02

  Send without routing (U)
- #define MSG_EOR 0x04

  Terminate a record (U)
- #define MSG_OOB 0x08

  Out-of-band data (U)
- #define MSG_PEEK 0x10

  Leave received data in queue.
- #define MSG_TRUNC 0x20

  Normal data truncated (U)
- #define MSG_WAITALL 0x40

  Attempt to fill read buffer.
- #define MSG_DONTWAIT 0x80

  Make this call non-blocking (non-standard)
- #define AF_UNSPEC 0

  Unspecified address family.
- #define AF_INET 1

  Internet domain sockets for use with IPv4 addresses.
- #define AF_INET6 2

  Internet domain sockets for use with IPv6 addresses.
- #define PF_UNSPEC AF_UNSPEC

  Unspecified protocol family.
- #define PF_INET AF_INET

  Protocol family for Internet domain sockets (IPv4).
- #define PF_INET6 AF_INET6

  Protocol family for Internet domain sockets (IPv6).
- #define SHUT_RD 0x00000001

  Disable further receive operations.
- #define SHUT_WR 0x00000002

  Disable further send operations.
- #define SHUT_RDWR (SHUT_RD | SHUT_WR)

  Disable further send and receive operations.
- #define SOMAXCONN 32

  Maximum backlog for a listening socket.

Typedefs

- typedef __uint32_t socklen_t

  Socket length type.
- typedef __uint8_t sa_family_t

  Socket address family type.

Functions

- int accept (int socket, struct sockaddr *address, socklen_t *address_len)

  Accept a new connection on a socket.
- int bind (int socket, const struct sockaddr *address, socklen_t address_len)

  Bind a name to a socket.
- int connect (int socket, const struct sockaddr *address, socklen_t address_len)

  Connect a socket.
- int listen (int socket, int backlog)

  Listen for socket connections and set the queue length.
- ssize_t recv (int socket, void *buffer, size_t length, int flags)

  Receive a message on a connected socket.
- ssize_t recvfrom (int socket, void *buffer, size_t length, int flags, struct sockaddr *address, socklen_t *address_len)

  Receive a message on a socket.
- ssize_t send (int socket, const void *message, size_t length, int flags)

  Send a message on a connected socket.
- ssize_t sendto (int socket, const void *message, size_t length, int flags, const struct sockaddr *dest_addr, socklen_t dest_len)

  Send a message on a socket.
- int shutdown (int socket, int how)

  Shutdown socket send and receive operations.
- int socket (int domain, int type, int protocol)

  Create an endpoint for communications.
- int getsockname (int socket, struct sockaddr *name, socklen_t *name_len)

  Get socket name.
- int getsockopt (int socket, int level, int option_name, void *option_value, socklen_t *option_len)

  Get socket options.
- int setsockopt (int socket, int level, int option_name, const void *option_value, socklen_t option_len)

  Set socket options.

9.122.1 Detailed Description

Main sockets header.

This file contains the standard definitions (as directed by the POSIX 2008 spec) for socket-related functionality in the AF_INET and AF_INET6 address families. This does not include anything related to UNIX domain sockets and is not guaranteed to have everything that one might have in a fully-standards compliant implementation of the POSIX spec.

Author

Lawrence Sebald

9.122.2 Macro Definition Documentation

_SS_ALIGNSIZE

```
#define _SS_ALIGNSIZE (sizeof(__uint64_t))
```

Desired alignment of struct sockaddr_storage.

_SS_MAXSIZE

```
#define _SS_MAXSIZE 128
```

Size of the struct sockaddr_storage. The size here is chosen for compatibility with anything that may expect the struct sockaddr_storage to be of size 128. Technically, since there are no current plans to support AF_UNIX sockets, this could be smaller, but most implementations make it 128 bytes.

_SS_PAD1SIZE

```
#define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
```

First padding size used within struct sockaddr_storage.

_SS_PAD2SIZE

```
#define _SS_PAD2SIZE
```

Value:

```
(_SS_MAXSIZE - (sizeof(sa_family_t) + \
_SS_PAD1SIZE + _SS_ALIGNSIZE))
```

Second padding size used within struct sockaddr_storage.

AF_INET

```
#define AF_INET 1
```

Internet domain sockets for use with IPv4 addresses.

AF_INET6

```
#define AF_INET6 2
```

Internet domain sockets for use with IPv6 addresses.

AF_UNSPEC

```
#define AF_UNSPEC 0
```

Unspecified address family.

PF_INET

```
#define PF_INET AF_INET
```

Protocol family for Internet domain sockets (IPv4).

PF_INET6

```
#define PF_INET6 AF_INET6
```

Protocol family for Internet domain sockets (IPv6).

PF_UNSPEC

```
#define PF_UNSPEC AF_UNSPEC
```

Unspecified protocol family.

SHUT_RD

```
#define SHUT_RD 0x00000001
```

Disable further receive operations.

SHUT_RDWR

```
#define SHUT_RDWR (SHUT_RD | SHUT_WR)
```

Disable further send and receive operations.

SHUT_WR

```
#define SHUT_WR 0x00000002
```

Disable further send operations.

SOCK_DGRAM

```
#define SOCK_DGRAM 1
```

Datagram socket type.

This socket type specifies that the socket in question transmits datagrams that may or may not be reliably transmitted. With IP, this implies using UDP as the underlying protocol.

SOCK_STREAM

```
#define SOCK_STREAM 2
```

Stream socket type.

This socket type specifies that the socket in question acts like a stream or pipe between the two endpoints. Sockets of this type can be assumed to be reliable – unless an error is returned, all packets will be received at the other end in the order they are sent. With IP, this implies using TCP as the underlying protocol.

SOL_SOCKET

```
#define SOL_SOCKET 1
```

Socket-level option setting.

This constant should be used with the setsockopt() or getsockopt() function to represent that options should be accessed at the socket level, not the protocol level.

SOMAXCONN

```
#define SOMAXCONN 32
```

Maximum backlog for a listening socket.

9.122.3 Typedef Documentation

sa_family_t

```
typedef __uint8_t sa_family_t
```

Socket address family type.

socklen_t

```
typedef __uint32_t socklen_t
```

Socket length type.

9.122.4 Function Documentation

accept()

```
int accept (
            int socket,
            struct sockaddr * address,
            socklen_t * address_len )
```

Accept a new connection on a socket.

This function extracts the first connection on the queue of connections of the specified socket, creating a new socket with the same protocol and address family as that socket for communication with the extracted connection.

Parameters

socket	A socket created with socket() that has been bound to an address with bind() and is listening for connections after a call to listen().
address	A pointer to a sockaddr structure where the address of the connecting socket will be returned (can be NULL).
address_len	A pointer to a socklen_t which specifies the amount of space in address on input, and the amount used of the space on output.

Returns

On success, the non-negative file descriptor of the new connection, otherwise -1 and errno will be set to the appropriate error value.

bind()

```
int bind (
            int socket,
            const struct sockaddr * address,
            socklen_t address_len )
```

Bind a name to a socket.

This function assigns the socket to a unique name (address).

Parameters

socket	A socket that is to be bound.
address	A pointer to a sockaddr structure where the name to be assigned to the socket resides.
address_len	The length of the address structure.

Return values

0	On success.
-1	On error, sets errno as appropriate.

connect()

```
int connect (
            int socket,
            const struct sockaddr * address,
            socklen_t address_len )
```

Connect a socket.

This function attempts to make a connection to a resource on a connection- mode socket, or sets/resets the peer address on a connectionless one.

Parameters

socket	A socket that is to be connected.
address	A pointer to a sockaddr structure where the name of the peer resides.
address_len	The length of the address structure.

Return values

0	On success.
-1	On error, sets errno as appropriate.

getsockname()

```
int getsockname (
            int socket,
            struct sockaddr * name,
            socklen_t * name_len )
```

Get socket name.

This function returns the locally bound address information for a specified socket.

Parameters

socket	The socket to get the name of.
name	Pointer to a sockaddr structure which will hold the resulting address information.
name_len	The amount of space pointed to by name, in bytes. On return, this is set to the actual size of the returned address information.

Return values

-1	On error, sets errno as appropriate.
0	On success.

getsockopt()

```
int getsockopt (
             int socket,
             int level,
             int option_name,
             void * option_value,
             socklen_t * option_len )
```

Get socket options.

This function retrieves options associated with a socket. This function shall attempt to retrieve the specified option (at the specified level) from the given socket.

Parameters

socket	The socket to get options for.
level	The protocol level to get options at.
option_name	The option to look up.
option_value	Storage for the value of the option.
option_len	The length of option_value on call, and the real option length (if less than the original value) on return.

Returns

Zero on success. -1 on error, and sets errno as appropriate.

See also

Socket-level options

IPv4 protocol level options

IPv6 protocol level options

UDP protocol level options

listen()

```
int listen (
             int socket,
             int backlog )
```

Listen for socket connections and set the queue length.

This function marks a connection-mode socket for incoming connections.

Parameters

socket	A connection-mode socket to listen on.
backlog	The number of queue entries.

Return values

0	On success.
-1	On error, sets errno as appropriate.

recv()

```
ssize_t recv (
             int socket,
             void * buffer,
             size_t length,
             int flags )
```

Receive a message on a connected socket.

This function receives messages from the peer on a connected socket.

Parameters

socket	The socket to receive on.
buffer	A pointer to a buffer to store the message in.
length	The length of the buffer.
flags	The type of message reception. Set to 0 for now.

Returns

On success, the length of the message in bytes. If no messages are available, and the socket has been shut down, 0. On error, -1, and sets errno as appropriate.

recvfrom()

```
ssize_t recvfrom (
```

```
            int socket,
            void * buffer,
            size_t length,
            int flags,
            struct sockaddr * address,
            socklen_t * address_len )
```

Receive a message on a socket.

This function receives messages from a peer on a (usually connectionless) socket.

Parameters

socket	The socket to receive on.
buffer	A pointer to a buffer to store the message in.
length	The length of the buffer.
flags	The type of message reception. Set to 0 for now.
address	A pointer to a sockaddr structure to store the peer's name in.
address_len	A pointer to the length of the address structure on input, the number of bytes used on output.

Returns

On success, the length of the message in bytes. If no messages are available, and the socket has been shut down, 0. On error, -1, and sets errno as appropriate.

send()

```
ssize_t send (
            int socket,
            const void * message,
            size_t length,
            int flags )
```

Send a message on a connected socket.

This function sends messages to the peer on a connected socket.

Parameters

socket	The socket to send on.
message	A pointer to a buffer with the message to send.
length	The length of the message.
flags	The type of message transmission. Set to 0 for now.

Returns

On success, the number of bytes sent. On error, -1, and sets errno as appropriate.

sendto()

```
ssize_t sendto (
            int socket,
            const void * message,
            size_t length,
            int flags,
            const struct sockaddr * dest_addr,
            socklen_t dest_len )
```

Send a message on a socket.

This function sends messages to the peer on a (usually connectionless) socket. If used on a connection-mode socket, this function may change the peer that the socket is connected to, or it may simply return error.

Parameters

socket	The socket to send on.
message	A pointer to a buffer with the message to send.
length	The length of the message.
flags	The type of message transmission. Set to 0 for now.
dest_addr	A pointer to a sockaddr structure with the peer's name.
dest_len	The length of dest_addr, in bytes.

Returns

On success, the number of bytes sent. On error, -1, and sets errno as appropriate.

setsockopt()

```
int setsockopt (
            int socket,
            int level,
            int option_name,
            const void * option_value,
            socklen_t option_len )
```

Set socket options.

This function sets options associated with a socket. This function shall attempt to set the specified option (at the specified level) from the given socket.

Parameters

socket	The socket to set options for.
level	The protocol level to set options at.
option_name	The option to set.
option_value	The value to set for the option.
option_len	The length of option_value in bytes.

Returns

Zero on success. -1 on error, and sets errno as appropriate.

See also

Socket-level options

IPv4 protocol level options

IPv6 protocol level options

UDP protocol level options

shutdown()

```
int shutdown (
             int socket,
             int how )
```

Shutdown socket send and receive operations.

This function closes a specific socket for the set of specified operations.

Parameters

socket	The socket to shutdown.
how	The type of shutdown.

Return values

0	On success.
-1	On error, sets errno as appropriate.

See also

SHUT_RD

SHUT_WR

SHUT_RDWR

socket()

```
int socket (
             int domain,
             int type,
             int protocol )
```

Create an endpoint for communications.

This function creates an unbound socket for communications with the specified parameters.

Parameters

domain	The domain to create the socket in (i.e, AF_INET).
type	The type of socket to be created (i.e, SOCK_DGRAM).
protocol	The protocol to use with the socket. May be 0 to allow a default to be used.

Returns

A non-negative file descriptor on success. -1 on error, and sets errno as appropriate.

9.123 socket.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    sys/socket.h
00004    Copyright (C) 2006, 2010, 2012, 2017 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   sys/socket.h
00009     \brief  Main sockets header.
00010
00011     This file contains the standard definitions (as directed by the POSIX 2008
00012     spec) for socket-related functionality in the AF_INET and AF_INET6 address
00013     families. This does not include anything related to UNIX domain sockets
00014     and is not guaranteed to have everything that one might have in a
00015     fully-standards compliant implementation of the POSIX spec.
00016
00017     \author Lawrence Sebald
00018 */
00019
00020 #ifndef __SYS_SOCKET_H
00021 #define __SYS_SOCKET_H
00022
00023 #include <sys/cdefs.h>
00024 #include <sys/types.h>
00025 #include <sys/uio.h>
00026
00027 __BEGIN_DECLS
00028
00029 /** \brief  Socket length type. */
00030 typedef __uint32_t socklen_t;
00031
00032 /** \brief  Socket address family type. */
00033 typedef __uint8_t sa_family_t;
00034
00035 /** \brief  Socket address structure.
00036     \headerfile sys/socket.h
00037 */
00038 struct sockaddr {
00039     /** \brief  Address family. */
00040     sa_family_t sa_family;
00041     /** \brief  Address data. */
00042     char        sa_data[];
00043 };
00044
00045 /** \brief  Size of the struct sockaddr_storage.
00046     The size here is chosen for compatibility with anything that may expect the
00047     struct sockaddr_storage to be of size 128. Technically, since there are no
00048     current plans to support AF_UNIX sockets, this could be smaller, but most
00049     implementations make it 128 bytes.
00050 */
00051 #define _SS_MAXSIZE 128
00052
00053 /** \brief  Desired alignment of struct sockaddr_storage. */
00054 #define _SS_ALIGNSIZE (sizeof(__uint64_t))
00055
00056 /** \brief  First padding size used within struct sockaddr_storage. */
00057 #define _SS_PAD1SIZE (_SS_ALIGNSIZE - sizeof(sa_family_t))
```

```
00058
00059 /** \brief  Second padding size used within struct sockaddr_storage. */
00060 #define _SS_PAD2SIZE (_SS_MAXSIZE - (sizeof(sa_family_t) + \
00061                                      _SS_PAD1SIZE + _SS_ALIGNSIZE))
00062
00063 /** \brief  Socket address structure of appropriate size to hold any supported
00064             socket type's addresses.
00065     \headerfile sys/socket.h
00066 */
00067 struct sockaddr_storage {
00068     /** \brief  Address family. */
00069     sa_family_t  ss_family;
00070
00071     /** \brief  First padding field. */
00072     char _ss_pad1[_SS_PAD1SIZE];
00073
00074     /** \brief  Used to force alignment. */
00075     __uint64_t _ss_align;
00076
00077     /** \brief  Second padding field to fill up the space required. */
00078     char _ss_pad2[_SS_PAD2SIZE];
00079 };
00080
00081 /** \brief  Datagram socket type.
00082
00083     This socket type specifies that the socket in question transmits datagrams
00084     that may or may not be reliably transmitted. With IP, this implies using UDP
00085     as the underlying protocol.
00086 */
00087 #define SOCK_DGRAM  1
00088
00089 /** \brief  Stream socket type.
00090
00091     This socket type specifies that the socket in question acts like a stream
00092     or pipe between the two endpoints. Sockets of this type can be assumed to be
00093     reliable -- unless an error is returned, all packets will be received at the
00094     other end in the order they are sent. With IP, this implies using TCP as the
00095     underlying protocol.
00096 */
00097 #define SOCK_STREAM 2
00098
00099 /** \brief  Socket-level option setting.
00100
00101     This constant should be used with the setsockopt() or getsockopt() function
00102     to represent that options should be accessed at the socket level, not the
00103     protocol level.
00104 */
00105 #define SOL_SOCKET  1
00106
00107 /** \defgroup so_opts                   Socket-level options
00108
00109     These are the various socket-level options that can be accessed with the
00110     setsockopt() and getsockopt() functions for the SOL_SOCKET level value.
00111
00112     Not all of these are currently supported, but they are listed for
00113     completeness.
00114
00115     \see                ipv6_opts
00116     \see                ipv4_opts
00117     \see                udp_opts
00118
00119     @{
00120 */
00121 #define SO_ACCEPTCONN   1   /**< \brief Socket is accepting connections (get) */
00122 #define SO_BROADCAST    2   /**< \brief Support broadcasting (get/set) */
00123 #define SO_DEBUG        3   /**< \brief Record debugging info (get/set) */
00124 #define SO_DONTROUTE    4   /**< \brief Do not route packets (get/set) */
00125 #define SO_ERROR        5   /**< \brief Retrieve error status (get) */
00126 #define SO_KEEPALIVE    6   /**< \brief Send keepalive messages (get/set) */
00127 #define SO_LINGER       7   /**< \brief Socket lingers on close (get/set) */
00128 #define SO_OOBINLINE    8   /**< \brief OOB data is inline (get/set) */
00129 #define SO_RCVBUF       9   /**< \brief Receive buffer size (get/set) */
00130 #define SO_RCVLOWAT     10  /**< \brief Receive low-water mark (get/set) */
00131 #define SO_RCVTIMEO     11  /**< \brief Receive timeout value (get/set) */
00132 #define SO_REUSEADDR    12  /**< \brief Reuse local addresses (get/set) */
00133 #define SO_SNDBUF       13  /**< \brief Send buffer size (get/set) */
00134 #define SO_SNDLOWAT     14  /**< \brief Send low-water mark (get/set) */
00135 #define SO_SNDTIMEO     15  /**< \brief Send timeout value (get/set) */
00136 #define SO_TYPE         16  /**< \brief Socket type (get) */
00137 /** @} */
00138
```

```
00139 /** \defgroup msg_flags                Socket message flags
00140
00141     The following flags can be used with the recv(), recvfrom(), send(),
00142     and sendto() functions as the flags parameter.
00143
00144     Note that not all of these are currently supported, but they are listed for
00145     completeness. Those that are unsupported have (U) at the end of their
00146     description. Also, for the time being, the supported flags are only
00147     supported for TCP.
00148
00149     @{
00150 */
00151 #define MSG_CTRUNC      0x01    /**< \brief Control data truncated (U) */
00152 #define MSG_DONTROUTE   0x02    /**< \brief Send without routing (U) */
00153 #define MSG_EOR         0x04    /**< \brief Terminate a record (U) */
00154 #define MSG_OOB         0x08    /**< \brief Out-of-band data (U) */
00155 #define MSG_PEEK        0x10    /**< \brief Leave received data in queue */
00156 #define MSG_TRUNC       0x20    /**< \brief Normal data truncated (U) */
00157 #define MSG_WAITALL     0x40    /**< \brief Attempt to fill read buffer */
00158 #define MSG_DONTWAIT    0x80    /**< \brief Make this call non-blocking (non-standard) */
00159 /** @} */
00160
00161 /** \brief  Unspecified address family. */
00162 #define AF_UNSPEC   0
00163
00164 /** \brief  Internet domain sockets for use with IPv4 addresses. */
00165 #define AF_INET     1
00166
00167 /** \brief  Internet domain sockets for use with IPv6 addresses. */
00168 #define AF_INET6    2
00169
00170 /** \brief  Unspecified protocol family. */
00171 #define PF_UNSPEC   AF_UNSPEC
00172
00173 /** \brief  Protocol family for Internet domain sockets (IPv4). */
00174 #define PF_INET     AF_INET
00175
00176 /** \brief  Protocol family for Internet domain sockets (IPv6). */
00177 #define PF_INET6    AF_INET6
00178
00179 /** \brief  Disable further receive operations. */
00180 #define SHUT_RD   0x00000001
00181
00182 /** \brief  Disable further send operations. */
00183 #define SHUT_WR   0x00000002
00184
00185 /** \brief  Disable further send and receive operations. */
00186 #define SHUT_RDWR (SHUT_RD | SHUT_WR)
00187
00188 /** \brief  Maximum backlog for a listening socket. */
00189 #define SOMAXCONN   32
00190
00191 /** \brief  Accept a new connection on a socket.
00192
00193     This function extracts the first connection on the queue of connections of
00194     the specified socket, creating a new socket with the same protocol and
00195     address family as that socket for communication with the extracted
00196     connection.
00197
00198     \param  socket      A socket created with socket() that has been bound to an
00199                         address with bind() and is listening for connections
00200                         after a call to listen().
00201     \param  address     A pointer to a sockaddr structure where the address of
00202                         the connecting socket will be returned (can be NULL).
00203     \param  address_len A pointer to a socklen_t which specifies the amount of
00204                         space in address on input, and the amount used of the
00205                         space on output.
00206     \return             On success, the non-negative file descriptor of the
00207                         new connection, otherwise -1 and errno will be set to
00208                         the appropriate error value.
00209 */
00210 int accept(int socket, struct sockaddr *address, socklen_t *address_len);
00211
00212 /** \brief  Bind a name to a socket.
00213
00214     This function assigns the socket to a unique name (address).
00215
00216     \param  socket      A socket that is to be bound.
00217     \param  address     A pointer to a sockaddr structure where the name to be
00218                         assigned to the socket resides.
00219     \param  address_len The length of the address structure.
```

```
00220     \retval 0           On success.
00221     \retval -1          On error, sets errno as appropriate.
00222 */
00223 int bind(int socket, const struct sockaddr *address, socklen_t address_len);
00224
00225 /** \brief  Connect a socket.
00226
00227     This function attempts to make a connection to a resource on a connection-
00228     mode socket, or sets/resets the peer address on a connectionless one.
00229
00230     \param  socket      A socket that is to be connected.
00231     \param  address     A pointer to a sockaddr structure where the name of the
00232                         peer resides.
00233     \param  address_len The length of the address structure.
00234     \retval 0           On success.
00235     \retval -1          On error, sets errno as appropriate.
00236 */
00237 int connect(int socket, const struct sockaddr *address, socklen_t address_len);
00238
00239 /** \brief  Listen for socket connections and set the queue length.
00240
00241     This function marks a connection-mode socket for incoming connections.
00242
00243     \param  socket      A connection-mode socket to listen on.
00244     \param  backlog     The number of queue entries.
00245     \retval 0           On success.
00246     \retval -1          On error, sets errno as appropriate.
00247 */
00248 int listen(int socket, int backlog);
00249
00250 /** \brief  Receive a message on a connected socket.
00251
00252     This function receives messages from the peer on a connected socket.
00253
00254     \param  socket      The socket to receive on.
00255     \param  buffer      A pointer to a buffer to store the message in.
00256     \param  length      The length of the buffer.
00257     \param  flags       The type of message reception. Set to 0 for now.
00258     \return             On success, the length of the message in bytes. If no
00259                         messages are available, and the socket has been shut
00260                         down, 0. On error, -1, and sets errno as appropriate.
00261 */
00262 ssize_t recv(int socket, void *buffer, size_t length, int flags);
00263
00264 /** \brief  Receive a message on a socket.
00265
00266     This function receives messages from a peer on a (usually connectionless)
00267     socket.
00268
00269     \param  socket      The socket to receive on.
00270     \param  buffer      A pointer to a buffer to store the message in.
00271     \param  length      The length of the buffer.
00272     \param  flags       The type of message reception. Set to 0 for now.
00273     \param  address     A pointer to a sockaddr structure to store the peer's
00274                         name in.
00275     \param  address_len A pointer to the length of the address structure on
00276                         input, the number of bytes used on output.
00277     \return             On success, the length of the message in bytes. If no
00278                         messages are available, and the socket has been shut
00279                         down, 0. On error, -1, and sets errno as appropriate.
00280 */
00281 ssize_t recvfrom(int socket, void *buffer, size_t length, int flags,
00282                  struct sockaddr *address, socklen_t *address_len);
00283
00284 /** \brief  Send a message on a connected socket.
00285
00286     This function sends messages to the peer on a connected socket.
00287
00288     \param  socket      The socket to send on.
00289     \param  message     A pointer to a buffer with the message to send.
00290     \param  length      The length of the message.
00291     \param  flags       The type of message transmission. Set to 0 for now.
00292     \return             On success, the number of bytes sent. On error, -1,
00293                         and sets errno as appropriate.
00294 */
00295 ssize_t send(int socket, const void *message, size_t length, int flags);
00296
00297 /** \brief  Send a message on a socket.
00298
00299     This function sends messages to the peer on a (usually connectionless)
00300     socket. If used on a connection-mode socket, this function may change the
```

```
00301     peer that the socket is connected to, or it may simply return error.
00302
00303     \param  socket      The socket to send on.
00304     \param  message     A pointer to a buffer with the message to send.
00305     \param  length      The length of the message.
00306     \param  flags       The type of message transmission. Set to 0 for now.
00307     \param  dest_addr   A pointer to a sockaddr structure with the peer's name.
00308     \param  dest_len    The length of dest_addr, in bytes.
00309     \return             On success, the number of bytes sent. On error, -1,
00310                         and sets errno as appropriate.
00311 */
00312 ssize_t sendto(int socket, const void *message, size_t length, int flags,
00313                const struct sockaddr *dest_addr, socklen_t dest_len);
00314
00315 /** \brief  Shutdown socket send and receive operations.
00316
00317     This function closes a specific socket for the set of specified operations.
00318
00319     \param  socket      The socket to shutdown.
00320     \param  how         The type of shutdown.
00321     \retval 0           On success.
00322     \retval -1          On error, sets errno as appropriate.
00323     \see                SHUT_RD
00324     \see                SHUT_WR
00325     \see                SHUT_RDWR
00326 */
00327 int shutdown(int socket, int how);
00328
00329 /** \brief  Create an endpoint for communications.
00330
00331     This function creates an unbound socket for communications with the
00332     specified parameters.
00333
00334     \param  domain      The domain to create the socket in (i.e, AF_INET).
00335     \param  type        The type of socket to be created (i.e, SOCK_DGRAM).
00336     \param  protocol    The protocol to use with the socket. May be 0 to allow
00337                         a default to be used.
00338     \return             A non-negative file descriptor on success. -1 on error,
00339                         and sets errno as appropriate.
00340 */
00341 int socket(int domain, int type, int protocol);
00342
00343 /** \brief  Get socket name.
00344
00345     This function returns the locally bound address information for a specified
00346     socket.
00347
00348     \param  socket          The socket to get the name of.
00349     \param  name            Pointer to a sockaddr structure which will hold the
00350                             resulting address information.
00351     \param  name_len        The amount of space pointed to by name, in bytes.
00352                             On return, this is set to the actual size of the
00353                             returned address information.
00354     \retval -1              On error, sets errno as appropriate.
00355     \retval 0               On success.
00356 */
00357 int getsockname(int socket, struct sockaddr *name, socklen_t *name_len);
00358
00359 /** \brief  Get socket options.
00360
00361     This function retrieves options associated with a socket. This function
00362     shall attempt to retrieve the specified option (at the specified level) from
00363     the given socket.
00364
00365     \param  socket          The socket to get options for.
00366     \param  level           The protocol level to get options at.
00367     \param  option_name     The option to look up.
00368     \param  option_value    Storage for the value of the option.
00369     \param  option_len      The length of option_value on call, and the real
00370                             option length (if less than the original value)
00371                             on return.
00372     \return                 Zero on success. -1 on error, and sets errno as
00373                             appropriate.
00374
00375     \see                    so_opts
00376     \see                    ipv4_opts
00377     \see                    ipv6_opts
00378     \see                    udp_opts
00379 */
00380 int getsockopt(int socket, int level, int option_name, void *option_value,
00381                socklen_t *option_len);
```

```
00382
00383 /** \brief  Set socket options.
00384
00385     This function sets options associated with a socket. This function shall
00386     attempt to set the specified option (at the specified level) from the given
00387     socket.
00388
00389     \param  socket          The socket to set options for.
00390     \param  level           The protocol level to set options at.
00391     \param  option_name     The option to set.
00392     \param  option_value    The value to set for the option.
00393     \param  option_len      The length of option_value in bytes.
00394     \return                 Zero on success. -1 on error, and sets errno as
00395                             appropriate.
00396
00397     \see                    so_opts
00398     \see                    ipv4_opts
00399     \see                    ipv6_opts
00400     \see                    udp_opts
00401 */
00402 int setsockopt(int socket, int level, int option_name, const void *option_value,
00403                socklen_t option_len);
00404
00405 __END_DECLS
00406
00407 #endif /* __SYS_SOCKET_H */
```

9.124 include/sys/stdio.h File Reference

Basic sys/stdio.h file from newlib.

```
#include <kos/dbglog.h>
```

Macros

- #define _flockfile(fp)

  No-op lock on a file.
- #define _funlockfile(fp)

  No-op unlock a file.

9.124.1 Detailed Description

Basic sys/stdio.h file from newlib.

This file simply defines locking for files as no-ops if they aren't defined already.

9.124.2 Macro Definition Documentation

_flockfile

```
#define _flockfile(
              fp )
```

No-op lock on a file.

Parameters

fp	The file to lock.

_funlockfile

```
#define _funlockfile(
              fp )
```

No-op unlock a file.

Parameters

fp	The file to unlock.

9.125 stdio.h

Go to the documentation of this file.

```
00001 /** \file   sys/stdio.h
00002     \brief  Basic sys/stdio.h file from newlib.
00003
00004     This file simply defines locking for files as no-ops if they aren't defined
00005     already.
00006 */
00007
00008 #ifndef _NEWLIB_STDIO_H
00009 #define _NEWLIB_STDIO_H
00010
00011 // Cribbed from newlib sys/stdio.h
00012
00013 /* Internal locking macros, used to protect stdio functions.  In the
00014    general case, expand to nothing. */
00015 #if !defined(_flockfile)
00016 /** \brief  No-op lock on a file
00017     \param  fp              The file to lock.
00018 */
00019 #  define _flockfile(fp)
00020 #endif
00021
00022 #if !defined(_funlockfile)
00023 /** \brief  No-op unlock a file
00024     \param  fp              The file to unlock.
00025 */
00026 #  define _funlockfile(fp)
00027 #endif
00028
00029 // Added for old KOS source compatability
00030 #include <kos/dbglog.h>
00031
00032 #endif /* _NEWLIB_STDIO_H */
```

9.126 include/sys/uio.h File Reference

Header for vector I/O.

```
#include <sys/cdefs.h>
#include <sys/types.h>
```

Data Structures

- struct iovec_t

  I/O vector structure.

Macros

- #define UIO_MAXIOV IOV_MAX

  Old alias for the maximum length of an iovec.

9.126.1 Detailed Description

Header for vector I/O.

This file contains definitions for vector I/O operations, as specified by the POSIX 2008 specification. Full vector-based I/O is not supported for file operations, but the stuff in here is still useful elsewhere.

Author

Lawrence Sebald

9.126.2 Macro Definition Documentation

UIO_MAXIOV

```
#define UIO_MAXIOV IOV_MAX
```

Old alias for the maximum length of an iovec.

9.127 uio.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    sys/uio.h
00004    Copyright (C) 2017 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   sys/uio.h
00009     \brief  Header for vector I/O.
00010
00011     This file contains definitions for vector I/O operations, as specified by
00012     the POSIX 2008 specification. Full vector-based I/O is not supported for
00013     file operations, but the stuff in here is still useful elsewhere.
00014
00015     \author Lawrence Sebald
00016 */
00017
00018 #ifndef __SYS_UIO_H
00019 #define __SYS_UIO_H
00020
00021 #include <sys/cdefs.h>
00022 #include <sys/types.h>
```

```
00023
00024 __BEGIN_DECLS
00025
00026 /** \brief  I/O vector structure
00027     \headerfile sys/uio.h
00028 */
00029 struct iovec {
00030     /** \brief  Base address of memory for I/O. */
00031     void * iov_base;
00032     /** \brief  Size of memory pointed to by iov_base. */
00033     size_t iov_len;
00034 };
00035
00036 /** \brief  Old alias for the maximum length of an iovec. */
00037 #define UIO_MAXIOV IOV_MAX
00038
00039 __END_DECLS
00040
00041 #endif /* __SYS_UIO_H */
```

9.128 include/sys/utsname.h File Reference

Definitions for the uname() function.

```
#include <sys/cdefs.h>
```

Data Structures

- struct utsname

  Kernel name/information structure.

Macros

- #define _UTSNAME_LENGTH 64

Functions

- int uname (struct utsname *n)

  Retrieve version and other similar information about the kernel.

9.128.1 Detailed Description

Definitions for the uname() function.

This file contains the definitions needed for using the uname() function, as directed by the POSIX 2008 standard (aka The Open Group Base Specifications Issue 7).

Author

Lawrence Sebald

9.128.2 Macro Definition Documentation

_UTSNAME_LENGTH

```
#define _UTSNAME_LENGTH 64
```

9.128.3 Function Documentation

uname()

```
int uname (
            struct utsname * n )
```

Retrieve version and other similar information about the kernel.

This function retrieves information about the current version of the kernel that is running, storing it in the provided buffer.

Parameters

n	The buffer to store version information in.

Returns

0 on success, -1 on error (setting errno appropriately).

9.129 utsname.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    sys/utsname.h
00004    Copyright (C) 2018 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   utsname.h
00009     \brief  Definitions for the uname() function.
00010
00011     This file contains the definitions needed for using the uname() function,
00012     as directed by the POSIX 2008 standard (aka The Open Group Base
00013     Specifications Issue 7).
00014
00015     \author Lawrence Sebald
00016 */
00017
00018 #ifndef __SYS_UTSNAME_H
00019 #define __SYS_UTSNAME_H
00020
00021 #include <sys/cdefs.h>
00022
00023 __BEGIN_DECLS
00024
00025 #define _UTSNAME_LENGTH 64
00026
00027 /** \brief  Kernel name/information structure.
00028
00029     This structure contains information about the kernel and is used by the
```

```
00030     uname() function for returning that information to a program.
00031
00032     \headerfile sys/utsname.h
00033  */
00034 struct utsname {
00035     char sysname[_UTSNAME_LENGTH];  /**< \brief OS Name ("KallistiOS"). */
00036     char nodename[_UTSNAME_LENGTH]; /**< \brief Name on network, if any. */
00037     char release[_UTSNAME_LENGTH];  /**< \brief Kernel release ("2.1.0"). */
00038     char version[_UTSNAME_LENGTH];  /**< \brief Kernel version string. */
00039     char machine[_UTSNAME_LENGTH];  /**< \brief Hardware identifier. */
00040 };
00041
00042 /** \brief  Retrieve version and other similar information about the kernel.
00043
00044     This function retrieves information about the current version of the kernel
00045     that is running, storing it in the provided buffer.
00046
00047     \param  n           The buffer to store version information in.
00048     \return             0 on success, -1 on error (setting errno appropriately).
00049 */
00050 int uname(struct utsname *n);
00051
00052 __END_DECLS
00053
00054 #endif /* !__SYS_UTSNAME_H */
```

9.130 include/threads.h File Reference

C11 Threading API.

```
#include <sys/cdefs.h>
#include <time.h>
#include <kos/thread.h>
#include <kos/once.h>
#include <kos/mutex.h>
#include <kos/cond.h>
#include <kos/tls.h>
```

Macros

- #define ONCE_FLAG_INIT KTHREAD_ONCE_INIT

  Macro to initiallize a once_flag object.
- #define TSS_DTOR_ITERATIONS 1

  Maximum number of iterations over TSS destructors.

Return values

C11 Thread function return values

Most of the C11 thread-related functions that return a result code return one of these.

- #define thrd_success 0

  Success.
- #define thrd_error -1

  Uncategorized error.
- #define thrd_timedout -2

  Time out error.
- #define thrd_busy -3

  Resource busy.

- #define thrd_nomem -4
  Out of memory.

Mutex types

C11 mutual exclusion lock types

These are the possible types of mutex locks that C11 allows for. Note that mtx_plain or mtx_recursive can be ORed with mtx_timed as well.

- #define mtx_plain (1 << 0)
  Plain mutex.
- #define mtx_recursive (1 << 1)
  Recursive mutex.
- #define mtx_timed (1 << 2)
  Mutex supporting the mtx_timedlock function.

Typedefs

- typedef kthread_once_t once_flag
  Object type backing call_once.
- typedef mutex_t mtx_t
  C11 mutual exclusion lock type.
- typedef condvar_t cnd_t
  C11 condition variable type.
- typedef kthread_t * thrd_t
  C11 thread identifier type.
- typedef int(* thrd_start_t) (void *)
  C11 thread start function type.
- typedef kthread_key_t tss_t
  C11 thread-specific storage type.
- typedef void(* tss_dtor_t) (void *)
  C11 thread-specific storage destructor type.

Functions

- void call_once (once_flag *flag, void(*func)(void))
  Call a function one time, no matter how many threads try.
- void mtx_destroy (mtx_t *mtx)
  Deinitialize a mutex lock.
- int mtx_init (mtx_t *mtx, int type)
  Initialize a mutex lock.
- int mtx_lock (mtx_t *mtx)
  Lock a mutex lock.
- int mtx_timedlock (mtx_t *__RESTRICT mtx, const struct timespec *__RESTRICT ts)
  Lock a mutex lock with a timeout.
- int mtx_trylock (mtx_t *mtx)
  Attempt to acquire a mutex lock.
- int mtx_unlock (mtx_t *mtx)

Unlock a previously acquired lock.

- int cnd_broadcast (cnd_t *cond)

  Broadcast to all threads locked on a condition variable.
- void cnd_destroy (cnd_t *cond)

  Deinitialize a condition variable.
- int cnd_init (cnd_t *cond)

  Initialize a condition variable.
- int cnd_signal (cnd_t *cond)

  Signal one thread locked on a condition variable.
- int cnd_timedwait (cnd_t *__RESTRICT cond, mtx_t *__RESTRICT mtx, const struct timespec *__RESTRICT ts)

  Wait on a condition variable (with a timeout).
- int cnd_wait (cnd_t *cond, mtx_t *mtx)

  Wait on a condition variable.
- int thrd_create (thrd_t *thr, thrd_start_t func, void *arg)

  Create and start a new thread.
- thrd_t thrd_current (void)

  Return the identifier of the currently running thread.
- int thrd_detach (thrd_t thr)

  Detach a running thread.
- int thrd_equal (thrd_t thr0, thrd_t thr1)

  Compare two threads for equality.
- _Noreturn void thrd_exit (int res)

  Terminate the current thread immediately.
- int thrd_join (thrd_t thr, int *res)

  Join a running thread.
- int thrd_sleep (const struct timespec *duration, struct timespec *remaining)

  Put the currently running thread to sleep.
- void thrd_yield (void)

  Yield the current thread's timeslice.
- int tss_create (tss_t *key, tss_dtor_t dtor)

  Create a thread-specific storage pointer.
- void tss_delete (tss_t key)

  Free resources associated with a thread-specific storage key.
- void * tss_get (tss_t key)

  Retrieve the value associated with a thread-specific storage key.
- int tss_set (tss_t key, void *val)

  Set the value associated with a thread-specific storage key.

9.130.1 Detailed Description

C11 Threading API.

This file contains the definitions needed for using C11 threads. The C11 standard defines a number of threading-related primitives, which we wrap neatly around KOS' built-in threading support here.

If you compile your code with a strict standard set (you use a -std= flag with GCC that doesn't start with gnu), you must use -std=c11 to use this functionality. If you don't pass a -std= flag to GCC, then you're probably fine.

Author

Lawrence Sebald

9.130.2 Macro Definition Documentation

mtx_plain

```
#define mtx_plain (1 << 0)
```

Plain mutex.

mtx_recursive

```
#define mtx_recursive (1 << 1)
```

Recursive mutex.

mtx_timed

```
#define mtx_timed (1 << 2)
```

Mutex supporting the mtx_timedlock function.

ONCE_FLAG_INIT

```
#define ONCE_FLAG_INIT KTHREAD_ONCE_INIT
```

Macro to initiallize a once_flag object.

thrd_busy

```
#define thrd_busy -3
```

Resource busy.

thrd_error

```
#define thrd_error -1
```

Uncategorized error.

thrd_nomem

```
#define thrd_nomem -4
```

Out of memory.

thrd_success

```
#define thrd_success 0
```

Success.

thrd_timedout

```
#define thrd_timedout -2
```

Time out error.

TSS_DTOR_ITERATIONS

```
#define TSS_DTOR_ITERATIONS 1
```

Maximum number of iterations over TSS destructors.

This macro defines the maximum number of iterations that will be performed over the destructors for thread-specific storage objects when a thread terminates.

9.130.3 Typedef Documentation

cnd_t

```
typedef condvar_t cnd_t
```

C11 condition variable type.

This type holds an identifier for a condition variable object that is to be used with C11 threading support.

mtx_t

```
typedef mutex_t mtx_t
```

C11 mutual exclusion lock type.

This type holds an identifier for a mutual exclusion (mutex) lock to be used with C11 threading support.

once_flag

```
typedef kthread_once_t once_flag
```

Object type backing call_once.

This object type holds a flag that is used by the call_once function to call a function one time. It should always be initialized with the ONCE_FLAG_INIT macro.

thrd_start_t

```
typedef int(* thrd_start_t) (void *)
```

C11 thread start function type.

This is a function pointer type representing a function used to begin a thread. The thread exits when the function returns or calls thrd_exit().

thrd_t

```
typedef kthread_t* thrd_t
```

C11 thread identifier type.

This type holds an identifier for a C11 thread.

tss_dtor_t

```
typedef void(* tss_dtor_t) (void *)
```

C11 thread-specific storage destructor type.

This is a function pointer type which describes a destructor for a thread-specific storage object.

tss_t

```
typedef kthread_key_t tss_t
```

C11 thread-specific storage type.

This type holds a thread-specific storage identifier, which allows a value to be associated with it for each and every thread running.

9.130.4 Function Documentation

call_once()

```
void call_once (
            once_flag * flag,
            void(*)(void) func )  [extern]
```

Call a function one time, no matter how many threads try.

This function uses the once_flag object passed in to ensure that a given function is called exactly once, regardless of how many threads attempt to call through the once_flag.

Parameters

flag	The once_flag to run against.
func	The function to call.

cnd_broadcast()

```
int cnd_broadcast (
            cnd_t * cond )  [extern]
```

Broadcast to all threads locked on a condition variable.

This function wakes all threads that are blocked on the condition variable cond at the time of the call. If no threads are currently blocked on cond, this call does nothing.

Parameters

cond	The condition variable to signal.

Return values

thrd_success	On success.
thrd_error	If the request cannot be honored.

cnd_destroy()

```
void cnd_destroy (
            cnd_t * cond )  [extern]
```

Deinitialize a condition variable.

This function cleans up all resources associated with the given condition variable. You must ensure that no threads are currently blocked on the condition variable before calling this function.

Parameters

cond	The condition variable to deinitialize.

Note

Deinitializing a condition variable that is currently being waited on by threads results in undefined behavior.

cnd_init()

```
int cnd_init (
            cnd_t * cond )  [extern]
```

Initialize a condition variable.

This function initializes the specified condition variable for use.

Parameters

cond	The condition variable to signal.

Return values

thrd_success	On success.
thrd_nomem	If memory cannot be allocated for the new condition variable.
thrd_error	If the request cannot be honored for some other reason.

cnd_signal()

```
int cnd_signal (
            cnd_t * cond )  [extern]
```

Signal one thread locked on a condition variable.

This function wakes one thread that is blocked on the condition variable cond at the time of the call. If no threads are currently blocked on cond, this call does nothing.

Parameters

cond	The condition variable to signal.

Return values

thrd_success	On success.
thrd_error	If the request cannot be honored.

cnd_timedwait()

```
int cnd_timedwait (
            cnd_t *__RESTRICT cond,
            mtx_t *__RESTRICT mtx,
            const struct timespec *__RESTRICT ts )  [extern]
```

Wait on a condition variable (with a timeout).

This function puts the calling thread to sleep until either the condition variable is signaled or the timeout specified expires, whichever happens first. The specified mutex must be held by the calling thread when calling this function and will be held by the thread again when it is unblocked.

Parameters

cond	The condition variable to wait on.
mtx	The mutex associated with the condition variable.
ts	The time to wait before timing out.

Return values

thrd_success	On success.
thrd_timedout	If the timeout was reached before the condition variable was signaled.
thrd_error	If the request cannot be honored for some other reason.

Note

Calling this function in an interrupt will result in an error being returned.

Although timeouts are specified in seconds and nanoseconds, the timeout will be rounded up to the nearest millisecond.

cnd_wait()

```
int cnd_wait (
              cnd_t * cond,
              mtx_t * mtx )  [extern]
```

Wait on a condition variable.

This function puts the calling thread to sleep until the condition variable is signaled. The specified mutex must be held by the calling thread when calling this function and will be held by the thread again when it is unblocked.

Parameters

cond	The condition variable to wait on.
mtx	The mutex associated with the condition variable.

Return values

thrd_success	On success.
thrd_error	If the request cannot be honored.

Note

Calling this function in an interrupt will result in an error being returned.

mtx_destroy()

```
void mtx_destroy (
            mtx_t * mtx )  [extern]
```

Deinitialize a mutex lock.

This function deinitializes a mutex lock that was previously created with mtx_init().

Parameters

mtx	The mutex to deinitialize.

mtx_init()

```
int mtx_init (
            mtx_t * mtx,
            int type )  [extern]
```

Initialize a mutex lock.

This function initializes a mutex lock of the given type for later use to protect critical sections of code.

Parameters

mtx	The mutex to initialize.
type	The type of mutex desired

Return values

thrd_success	On success.
thrd_error	If the request could not be honored.

mtx_lock()

```
int mtx_lock (
            mtx_t * mtx )  [extern]
```

Lock a mutex lock.

This function locks the specified mutex, preventing any other threads from obtaining the same lock.

This function will block until the lock can be obtained.

Parameters

mtx	The mutex to lock.

Return values

thrd_success	On success.
thrd_error	If the request could not be honored.

Note

Calling this function in an interrupt will result in an error being returned.

mtx_timedlock()

```
int mtx_timedlock (
            mtx_t *__RESTRICT mtx,
            const struct timespec *__RESTRICT ts )  [extern]
```

Lock a mutex lock with a timeout.

This function locks the specified mutex, assuming that the lock can be obtained in the time period specified.

This function will block until the lock can be obtained or the timeout expires.

Parameters

mtx	The mutex to lock.
ts	The amount of time to wait before timing out.

Return values

thrd_success	On success.
thrd_error	If the request could not be honored for some other reason than a timeout.
thrd_timedout	If the timeout specified passes without obtaining the lock.

Note

Calling this function in an interrupt will result in an error being returned.

Although timeouts are specified in seconds and nanoseconds, the timeout will be rounded up to the nearest millisecond.

mtx_trylock()

```
int mtx_trylock (
            mtx_t * mtx )  [extern]
```

Attempt to acquire a mutex lock.

This function attempts to acquire the specififed mutex and will not block if it cannot be obtained.

Parameters

mtx	The mutex to lock.

Return values

thrd_success	On success.
thrd_busy	If the lock is already locked by a thread.
thrd_error	If the request could not be honored for some other reason.

Note

This function is safe to call in an interrupt.

Always check the return value to ensure that the lock was obtained.

mtx_unlock()

```
int mtx_unlock (
            mtx_t * mtx )  [extern]
```

Unlock a previously acquired lock.

This function releases the specified mutex lock, allowing other threads to acquire it.

Parameters

mtx	The mutex to unlock.

Return values

thrd_success	On success.
thrd_error	If the request cannot be honored.

Note

Unlocking a mutex that was not previously locked by the calling thread results in undefined behavior.

thrd_create()

```
int thrd_create (
            thrd_t * thr,
```

```
            thrd_start_t func,
            void * arg )  [extern]
```

Create and start a new thread.

This function creates a new thread, calling the function specified. The thread is immediately added to the runnable queue of the scheduler and can start at any moment after that. The thread ends when either the function specified returns or when the thread calls thrd_exit().

Parameters

thr	Storage for the thread identifier.
func	The function to call in the new thread.
arg	Argument to pass to the function called.

Return values

thrd_success	On success.
thrd_nomem	If memory cannot be allocated to satisfy the request.
thrd_error	If the request cannot be honored for some other reason.

Note

All threads created are joinable threads by default. That means that in order to free all resources at thread termination, the thread must be joined with the thrd_join() function or detached at some point with thrd_detach().

thrd_current()

```
thrd_t thrd_current (
            void  )  [extern]
```

Return the identifier of the currently running thread.

Returns

The current thread's ID.

thrd_detach()

```
int thrd_detach (
            thrd_t thr )  [extern]
```

Detach a running thread.

This function detaches a thread, which informs the kernel that any resources associated with the thread should be freed immediately when it terminates.

Parameters

thr	The thread to detach.

Return values

thrd_success	On success.
thrd_error	If the request cannot be honored.

Note

Detaching an already detached thread has no effect.

Detaching a thread that has been joined with another thread results in undefined behavior.

thrd_equal()

```
int thrd_equal (
            thrd_t thr0,
            thrd_t thr1 )  [extern]
```

Compare two threads for equality.

This function checks the two two thread identifiers passed in to see if they refer to the same thread.

Parameters

thr0	The first thread to compare.
thr1	The second thread to compare.

Returns

0 if the threads are not equal, nonzero if the threads are equal.

thrd_exit()

```
_Noreturn void thrd_exit (
            int res )  [extern]
```

Terminate the current thread immediately.

This function terminates the calling thread immediately, setting the return value of the thread to the value specified.

Parameters

res	The return value of the thread.

Note

This function will not return.

thrd_join()

```
int thrd_join (
              thrd_t thr,
              int * res )  [extern]
```

Join a running thread.

This function joins the current thread with the specified thread, blocking until that thread has terminated.

Parameters

thr	The thread to join with.
res	Pointer to storage for the result code of the other thread. Set to NULL if you don't care about the result value.

Return values

thrd_success	On success.
thrd_error	If the request cannot be honored.

Note

Joining with a previously detached thread results in undefined behavior.

Joining with a thread that has already been joined to another thread results in undefined behavior.

Calling this function in an interrupt will result in an error being returned.

thrd_sleep()

```
int thrd_sleep (
              const struct timespec * duration,
              struct timespec * remaining )  [extern]
```

Put the currently running thread to sleep.

This function puts the currently running thread to sleep for the specified duration of time, returning any left over time (if interrupted by a signal, for instance) in the second parameter.

Parameters

duration	The amount of time to sleep.
remaining	Any remaining time from the duration that the thread did not sleep for.

Returns

0 if the requested time elapsed, a negative value otherwise.

Note

Although the duration is expressed in seconds and nanoseconds, all sleeping is done in millisecond increments. The value specified will be rounded up if it is not an even number of milliseconds.

KOS does not support signals, so remaining will only ever have a value after the function if there is some sort of error.

Calling this function in an interrupt will result in an error being returned.

thrd_yield()

```
void thrd_yield (
            void  )  [extern]
```

Yield the current thread's timeslice.

This function immediately pauses the current thread's execution and switches to another thread in the ready queue (if there are any threads ready to execute).

Note

Calling this function in an interrupt will not have any effect.

tss_create()

```
int tss_create (
            tss_t * key,
            tss_dtor_t dtor )  [extern]
```

Create a thread-specific storage pointer.

This function creates a thread-specific storage pointer and associates the destructor function supplied with it. After creating the pointer, each thread may associate a piece of data with the key.

Parameters

key	The key to initialize.
dtor	The destructor to associate with the key.

Return values

thrd_success	On success.
thrd_error	On failure.

tss_delete()

```
void tss_delete (
            tss_t key )  [extern]
```

Free resources associated with a thread-specific storage key.

This function releases any resources used by the thread-specific storage key specified. Note that this DOES NOT call any destructors.

Parameters

key	The key to deinitialize.

tss_get()

```
void * tss_get (
            tss_t key )  [extern]
```

Retrieve the value associated with a thread-specific storage key.

This function retrieves the value associated with the specified thread-specific storage key and returns it to the caller. If no value has been set in the current thread, NULL is returned.

Parameters

key	The key to look up the value associated with.

Returns

The value associated with the key.

tss_set()

```
int tss_set (
            tss_t key,
            void * val )  [extern]
```

Set the value associated with a thread-specific storage key.

This function sets the value to be associated with the specified thread-specific storage key, overwriting any previous keys. Note that this DOES NOT call any destructors.

Parameters

key	The key to set the value for.
val	The value to set.

Return values

thrd_success	On success.
thrd_error	If the request cannot be honored.

9.131 threads.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    threads.h
00004    Copyright (C) 2014 Lawrence Sebald
00005 */
00006
00007 /** \file   threads.h
00008     \brief  C11 Threading API.
00009     \ingroup threading
00010
00011     This file contains the definitions needed for using C11 threads. The C11
00012     standard defines a number of threading-related primitives, which we wrap
00013     neatly around KOS' built-in threading support here.
00014
00015     If you compile your code with a strict standard set (you use a -std= flag
00016     with GCC that doesn't start with gnu), you must use -std=c11 to use this
00017     functionality. If you don't pass a -std= flag to GCC, then you're probably
00018     fine.
00019
00020     \author Lawrence Sebald
00021 */
00022
00023 #ifndef __THREADS_H
00024 #define __THREADS_H
00025
00026 #if !defined(__STRICT_ANSI__) || (__STDC_VERSION__ >= 201112L)
00027
00028 #include <sys/cdefs.h>
00029 #include <time.h>
00030
00031 /* Bring in all the threading-related stuff we'll need. */
00032 #include <kos/thread.h>
00033 #include <kos/once.h>
00034 #include <kos/mutex.h>
00035 #include <kos/cond.h>
00036 #include <kos/tls.h>
00037
00038 __BEGIN_DECLS
00039
00040 /** \name Return values
00041     \brief C11 Thread function return values
00042
00043     Most of the C11 thread-related functions that return a result code return
00044     one of these.
00045
00046     @{
00047 */
00048 #define thrd_success    0                   /**< \brief Success */
00049 #define thrd_error      -1                  /**< \brief Uncategorized error */
00050 #define thrd_timedout   -2                  /**< \brief Time out error */
00051 #define thrd_busy       -3                  /**< \brief Resource busy */
00052 #define thrd_nomem      -4                  /**< \brief Out of memory */
00053 /** @} */
00054
00055 /** \brief  Object type backing call_once.
00056
00057     This object type holds a flag that is used by the call_once function to call
00058     a function one time. It should always be initialized with the ONCE_FLAG_INIT
00059     macro.
00060
00061     \headerfile threads.h
00062 */
00063 typedef kthread_once_t once_flag;
00064
```

```
00065 /** \brief  Macro to initiallize a once_flag object. */
00066 #define ONCE_FLAG_INIT KTHREAD_ONCE_INIT
00067
00068 /** \brief  Call a function one time, no matter how many threads try.
00069
00070     This function uses the once_flag object passed in to ensure that a given
00071     function is called exactly once, regardless of how many threads attempt to
00072     call through the once_flag.
00073
00074     \param  flag            The once_flag to run against.
00075     \param  func            The function to call.
00076 */
00077 extern void call_once(once_flag *flag, void (*func)(void));
00078
00079 /** \brief  C11 mutual exclusion lock type.
00080
00081     This type holds an identifier for a mutual exclusion (mutex) lock to be used
00082     with C11 threading support.
00083
00084     \headerfile threads.h
00085 */
00086 typedef mutex_t mtx_t;
00087
00088 /** \name  Mutex types
00089     \brief C11 mutual exclusion lock types
00090
00091     These are the possible types of mutex locks that C11 allows for. Note that
00092     mtx_plain or mtx_recursive can be ORed with mtx_timed as well.
00093
00094     @{
00095 */
00096 #define mtx_plain       (1 « 0)           /**< \brief Plain mutex */
00097 #define mtx_recursive   (1 « 1)           /**< \brief Recursive mutex */
00098 #define mtx_timed       (1 « 2)           /**< \brief Mutex supporting the
00099                                                 mtx_timedlock function. */
00100 /** @} */
00101
00102 /** \brief  Deinitialize a mutex lock.
00103
00104     This function deinitializes a mutex lock that was previously created with
00105     mtx_init().
00106
00107     \param  mtx         The mutex to deinitialize.
00108 */
00109 extern void mtx_destroy(mtx_t *mtx);
00110
00111 /** \brief  Initialize a mutex lock.
00112
00113     This function initializes a mutex lock of the given type for later use to
00114     protect critical sections of code.
00115
00116     \param  mtx             The mutex to initialize.
00117     \param  type            The type of mutex desired
00118     \retval thrd_success    On success.
00119     \retval thrd_error      If the request could not be honored.
00120 */
00121 extern int mtx_init(mtx_t *mtx, int type);
00122
00123 /** \brief  Lock a mutex lock.
00124
00125     This function locks the specified mutex, preventing any other threads from
00126     obtaining the same lock.
00127
00128     This function will block until the lock can be obtained.
00129
00130     \param  mtx             The mutex to lock.
00131     \retval thrd_success    On success.
00132     \retval thrd_error      If the request could not be honored.
00133
00134     \note   Calling this function in an interrupt will result in an error being
00135             returned.
00136 */
00137 extern int mtx_lock(mtx_t *mtx);
00138
00139 /** \brief  Lock a mutex lock with a timeout.
00140
00141     This function locks the specified mutex, assuming that the lock can be
00142     obtained in the time period specified.
00143
00144     This function will block until the lock can be obtained or the timeout
00145     expires.
```

```
00146
00147     \param  mtx             The mutex to lock.
00148     \param  ts              The amount of time to wait before timing out.
00149     \retval thrd_success    On success.
00150     \retval thrd_error      If the request could not be honored for some other
00151                             reason than a timeout.
00152     \retval thrd_timedout   If the timeout specified passes without obtaining
00153                             the lock.
00154
00155     \note   Calling this function in an interrupt will result in an error being
00156             returned.
00157     \note   Although timeouts are specified in seconds and nanoseconds, the
00158             timeout will be rounded up to the nearest millisecond.
00159 */
00160 extern int mtx_timedlock(mtx_t *__RESTRICT mtx,
00161                          const struct timespec *__RESTRICT ts);
00162
00163 /** \brief  Attempt to acquire a mutex lock.
00164
00165     This function attempts to acquire the specififed mutex and will not block if
00166     it cannot be obtained.
00167
00168     \param  mtx             The mutex to lock.
00169     \retval thrd_success    On success.
00170     \retval thrd_busy       If the lock is already locked by a thread.
00171     \retval thrd_error      If the request could not be honored for some other
00172                             reason.
00173
00174     \note   This function is safe to call in an interrupt.
00175     \note   Always check the return value to ensure that the lock was obtained.
00176 */
00177 extern int mtx_trylock(mtx_t *mtx);
00178
00179 /** \brief  Unlock a previously acquired lock.
00180
00181     This function releases the specified mutex lock, allowing other threads to
00182     acquire it.
00183
00184     \param  mtx             The mutex to unlock.
00185     \retval thrd_success    On success.
00186     \retval thrd_error      If the request cannot be honored.
00187
00188     \note   Unlocking a mutex that was not previously locked by the calling
00189             thread results in undefined behavior.
00190 */
00191 extern int mtx_unlock(mtx_t *mtx);
00192
00193 /** \brief  C11 condition variable type.
00194
00195     This type holds an identifier for a condition variable object that is to be
00196     used with C11 threading support.
00197
00198     \headerfile threads.h
00199 */
00200 typedef condvar_t cnd_t;
00201
00202 /** \brief  Broadcast to all threads locked on a condition variable.
00203
00204     This function wakes all threads that are blocked on the condition variable
00205     cond at the time of the call. If no threads are currently blocked on cond,
00206     this call does nothing.
00207
00208     \param  cond            The condition variable to signal.
00209     \retval thrd_success    On success.
00210     \retval thrd_error      If the request cannot be honored.
00211 */
00212 extern int cnd_broadcast(cnd_t *cond);
00213
00214 /** \brief  Deinitialize a condition variable.
00215
00216     This function cleans up all resources associated with the given condition
00217     variable. You must ensure that no threads are currently blocked on the
00218     condition variable before calling this function.
00219
00220     \param  cond            The condition variable to deinitialize.
00221
00222     \note   Deinitializing a condition variable that is currently being waited
00223             on by threads results in undefined behavior.
00224 */
00225 extern void cnd_destroy(cnd_t *cond);
00226
```

```
00227 /** \brief  Initialize a condition variable.
00228
00229     This function initializes the specified condition variable for use.
00230
00231     \param  cond            The condition variable to signal.
00232     \retval thrd_success    On success.
00233     \retval thrd_nomem      If memory cannot be allocated for the new condition
00234                             variable.
00235     \retval thrd_error      If the request cannot be honored for some other
00236                             reason.
00237 */
00238 extern int cnd_init(cnd_t *cond);
00239
00240 /** \brief  Signal one thread locked on a condition variable.
00241
00242     This function wakes one thread that is blocked on the condition variable
00243     cond at the time of the call. If no threads are currently blocked on cond,
00244     this call does nothing.
00245
00246     \param  cond            The condition variable to signal.
00247     \retval thrd_success    On success.
00248     \retval thrd_error      If the request cannot be honored.
00249 */
00250 extern int cnd_signal(cnd_t *cond);
00251
00252 /** \brief  Wait on a condition variable (with a timeout).
00253
00254     This function puts the calling thread to sleep until either the condition
00255     variable is signaled or the timeout specified expires, whichever happens
00256     first. The specified mutex must be held by the calling thread when calling
00257     this function and will be held by the thread again when it is unblocked.
00258
00259     \param  cond            The condition variable to wait on.
00260     \param  mtx             The mutex associated with the condition variable.
00261     \param  ts              The time to wait before timing out.
00262     \retval thrd_success    On success.
00263     \retval thrd_timedout   If the timeout was reached before the condition
00264                             variable was signaled.
00265     \retval thrd_error      If the request cannot be honored for some other
00266                             reason.
00267
00268     \note   Calling this function in an interrupt will result in an error being
00269             returned.
00270     \note   Although timeouts are specified in seconds and nanoseconds, the
00271             timeout will be rounded up to the nearest millisecond.
00272 */
00273 extern int cnd_timedwait(cnd_t *__RESTRICT cond, mtx_t *__RESTRICT mtx,
00274                          const struct timespec *__RESTRICT ts);
00275
00276 /** \brief  Wait on a condition variable.
00277
00278     This function puts the calling thread to sleep until the condition variable
00279     is signaled. The specified mutex must be held by the calling thread when
00280     calling this function and will be held by the thread again when it is
00281     unblocked.
00282
00283     \param  cond            The condition variable to wait on.
00284     \param  mtx             The mutex associated with the condition variable.
00285     \retval thrd_success    On success.
00286     \retval thrd_error      If the request cannot be honored.
00287
00288     \note   Calling this function in an interrupt will result in an error being
00289             returned.
00290 */
00291 extern int cnd_wait(cnd_t *cond, mtx_t *mtx);
00292
00293 /** \brief  C11 thread identifier type.
00294
00295     This type holds an identifier for a C11 thread.
00296
00297     \headerfile threads.h
00298 */
00299 typedef kthread_t *thrd_t;
00300
00301 /** \brief  C11 thread start function type.
00302
00303     This is a function pointer type representing a function used to begin a
00304     thread. The thread exits when the function returns or calls thrd_exit().
00305
00306     \headerfile threads.h
00307 */
```

```
00308 typedef int (*thrd_start_t)(void *);
00309
00310 /** \brief  Create and start a new thread.
00311
00312     This function creates a new thread, calling the function specified. The
00313     thread is immediately added to the runnable queue of the scheduler and can
00314     start at any moment after that. The thread ends when either the function
00315     specified returns or when the thread calls thrd_exit().
00316
00317     \param  thr             Storage for the thread identifier.
00318     \param  func            The function to call in the new thread.
00319     \param  arg             Argument to pass to the function called.
00320     \retval thrd_success    On success.
00321     \retval thrd_nomem      If memory cannot be allocated to satisfy the
00322                             request.
00323     \retval thrd_error      If the request cannot be honored for some other
00324                             reason.
00325
00326     \note   All threads created are joinable threads by default. That means that
00327             in order to free all resources at thread termination, the thread
00328             must be joined with the thrd_join() function or detached at some
00329             point with thrd_detach().
00330 */
00331 extern int thrd_create(thrd_t *thr, thrd_start_t func, void *arg);
00332
00333 /** \brief  Return the identifier of the currently running thread.
00334
00335     \return                 The current thread's ID.
00336 */
00337 extern thrd_t thrd_current(void);
00338
00339 /** \brief  Detach a running thread.
00340
00341     This function detaches a thread, which informs the kernel that any resources
00342     associated with the thread should be freed immediately when it terminates.
00343
00344     \param  thr             The thread to detach.
00345     \retval thrd_success    On success.
00346     \retval thrd_error      If the request cannot be honored.
00347
00348     \note   Detaching an already detached thread has no effect.
00349     \note   Detaching a thread that has been joined with another thread results
00350             in undefined behavior.
00351 */
00352 extern int thrd_detach(thrd_t thr);
00353
00354 /** \brief  Compare two threads for equality.
00355
00356     This function checks the two two thread identifiers passed in to see if they
00357     refer to the same thread.
00358
00359     \param  thr0            The first thread to compare.
00360     \param  thr1            The second thread to compare.
00361     \return                 0 if the threads are not equal, nonzero if the
00362                             threads are equal.
00363 */
00364 extern int thrd_equal(thrd_t thr0, thrd_t thr1);
00365
00366 /** \brief  Terminate the current thread immediately.
00367
00368     This function terminates the calling thread immediately, setting the return
00369     value of the thread to the value specified.
00370
00371     \param  res             The return value of the thread.
00372     \note                   This function will not return.
00373 */
00374 _Noreturn extern void thrd_exit(int res);
00375
00376 /** \brief  Join a running thread.
00377
00378     This function joins the current thread with the specified thread, blocking
00379     until that thread has terminated.
00380
00381     \param  thr             The thread to join with.
00382     \param  res             Pointer to storage for the result code of the other
00383                             thread. Set to NULL if you don't care about the
00384                             result value.
00385     \retval thrd_success    On success.
00386     \retval thrd_error      If the request cannot be honored.
00387
00388     \note   Joining with a previously detached thread results in undefined
```

```
00389            behavior.
00390     \note   Joining with a thread that has already been joined to another thread
00391            results in undefined behavior.
00392     \note   Calling this function in an interrupt will result in an error being
00393            returned.
00394 */
00395 extern int thrd_join(thrd_t thr, int *res);
00396
00397 /** \brief  Put the currently running thread to sleep.
00398
00399     This function puts the currently running thread to sleep for the specified
00400     duration of time, returning any left over time (if interrupted by a signal,
00401     for instance) in the second parameter.
00402
00403     \param  duration        The amount of time to sleep.
00404     \param  remaining       Any remaining time from the duration that the thread
00405                             did not sleep for.
00406     \return                 0 if the requested time elapsed, a negative value
00407                             otherwise.
00408
00409     \note   Although the duration is expressed in seconds and nanoseconds, all
00410             sleeping is done in millisecond increments. The value specified will
00411             be rounded up if it is not an even number of milliseconds.
00412     \note   KOS does not support signals, so remaining will only ever have a
00413             value after the function if there is some sort of error.
00414     \note   Calling this function in an interrupt will result in an error being
00415             returned.
00416 */
00417 extern int thrd_sleep(const struct timespec *duration,
00418                       struct timespec *remaining);
00419
00420 /** \brief  Yield the current thread's timeslice.
00421
00422     This function immediately pauses the current thread's execution and switches
00423     to another thread in the ready queue (if there are any threads ready to
00424     execute).
00425
00426     \note   Calling this function in an interrupt will not have any effect.
00427 */
00428 extern void thrd_yield(void);
00429
00430 /** \brief  Maximum number of iterations over TSS destructors.
00431
00432     This macro defines the maximum number of iterations that will be performed
00433     over the destructors for thread-specific storage objects when a thread
00434     terminates.
00435 */
00436 #define TSS_DTOR_ITERATIONS     1
00437
00438 /** \brief  C11 thread-specific storage type.
00439
00440     This type holds a thread-specific storage identifier, which allows a value
00441     to be associated with it for each and every thread running.
00442
00443     \headerfile threads.h
00444 */
00445 typedef kthread_key_t tss_t;
00446
00447 /** \brief  C11 thread-specific storage destructor type.
00448
00449     This is a function pointer type which describes a destructor for a
00450     thread-specific storage object.
00451
00452     \headerfile threads.h
00453 */
00454 typedef void (*tss_dtor_t)(void *);
00455
00456 /** \brief  Create a thread-specific storage pointer.
00457
00458     This function creates a thread-specific storage pointer and associates the
00459     destructor function supplied with it. After creating the pointer, each
00460     thread may associate a piece of data with the key.
00461
00462     \param  key             The key to initialize.
00463     \param  dtor            The destructor to associate with the key.
00464     \retval thrd_success    On success.
00465     \retval thrd_error      On failure.
00466 */
00467 extern int tss_create(tss_t *key, tss_dtor_t dtor);
00468
00469 /** \brief  Free resources associated with a thread-specific storage key.
```

```
00470
00471     This function releases any resources used by the thread-specific storage
00472     key specified. Note that this DOES NOT call any destructors.
00473
00474     \param  key             The key to deinitialize.
00475 */
00476 extern void tss_delete(tss_t key);
00477
00478 /** \brief  Retrieve the value associated with a thread-specific storage key.
00479
00480     This function retrieves the value associated with the specified
00481     thread-specific storage key and returns it to the caller. If no value has
00482     been set in the current thread, NULL is returned.
00483
00484     \param  key             The key to look up the value associated with.
00485     \return                 The value associated with the key.
00486 */
00487 extern void *tss_get(tss_t key);
00488
00489 /** \brief  Set the value associated with a thread-specific storage key.
00490
00491     This function sets the value to be associated with the specified
00492     thread-specific storage key, overwriting any previous keys. Note that this
00493     DOES NOT call any destructors.
00494
00495     \param  key             The key to set the value for.
00496     \param  val             The value to set.
00497     \retval thrd_success    On success.
00498     \retval thrd_error      If the request cannot be honored.
00499 */
00500 extern int tss_set(tss_t key, void *val);
00501
00502 __END_DECLS
00503
00504 #endif /* !defined(__STRICT_ANSI__) || (__STDC_VERSION__ >= 201112L) */
00505
00506 #endif /* !__THREADS_H */
```

9.132 kernel/arch/dreamcast/include/arch/arch.h File Reference

Dreamcast architecture specific options.

```
#include <kos/cdefs.h>
#include <arch/types.h>
#include <kos/init.h>
```

Macros

- #define _arch_mem_top ((uint32) 0x8d000000)

  Top of memory available, depending on memory size.
- #define PAGESIZE 4096

  Page size (for MMU)
- #define PAGESIZE_BITS 12

  Bits for page size.
- #define PAGEMASK (PAGESIZE - 1)

  Mask for page offset.
- #define page_count ((_arch_mem_top - page_phys_base) / PAGESIZE)

  Page count "variable".
- #define page_phys_base 0x8c010000

  Base address of available physical pages.

- #define HZ 100

  Number of timer ticks per second.
- #define THD_STACK_SIZE 32768

  Default thread stack size.
- #define DEFAULT_VID_MODE DM_640x480

  Default video mode.
- #define DEFAULT_PIXEL_MODE PM_RGB565

  Default pixel mode for video.
- #define DEFAULT_SERIAL_BAUD 115200

  Default serial bitrate.
- #define DEFAULT_SERIAL_FIFO 1

  Default serial FIFO behavior.
- #define ELF_SYM_PREFIX "_"

  Global symbol prefix in ELF files.
- #define ELF_SYM_PREFIX_LEN 1

  Length of global symbol prefix in ELF files.
- #define ARCH_EXIT_RETURN 1

  Return to loader.
- #define ARCH_EXIT_MENU 2

  Return to system menu.
- #define ARCH_EXIT_REBOOT 3

  Reboot the machine.
- #define HW_MEM_16 16777216

  16M retail Dreamcast
- #define HW_MEM_32 33554432

  32M NAOMI/modded Dreamcast
- #define HW_MEMSIZE (_arch_mem_top - 0x8c000000)

  Determine how much memory is installed in current machine.
- #define DBL_MEM (_arch_mem_top - 0x8d000000)

  Use this macro to easily determine if system has 32MB of RAM.
- #define HW_TYPE_RETAIL 0x0

  A retail Dreamcast.
- #define HW_TYPE_SET5 0x9

  A Set5.xx devkit.
- #define HW_REGION_UNKNOWN 0x0

  Unknown region.
- #define HW_REGION_ASIA 0x1

  Japan/Asia (NTSC)
- #define HW_REGION_US 0x4

  North America.
- #define HW_REGION_EUROPE 0xC

  Europe (PAL)
- #define arch_sleep()

  Dreamcast specific sleep mode "function".
- #define arch_get_ret_addr()

  DC specific "function" to get the return address from the current function.
- #define arch_get_fptr()

DC specific "function" to get the frame pointer from the current function.

- #define arch_fptr_ret_addr(fptr) (*((uint32*)fptr))

  Pass in a frame pointer value to get the return address for the given frame.

- #define arch_fptr_next(fptr) (*((uint32*)(fptr+4)))

  Pass in a frame pointer value to get the previous frame pointer for the given frame.

- #define arch_valid_address(ptr) ((ptr_t)(ptr) >= 0x8c010000 && (ptr_t)(ptr) < _arch_mem_top)

  Returns true if the passed address is likely to be valid. Doesn't have to be exact, just a sort of general idea.

Functions

- void arch_panic (const char *str) __noreturn

  Panic function.

- void arch_main (void) __noreturn

  Kernel C-level entry point.

- void arch_set_exit_path (int path)

  Set the exit path.

- void arch_exit (void) __noreturn

  Generic kernel "exit" point.

- void arch_return (int ret_code) __noreturn

  Kernel "return" point.

- void arch_abort (void) __noreturn

  Kernel "abort" point.

- void arch_reboot (void) __noreturn

  Kernel "reboot" call.

- void arch_menu (void) __noreturn

  Kernel "exit to menu" call.

- int mm_init (void)

  Initialize the memory management system.

- void * mm_sbrk (unsigned long increment)

  Request more core memory from the system.

- void arch_real_exit (int ret_code) __noreturn

  Jump back to the bootloader.

- int hardware_sys_init (void)

  Initialize bare-bones hardware systems.

- int hardware_periph_init (void)

  Initialize some peripheral systems.

- void hardware_shutdown (void)

  Shut down hardware that was initted.

- int hardware_sys_mode (int *region)

  Retrieve the system mode of the console in use.

- const char * kos_get_banner (void)

  Retrieve the banner printed at program initialization.

- const char * kos_get_license (void)

  Retrieve the license information for the compiled copy of KOS.

- const char * kos_get_authors (void)

  Retrieve a list of authors and the dates of their contributions.

9.132.1 Detailed Description

Dreamcast architecture specific options.

This file has various architecture specific options defined in it. Also, any functions that start with arch_ are in here.

Author

Megan Potter

9.132.2 Macro Definition Documentation

_arch_mem_top

```
#define _arch_mem_top ((uint32) 0x8d000000)
```

Top of memory available, depending on memory size.

arch_fptr_next

```
#define arch_fptr_next(
              fptr ) (*((uint32*)(fptr+4)))
```

Pass in a frame pointer value to get the previous frame pointer for the given frame.

Parameters

fptr	The frame pointer to look at.

Returns

The previous frame pointer.

arch_fptr_ret_addr

```
#define arch_fptr_ret_addr(
              fptr ) (*((uint32*)fptr))
```

Pass in a frame pointer value to get the return address for the given frame.

Parameters

fptr	The frame pointer to look at.

Returns

The return address of the pointer.

arch_get_fptr

```
#define arch_get_fptr( )
```

Value:

```
        ({ \
        uint32 fp; \
        __asm__ __volatile__("mov       r14,%0\n" \
                             : "=&z" (fp) \
                             : /* no inputs */ \
                             : "memory" ); \
        fp; })
```

DC specific "function" to get the frame pointer from the current function.

Returns

The frame pointer from the current function.

Note

This only works if you don't disable frame pointers.

arch_get_ret_addr

```
#define arch_get_ret_addr( )
```

Value:

```
        ({ \
        uint32 pr; \
        __asm__ __volatile__("sts       pr,%0\n" \
                             : "=&z" (pr) \
                             : /* no inputs */ \
                             : "memory" ); \
        pr; })
```

DC specific "function" to get the return address from the current function.

Returns

The return address of the current function.

arch_sleep

```
#define arch_sleep( )
```

Value:

```
        do { \
        __asm__ __volatile__("sleep"); \
    } while(0)
```

Dreamcast specific sleep mode "function".

arch_valid_address

```
#define arch_valid_address(
             ptr ) ((ptr_t)(ptr) >= 0x8c010000 && (ptr_t)(ptr) < _arch_mem_top)
```

Returns true if the passed address is likely to be valid. Doesn't have to be exact, just a sort of general idea.

Returns

Whether the address is valid or not for normal memory access.

DBL_MEM

```
#define DBL_MEM (_arch_mem_top - 0x8d000000)
```

Use this macro to easily determine if system has 32MB of RAM.

Returns

Non-zero if console has 32MB of RAM, zero otherwise

DEFAULT_PIXEL_MODE

```
#define DEFAULT_PIXEL_MODE PM_RGB565
```

Default pixel mode for video.

DEFAULT_SERIAL_BAUD

```
#define DEFAULT_SERIAL_BAUD 115200
```

Default serial bitrate.

DEFAULT_SERIAL_FIFO

```
#define DEFAULT_SERIAL_FIFO 1
```

Default serial FIFO behavior.

DEFAULT_VID_MODE

```
#define DEFAULT_VID_MODE DM_640x480
```

Default video mode.

ELF_SYM_PREFIX

```
#define ELF_SYM_PREFIX "_"
```

Global symbol prefix in ELF files.

ELF_SYM_PREFIX_LEN

```
#define ELF_SYM_PREFIX_LEN 1
```

Length of global symbol prefix in ELF files.

HW_MEMSIZE

```
#define HW_MEMSIZE (_arch_mem_top - 0x8c000000)
```

Determine how much memory is installed in current machine.

Returns

The total size of system memory in bytes.

HZ

```
#define HZ 100
```

Number of timer ticks per second.

page_count

```
#define page_count ((_arch_mem_top - page_phys_base) / PAGESIZE)
```

Page count "variable".

The number of pages is static, so we can optimize this quite a bit.

page_phys_base

```
#define page_phys_base 0x8c010000
```

Base address of available physical pages.

PAGEMASK

```
#define PAGEMASK (PAGESIZE - 1)
```

Mask for page offset.

PAGESIZE

```
#define PAGESIZE 4096
```

Page size (for MMU)

PAGESIZE_BITS

```
#define PAGESIZE_BITS 12
```

Bits for page size.

THD_STACK_SIZE

```
#define THD_STACK_SIZE 32768
```

Default thread stack size.

9.132.3 Function Documentation

arch_abort()

```
void arch_abort (
            void  )
```

Kernel "abort" point.

Note

This function will never return!

arch_exit()

```
void arch_exit (
            void  )
```

Generic kernel "exit" point.

Note

This function will never return!

arch_main()

```
void arch_main (
            void  )
```

Kernel C-level entry point.

Note

This function will never return!

arch_menu()

```
void arch_menu (
            void  )
```

Kernel "exit to menu" call.

Note

This function will never return!

arch_panic()

```
void arch_panic (
            const char * str )
```

Panic function.

This function will cause a kernel panic, printing the specified message.

Parameters

str	The error message to print.

Note

This function will never return!

arch_real_exit()

```
void arch_real_exit (
            int ret_code )
```

Jump back to the bootloader.

You generally shouldn't use this function, but rather use arch_exit() or exit() instead.

Note

This function will never return!

arch_reboot()

```
void arch_reboot (
             void  )
```

Kernel "reboot" call.

Note

This function will never return!

arch_return()

```
void arch_return (
             int ret_code )
```

Kernel "return" point.

Note

This function will never return!

arch_set_exit_path()

```
void arch_set_exit_path (
             int path )
```

Set the exit path.

The default, if you don't call this, is ARCH_EXIT_RETURN.

Parameters

path	What arch_exit() should do.

See also

Potential exit paths from the kernel on

hardware_periph_init()

```
int hardware_periph_init (
            void  )
```

Initialize some peripheral systems.

This will be done automatically for you on start by the default arch_main(), so you shouldn't have to deal with this yourself.

Return values

0	On success (no error conditions defined).

hardware_shutdown()

```
void hardware_shutdown (
            void  )
```

Shut down hardware that was initted.

This function will shut down anything initted with hardware_sys_init() and hardware_periph_init(). This will be done for you automatically by the various exit points, so you shouldn't have to do this yourself.

hardware_sys_init()

```
int hardware_sys_init (
            void  )
```

Initialize bare-bones hardware systems.

This will be done automatically for you on start by the default arch_main(), so you shouldn't have to deal with this yourself.

Return values

0	On success (no error conditions defined).

hardware_sys_mode()

```
int hardware_sys_mode (
            int * region )
```

Retrieve the system mode of the console in use.

This function retrieves the system mode register of the console that is in use. This register details the actual system type in use (and in some system types the region of the device).

Parameters

region	On return, the region code (one of the Region codes) of the device if the console type allows reading it through the system mode register – otherwise, you must retrieve the region from the flashrom.

Returns

The console type (one of the Console types).

kos_get_authors()

```
const char * kos_get_authors (
             void  )
```

Retrieve a list of authors and the dates of their contributions.

This function retrieves the copyright information for the version of KOS in use. This function can be used to add such information to the credits of programs using KOS to give the appropriate credit to those that have worked on KOS.

Remember, you do need to give credit where credit is due, and this is an easy way to do so. ;-)

Return values

A	pointer to the authors' copyright information.

kos_get_banner()

```
const char * kos_get_banner (
             void  )
```

Retrieve the banner printed at program initialization.

This function retrieves the banner string that is printed at initialization time by the kernel. This contains the version of KOS in use and basic information about the environment in which it was compiled.

Return values

A	pointer to the banner string.

kos_get_license()

```
const char * kos_get_license (
             void  )
```

Retrieve the license information for the compiled copy of KOS.

This function retrieves a string containing the license terms that the version of KOS in use is distributed under. This can be used to easily add information to your program to be displayed at runtime.

Return values

A	pointer to the license terms.

mm_init()

```
int mm_init (
             void  )
```

Initialize the memory management system.

Return values

0	On success (no error conditions defined).

mm_sbrk()

```
void * mm_sbrk (
             unsigned long increment )
```

Request more core memory from the system.

Parameters

increment	The number of bytes requested.

Returns

A pointer to the memory.

Note

This function will panic if no memory is available.

9.133 arch.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
```

```
00003    arch/dreamcast/include/arch.h
00004    Copyright (C) 2001 Megan Potter
00005    Copyright (C) 2013, 2020 Lawrence Sebald
00006
00007 */
00008
00009 /** \file   arch/arch.h
00010     \brief  Dreamcast architecture specific options.
00011
00012     This file has various architecture specific options defined in it. Also, any
00013     functions that start with arch_ are in here.
00014
00015     \author Megan Potter
00016 */
00017
00018 #ifndef __ARCH_ARCH_H
00019 #define __ARCH_ARCH_H
00020
00021 #include <kos/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 #include <arch/types.h>
00025
00026 /** \brief  Top of memory available, depending on memory size. */
00027 #ifdef __KOS_GCC_32MB__
00028 extern uint32 _arch_mem_top;
00029 #else
00030 #pragma message "Outdated toolchain: not patched for 32MB support, limiting "\
00031     "KOS to 16MB-only behavior to retain maximum compatibility. Please "\
00032     "update your toolchain."
00033 #define _arch_mem_top   ((uint32) 0x8d000000)
00034 #endif
00035
00036 #define PAGESIZE        4096            /**< \brief Page size (for MMU) */
00037 #define PAGESIZE_BITS   12              /**< \brief Bits for page size */
00038 #define PAGEMASK        (PAGESIZE - 1)  /**< \brief Mask for page offset */
00039
00040 /** \brief  Page count "variable".
00041
00042     The number of pages is static, so we can optimize this quite a bit. */
00043 #define page_count      ((_arch_mem_top - page_phys_base) / PAGESIZE)
00044
00045 /** \brief  Base address of available physical pages. */
00046 #define page_phys_base  0x8c010000
00047
00048 /** \brief  Number of timer ticks per second. */
00049 #define HZ              100
00050
00051 /** \brief  Default thread stack size. */
00052 #define THD_STACK_SIZE  32768
00053
00054 /** \brief  Default video mode. */
00055 #define DEFAULT_VID_MODE    DM_640x480
00056
00057 /** \brief  Default pixel mode for video. */
00058 #define DEFAULT_PIXEL_MODE  PM_RGB565
00059
00060 /** \brief  Default serial bitrate. */
00061 #define DEFAULT_SERIAL_BAUD 115200
00062
00063 /** \brief  Default serial FIFO behavior. */
00064 #define DEFAULT_SERIAL_FIFO 1
00065
00066 /** \brief  Global symbol prefix in ELF files. */
00067 #define ELF_SYM_PREFIX      "_"
00068
00069 /** \brief  Length of global symbol prefix in ELF files. */
00070 #define ELF_SYM_PREFIX_LEN  1
00071
00072 /** \brief  Panic function.
00073
00074     This function will cause a kernel panic, printing the specified message.
00075
00076     \param  str             The error message to print.
00077     \note                   This function will never return!
00078 */
00079 void arch_panic(const char *str) __noreturn;
00080
00081 /** \brief  Kernel C-level entry point.
00082     \note                   This function will never return!
00083 */
```

```
00084 void arch_main(void) __noreturn;
00085
00086 /** \defgroup arch_retpaths         Potential exit paths from the kernel on
00087                                      arch_exit()
00088
00089     @{
00090 */
00091 #define ARCH_EXIT_RETURN    1   /**< \brief Return to loader */
00092 #define ARCH_EXIT_MENU      2   /**< \brief Return to system menu */
00093 #define ARCH_EXIT_REBOOT    3   /**< \brief Reboot the machine */
00094 /** @} */
00095
00096 /** \brief  Set the exit path.
00097
00098     The default, if you don't call this, is ARCH_EXIT_RETURN.
00099
00100     \param  path            What arch_exit() should do.
00101     \see    arch_retpaths
00102 */
00103 void arch_set_exit_path(int path);
00104
00105 /** \brief  Generic kernel "exit" point.
00106     \note                   This function will never return!
00107 */
00108 void arch_exit(void) __noreturn;
00109
00110 /** \brief  Kernel "return" point.
00111     \note                   This function will never return!
00112 */
00113 void arch_return(int ret_code) __noreturn;
00114
00115 /** \brief  Kernel "abort" point.
00116     \note                   This function will never return!
00117 */
00118 void arch_abort(void) __noreturn;
00119
00120 /** \brief  Kernel "reboot" call.
00121     \note                   This function will never return!
00122 */
00123 void arch_reboot(void) __noreturn;
00124
00125 /** \brief Kernel "exit to menu" call.
00126     \note                   This function will never return!
00127 */
00128 void arch_menu(void) __noreturn;
00129
00130 /** \defgroup hw_memsizes           Console memory sizes
00131     These are the various memory sizes, in bytes, that can be returned by the
00132     HW_MEMSIZE macro.
00133     @{
00134 */
00135 #define HW_MEM_16           16777216    /**< \brief 16M retail Dreamcast */
00136 #define HW_MEM_32           33554432    /**< \brief 32M NAOMI/modded Dreamcast */
00137 /** @} */
00138
00139 /** \brief  Determine how much memory is installed in current machine.
00140     \return The total size of system memory in bytes.
00141 */
00142 #define HW_MEMSIZE (_arch_mem_top - 0x8c000000)
00143
00144 /** \brief Use this macro to easily determine if system has 32MB of RAM.
00145     \return Non-zero if console has 32MB of RAM, zero otherwise
00146 */
00147 #define DBL_MEM (_arch_mem_top - 0x8d000000)
00148
00149 /* These are in mm.c */
00150 /** \brief  Initialize the memory management system.
00151     \retval 0               On success (no error conditions defined).
00152 */
00153 int mm_init(void);
00154
00155 /** \brief  Request more core memory from the system.
00156     \param  increment       The number of bytes requested.
00157     \return                 A pointer to the memory.
00158     \note                   This function will panic if no memory is available.
00159 */
00160 void * mm_sbrk(unsigned long increment);
00161
00162 /* Bring in the init flags for compatibility with old code that expects them
00163    here. */
00164 #include <kos/init.h>
```

```
00165
00166 /* Dreamcast-specific arch init things */
00167 /** \brief  Jump back to the bootloader.
00168
00169     You generally shouldn't use this function, but rather use arch_exit() or
00170     exit() instead.
00171
00172     \note                   This function will never return!
00173 */
00174 void arch_real_exit(int ret_code) __noreturn;
00175
00176 /** \brief  Initialize bare-bones hardware systems.
00177
00178     This will be done automatically for you on start by the default arch_main(),
00179     so you shouldn't have to deal with this yourself.
00180
00181     \retval 0               On success (no error conditions defined).
00182 */
00183 int hardware_sys_init(void);
00184
00185 /** \brief  Initialize some peripheral systems.
00186
00187     This will be done automatically for you on start by the default arch_main(),
00188     so you shouldn't have to deal with this yourself.
00189
00190     \retval 0               On success (no error conditions defined).
00191 */
00192 int hardware_periph_init(void);
00193
00194 /** \brief  Shut down hardware that was initted.
00195
00196     This function will shut down anything initted with hardware_sys_init() and
00197     hardware_periph_init(). This will be done for you automatically by the
00198     various exit points, so you shouldn't have to do this yourself.
00199 */
00200 void hardware_shutdown(void);
00201
00202 /** \defgroup hw_consoles           Console types
00203
00204     These are the various console types that can be returned by the
00205     hardware_sys_mode() function.
00206
00207     @{
00208 */
00209 #define HW_TYPE_RETAIL      0x0     /**< \brief A retail Dreamcast. */
00210 #define HW_TYPE_SET5        0x9     /**< \brief A Set5.xx devkit. */
00211 /** @} */
00212
00213 /** \defgroup hw_regions            Region codes
00214
00215     These are the various region codes that can be returned by the
00216     hardware_sys_mode() function. Note that a retail Dreamcast will always
00217     return 0 for the region code. You must read the region of a retail device
00218     from the flashrom.
00219
00220     \see    fr_region
00221     \see    flashrom_get_region()
00222
00223     @{
00224 */
00225 #define HW_REGION_UNKNOWN   0x0     /**< \brief Unknown region. */
00226 #define HW_REGION_ASIA      0x1     /**< \brief Japan/Asia (NTSC) */
00227 #define HW_REGION_US        0x4     /**< \brief North America */
00228 #define HW_REGION_EUROPE    0xC     /**< \brief Europe (PAL) */
00229 /** @} */
00230
00231 /** \brief  Retrieve the system mode of the console in use.
00232
00233     This function retrieves the system mode register of the console that is in
00234     use. This register details the actual system type in use (and in some system
00235     types the region of the device).
00236
00237     \param  region          On return, the region code (one of the
00238                             \ref hw_regions) of the device if the console type
00239                             allows reading it through the system mode register
00240                             -- otherwise, you must retrieve the region from the
00241                             flashrom.
00242     \return                 The console type (one of the \ref hw_consoles).
00243 */
00244 int hardware_sys_mode(int *region);
00245
```

```
00246 /* These three aught to be in their own header file at some point, but for now,
00247    they'll stay here. */
00248
00249 /** \brief  Retrieve the banner printed at program initialization.
00250
00251     This function retrieves the banner string that is printed at initialization
00252     time by the kernel. This contains the version of KOS in use and basic
00253     information about the environment in which it was compiled.
00254
00255     \retval                 A pointer to the banner string.
00256 */
00257 const char *kos_get_banner(void);
00258
00259 /** \brief  Retrieve the license information for the compiled copy of KOS.
00260
00261     This function retrieves a string containing the license terms that the
00262     version of KOS in use is distributed under. This can be used to easily add
00263     information to your program to be displayed at runtime.
00264
00265     \retval                 A pointer to the license terms.
00266 */
00267 const char *kos_get_license(void);
00268
00269 /** \brief  Retrieve a list of authors and the dates of their contributions.
00270
00271     This function retrieves the copyright information for the version of KOS in
00272     use. This function can be used to add such information to the credits of
00273     programs using KOS to give the appropriate credit to those that have worked
00274     on KOS.
00275
00276     Remember, you do need to give credit where credit is due, and this is an
00277     easy way to do so. ;-)
00278
00279     \retval                 A pointer to the authors' copyright information.
00280 */
00281 const char *kos_get_authors(void);
00282
00283 /** \brief  Dreamcast specific sleep mode "function". */
00284 #define arch_sleep() do { \
00285         __asm__ __volatile__("sleep"); \
00286     } while(0)
00287
00288 /** \brief  DC specific "function" to get the return address from the current
00289             function.
00290     \return                 The return address of the current function.
00291 */
00292 #define arch_get_ret_addr() ({ \
00293         uint32 pr; \
00294         __asm__ __volatile__("sts       pr,%0\n" \
00295                              : "=&z" (pr) \
00296                              : /* no inputs */ \
00297                              : "memory" ); \
00298         pr; })
00299
00300 /* Please note that all of the following frame pointer macros are ONLY
00301    valid if you have compiled your code WITHOUT -fomit-frame-pointer. These
00302    are mainly useful for getting a stack trace from an error. */
00303
00304 /** \brief  DC specific "function" to get the frame pointer from the current
00305             function.
00306     \return                 The frame pointer from the current function.
00307     \note                   This only works if you don't disable frame pointers.
00308 */
00309 #define arch_get_fptr() ({ \
00310         uint32 fp; \
00311         __asm__ __volatile__("mov       r14,%0\n" \
00312                              : "=&z" (fp) \
00313                              : /* no inputs */ \
00314                              : "memory" ); \
00315         fp; })
00316
00317 /** \brief  Pass in a frame pointer value to get the return address for the
00318             given frame.
00319
00320     \param  fptr            The frame pointer to look at.
00321     \return                 The return address of the pointer.
00322 */
00323 #define arch_fptr_ret_addr(fptr) (*((uint32*)fptr))
00324
00325 /** \brief  Pass in a frame pointer value to get the previous frame pointer for
00326             the given frame.
```

```
00327
00328     \param  fptr            The frame pointer to look at.
00329     \return                 The previous frame pointer.
00330 */
00331 #define arch_fptr_next(fptr) (*((uint32*)(fptr+4)))
00332
00333 /** \brief  Returns true if the passed address is likely to be valid. Doesn't
00334             have to be exact, just a sort of general idea.
00335
00336     \return                 Whether the address is valid or not for normal
00337                             memory access.
00338 */
00339 #define arch_valid_address(ptr) ((ptr_t)(ptr) >= 0x8c010000 && (ptr_t)(ptr) < _arch_mem_top)
00340
00341 __END_DECLS
00342
00343 #endif  /* __ARCH_ARCH_H */
```

9.134 kernel/arch/dreamcast/include/arch/byteorder.h File Reference

Byte-order related macros.

```
#include <sys/cdefs.h>
#include <sys/_types.h>
```

Macros

- #define BYTE_ORDER LITTLE_ENDIAN

  Define the byte-order of the platform in use.
- #define arch_swap16(x)

  Swap the byte order of a 16-bit integer.
- #define arch_swap32(x)

  Swap the byte order of a 32-bit integer.
- #define arch_ntohs(x) arch_swap16(x)

  Convert network-to-host short.
- #define arch_ntohl(x) arch_swap32(x)

  Convert network-to-host long.
- #define arch_htons(x) arch_swap16(x)

  Convert host-to-network short.
- #define arch_htonl(x) arch_swap32(x)

  Convert host-to-network long.

9.134.1 Detailed Description

Byte-order related macros.

This file contains architecture-specific byte-order related macros and/or functions. Each platform should define six macros/functions in this file: arch_swap16, arch_swap32, arch_ntohs, arch_ntohl, arch_htons, and arch_htonl. The first two of these swap the byte order of 16-bit and 32-bit integers, respectively. The other four macros will be used by the kernel to implement the network-related byte order functions.

Author

Lawrence Sebald

9.134.2 Macro Definition Documentation

arch_htonl

```
#define arch_htonl(
              x ) arch_swap32(x)
```

Convert host-to-network long.

This macro converts a value in the host's native byte order to network byte order (big endian). On a little endian system (like the Dreamcast), this should just call arch_swap32(). On a big endian system, this should be a no-op.

Parameters

x	The value to be converted. This should be a uint32, or equivalent.

Returns

The converted value.

arch_htons

```
#define arch_htons(
              x ) arch_swap16(x)
```

Convert host-to-network short.

This macro converts a value in the host's native byte order to network byte order (big endian). On a little endian system (like the Dreamcast), this should just call arch_swap16(). On a big endian system, this should be a no-op.

Parameters

x	The value to be converted. This should be a uint16, or equivalent.

Returns

The converted value.

arch_ntohl

```
#define arch_ntohl(
              x ) arch_swap32(x)
```

Convert network-to-host long.

This macro converts a network byte order (big endian) value to the host's native byte order. On a little endian system (like the Dreamcast), this should just call arch_swap32(). On a big endian system, this should be a no-op.

Parameters

x	The value to be converted. This should be a uint32, or equivalent.

Returns

The converted value.

arch_ntohs

```
#define arch_ntohs(
              x ) arch_swap16(x)
```

Convert network-to-host short.

This macro converts a network byte order (big endian) value to the host's native byte order. On a little endian system (like the Dreamcast), this should just call arch_swap16(). On a big endian system, this should be a no-op.

Parameters

x	The value to be converted. This should be a uint16, or equivalent.

Returns

The converted value.

arch_swap16

```
#define arch_swap16(
              x )
```

Value:

```
    ({ \
    uint16 __x = (x); \
    __asm__ __volatile__("swap.b %0, %0" : "=r" (__x) : "0" (__x)); \
    __x; \
})
```

Swap the byte order of a 16-bit integer.

This macro swaps the byte order of a 16-bit integer in an architecture- defined manner.

Parameters

x	The value to be byte-swapped. This should be a uint16, or equivalent.

Returns

The swapped value.

arch_swap32

```
#define arch_swap32(
                x )
```

Value:

```
    ({ \
    uint32 __x = (x); \
    __asm__ __volatile__("swap.b %0, %0\n\t" \
                         "swap.w %0, %0\n\t" \
                         "swap.b %0, %0\n\t" : "=r"(__x) : "0" (__x)); \
    __x; \
})
```

Swap the byte order of a 32-bit integer.

This macro swaps the byte order of a 32-bit integer in an architecture- defined manner.

Parameters

x	The value to be byte-swapped. This should be a uint32, or equivalent.

Returns

The swapped value.

BYTE_ORDER

```
#define BYTE_ORDER LITTLE_ENDIAN
```

Define the byte-order of the platform in use.

9.135 byteorder.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/arch/byteorder.h
00004    Copyright (C) 2015 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   arch/byteorder.h
00009     \brief  Byte-order related macros.
00010
00011     This file contains architecture-specific byte-order related macros and/or
00012     functions. Each platform should define six macros/functions in this file:
00013     arch_swap16, arch_swap32, arch_ntohs, arch_ntohl, arch_htons, and
00014     arch_htonl. The first two of these swap the byte order of 16-bit and 32-bit
00015     integers, respectively. The other four macros will be used by the kernel to
```

```
00016     implement the network-related byte order functions.
00017
00018     \author Lawrence Sebald
00019 */
00020
00021 #ifndef __ARCH_BYTEORDER_H
00022 #define __ARCH_BYTEORDER_H
00023
00024 #include <sys/cdefs.h>
00025 __BEGIN_DECLS
00026
00027 #include <sys/_types.h>
00028
00029 #ifdef BYTE_ORDER
00030 /* If we've included <arch/types.h>, this might already be defined... */
00031 #undef BYTE_ORDER
00032 #endif
00033
00034 /** \brief  Define the byte-order of the platform in use. */
00035 #define BYTE_ORDER      LITTLE_ENDIAN
00036
00037 /** \brief  Swap the byte order of a 16-bit integer.
00038
00039     This macro swaps the byte order of a 16-bit integer in an architecture-
00040     defined manner.
00041
00042     \param  x           The value to be byte-swapped. This should be a uint16,
00043                         or equivalent.
00044     \return             The swapped value.
00045 */
00046 #define arch_swap16(x) ({ \
00047     uint16 __x = (x); \
00048     __asm__ __volatile__("swap.b %0, %0" : "=r" (__x) : "0" (__x)); \
00049     __x; \
00050 })
00051
00052 /** \brief  Swap the byte order of a 32-bit integer.
00053
00054     This macro swaps the byte order of a 32-bit integer in an architecture-
00055     defined manner.
00056
00057     \param  x           The value to be byte-swapped. This should be a uint32,
00058                         or equivalent.
00059     \return             The swapped value.
00060 */
00061 #define arch_swap32(x) ({ \
00062     uint32 __x = (x); \
00063     __asm__ __volatile__("swap.b %0, %0\n\t" \
00064                          "swap.w %0, %0\n\t" \
00065                          "swap.b %0, %0\n\t" : "=r"(__x) : "0" (__x)); \
00066     __x; \
00067 })
00068
00069 /** \brief  Convert network-to-host short.
00070
00071     This macro converts a network byte order (big endian) value to the host's
00072     native byte order. On a little endian system (like the Dreamcast), this
00073     should just call arch_swap16(). On a big endian system, this should be a
00074     no-op.
00075
00076     \param  x           The value to be converted. This should be a uint16,
00077                         or equivalent.
00078     \return             The converted value.
00079 */
00080 #define arch_ntohs(x) arch_swap16(x)
00081
00082 /** \brief  Convert network-to-host long.
00083
00084     This macro converts a network byte order (big endian) value to the host's
00085     native byte order. On a little endian system (like the Dreamcast), this
00086     should just call arch_swap32(). On a big endian system, this should be a
00087     no-op.
00088
00089     \param  x           The value to be converted. This should be a uint32,
00090                         or equivalent.
00091     \return             The converted value.
00092 */
00093 #define arch_ntohl(x) arch_swap32(x)
00094
00095 /** \brief  Convert host-to-network short.
00096
```

```
00097     This macro converts a value in the host's native byte order to network byte
00098     order (big endian). On a little endian system (like the Dreamcast), this
00099     should just call arch_swap16(). On a big endian system, this should be a
00100     no-op.
00101
00102     \param  x           The value to be converted. This should be a uint16,
00103                         or equivalent.
00104     \return             The converted value.
00105 */
00106 #define arch_htons(x) arch_swap16(x)
00107
00108 /** \brief  Convert host-to-network long.
00109
00110     This macro converts a value in the host's native byte order to network byte
00111     order (big endian). On a little endian system (like the Dreamcast), this
00112     should just call arch_swap32(). On a big endian system, this should be a
00113     no-op.
00114
00115     \param  x           The value to be converted. This should be a uint32,
00116                         or equivalent.
00117     \return             The converted value.
00118 */
00119 #define arch_htonl(x) arch_swap32(x)
00120
00121 __END_DECLS
00122
00123 #endif /* !__ARCH_BYTEORDER_H */
```

9.136 kernel/arch/dreamcast/include/arch/cache.h File Reference

Cache management functionality.

```
#include <sys/cdefs.h>
#include <stdint.h>
#include <arch/types.h>
```

Macros

- #define CPU_CACHE_BLOCK_SIZE 32

  SH4 cache block size.

Functions

- void icache_flush_range (uintptr_t start, size_t count)

  Flush the instruction cache.
- void dcache_inval_range (uintptr_t start, size_t count)

  Invalidate the data/operand cache.
- void dcache_flush_range (uintptr_t start, size_t count)

  Flush the data/operand cache.
- void dcache_flush_all (void)

  Flush all the data/operand cache.
- void dcache_purge_range (uintptr_t start, size_t count)

  Purge the data/operand cache.
- void dcache_purge_all (void)

  Purge all the data/operand cache.

- void dcache_purge_all_with_buffer (uintptr_t start, size_t count)

  Purge all the data/operand cache with buffer.
- static __always_inline void dcache_pref_block (const void *src)

  Prefetch one block to the data/operand cache.
- static __always_inline void dcache_alloc_block (const void *src, uint32_t value)

  Allocate one block of the data/operand cache.

9.136.1 Detailed Description

Cache management functionality.

This file contains definitions for functions that manage the cache in the Dreamcast, including functions to flush, invalidate, purge, prefetch and allocate the caches.

Author

Megan Potter

Ruslan Rostovtsev

Andy Barajas

9.136.2 Macro Definition Documentation

CPU_CACHE_BLOCK_SIZE

```
#define CPU_CACHE_BLOCK_SIZE 32
```

SH4 cache block size.

The size of a cache block.

9.136.3 Function Documentation

dcache_alloc_block()

```
static __always_inline void dcache_alloc_block (
            const void * src,
            uint32_t value )  [static]
```

Allocate one block of the data/operand cache.

This function allocate a block of the data/operand cache.

Parameters

src	The physical address to allocate.
value	The value written to first 4-byte.

dcache_flush_all()

```
void dcache_flush_all (
            void  )
```

Flush all the data/operand cache.

This function flushes all the data/operand cache, forcing a write- back on all of the cache blocks that are marked as dirty.

Note

dcache_flush_range() is faster than dcache_flush_all() if the count param is 66560 or less.

dcache_flush_range()

```
void dcache_flush_range (
            uintptr_t start,
            size_t count )
```

Flush the data/operand cache.

This function flushes a range of the data/operand cache, forcing a write- back on all of the data in the specified range. This does not invalidate the cache in the process (meaning the blocks will still be in the cache, just not marked as dirty after this has completed). If you wish to invalidate the cache as well, call dcache_inval_range() after calling this function or use dcache_purge_range() instead of dcache_flush_range().

Parameters

start	The physical address to begin flushing at.
count	The number of bytes to flush.

dcache_inval_range()

```
void dcache_inval_range (
            uintptr_t start,
            size_t count )
```

Invalidate the data/operand cache.

This function invalidates a range of the data/operand cache. If you care about the contents of the cache that have not been written back yet, use dcache_flush_range() before using this function.

Parameters

start	The physical address to begin invalidating at.
count	The number of bytes to invalidate.

dcache_pref_block()

```
static __always_inline void dcache_pref_block (
            const void * src )  [static]
```

Prefetch one block to the data/operand cache.

This function prefetch a block of the data/operand cache.

Parameters

src	The physical address to prefetch.

dcache_purge_all()

```
void dcache_purge_all (
            void  )
```

Purge all the data/operand cache.

This function flushes the entire data/operand cache, ensuring that all cache blocks marked as dirty are written back to memory and all cache entries are invalidated. It does not require an additional buffer and is preferred when memory resources are constrained.

Note

dcache_purge_range() is faster than dcache_purge_all() if the count param is 39936 or less.

dcache_purge_all_with_buffer()

```
void dcache_purge_all_with_buffer (
            uintptr_t start,
            size_t count )
```

Purge all the data/operand cache with buffer.

This function performs a purge of all data/operand cache blocks by utilizing an external buffer to speed up the write-back and invalidation process. It is always faster than dcache_purge_all() and is recommended where maximum speed is required.

Note

While this function offers a superior purge rate, it does require the use of a temporary buffer. So use this function if you have an extra 8/16 kb of memory laying around that you can utilize for no other purpose than for this function.

Parameters

start	The physical address for temporary buffer (32-byte aligned)
count	The size of the temporary buffer, which can be either 8 KB or 16 KB, depending on cache configuration - 8 KB buffer with OCRAM enabled, otherwise 16 KB.

dcache_purge_range()

```
void dcache_purge_range (
            uintptr_t start,
            size_t count )
```

Purge the data/operand cache.

This function flushes a range of the data/operand cache, forcing a write- back and then invalidates all of the data in the specified range.

Parameters

start	The physical address to begin purging at.
count	The number of bytes to purge.

icache_flush_range()

```
void icache_flush_range (
            uintptr_t start,
            size_t count )
```

Flush the instruction cache.

This function flushes a range of the instruction cache.

Parameters

start	The physical address to begin flushing at.
count	The number of bytes to flush.

9.137 cache.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/cache.h
00004    Copyright (C) 2001 Megan Potter
00005    Copyright (C) 2014, 2016, 2023 Ruslan Rostovtsev
```

```
00006    Copyright (C) 2023 Andy Barajas
00007 */
00008
00009 /** \file   arch/cache.h
00010     \brief  Cache management functionality.
00011
00012     This file contains definitions for functions that manage the cache in the
00013     Dreamcast, including functions to flush, invalidate, purge, prefetch and
00014     allocate the caches.
00015
00016     \author Megan Potter
00017     \author Ruslan Rostovtsev
00018     \author Andy Barajas
00019 */
00020
00021 #ifndef __ARCH_CACHE_H
00022 #define __ARCH_CACHE_H
00023
00024 #include <sys/cdefs.h>
00025 __BEGIN_DECLS
00026
00027 #include <stdint.h>
00028 #include <arch/types.h>
00029
00030 /** \brief  SH4 cache block size.
00031
00032     The size of a cache block.
00033 */
00034 #define CPU_CACHE_BLOCK_SIZE 32
00035
00036 /** \brief  Flush the instruction cache.
00037
00038     This function flushes a range of the instruction cache.
00039
00040     \param  start           The physical address to begin flushing at.
00041     \param  count           The number of bytes to flush.
00042 */
00043 void icache_flush_range(uintptr_t start, size_t count);
00044
00045 /** \brief  Invalidate the data/operand cache.
00046
00047     This function invalidates a range of the data/operand cache. If you care
00048     about the contents of the cache that have not been written back yet, use
00049     dcache_flush_range() before using this function.
00050
00051     \param  start           The physical address to begin invalidating at.
00052     \param  count           The number of bytes to invalidate.
00053 */
00054 void dcache_inval_range(uintptr_t start, size_t count);
00055
00056 /** \brief  Flush the data/operand cache.
00057
00058     This function flushes a range of the data/operand cache, forcing a write-
00059     back on all of the data in the specified range. This does not invalidate
00060     the cache in the process (meaning the blocks will still be in the cache,
00061     just not marked as dirty after this has completed). If you wish to
00062     invalidate the cache as well, call dcache_inval_range() after calling this
00063     function or use dcache_purge_range() instead of dcache_flush_range().
00064
00065     \param  start           The physical address to begin flushing at.
00066     \param  count           The number of bytes to flush.
00067 */
00068 void dcache_flush_range(uintptr_t start, size_t count);
00069
00070 /** \brief  Flush all the data/operand cache.
00071
00072     This function flushes all the data/operand cache, forcing a write-
00073     back on all of the cache blocks that are marked as dirty.
00074
00075     \note
00076     dcache_flush_range() is faster than dcache_flush_all() if the count
00077     param is 66560 or less.
00078 */
00079 void dcache_flush_all(void);
00080
00081 /** \brief  Purge the data/operand cache.
00082
00083     This function flushes a range of the data/operand cache, forcing a write-
00084     back and then invalidates all of the data in the specified range.
00085
00086     \param  start           The physical address to begin purging at.
```

```
00087     \param  count           The number of bytes to purge.
00088 */
00089 void dcache_purge_range(uintptr_t start, size_t count);
00090
00091 /** \brief  Purge all the data/operand cache.
00092
00093     This function flushes the entire data/operand cache, ensuring that all
00094     cache blocks marked as dirty are written back to memory and all cache
00095     entries are invalidated. It does not require an additional buffer and is
00096     preferred when memory resources are constrained.
00097
00098     \note
00099     dcache_purge_range() is faster than dcache_purge_all() if the count
00100     param is 39936 or less.
00101 */
00102 void dcache_purge_all(void);
00103
00104 /** \brief  Purge all the data/operand cache with buffer.
00105
00106     This function performs a purge of all data/operand cache blocks by
00107     utilizing an external buffer to speed up the write-back and invalidation
00108     process. It is always faster than dcache_purge_all() and is recommended
00109     where maximum speed is required.
00110
00111     \note While this function offers a superior purge rate, it does require
00112     the use of a temporary buffer. So use this function if you have an extra
00113     8/16 kb of memory laying around that you can utilize for no other purpose
00114     than for this function.
00115
00116     \param  start           The physical address for temporary buffer (32-byte
00117                             aligned)
00118     \param  count           The size of the temporary buffer, which can be
00119                             either 8 KB or 16 KB, depending on cache
00120                             configuration - 8 KB buffer with OCRAM enabled,
00121                             otherwise 16 KB.
00122
00123 */
00124 void dcache_purge_all_with_buffer(uintptr_t start, size_t count);
00125
00126 /** \brief  Prefetch one block to the data/operand cache.
00127
00128     This function prefetch a block of the data/operand cache.
00129
00130     \param  src             The physical address to prefetch.
00131 */
00132 static __always_inline void dcache_pref_block(const void *src) {
00133     __asm__ __volatile__("pref @%0\n"
00134                          :
00135                          : "r" (src)
00136                          : "memory"
00137     );
00138 }
00139
00140 /** \brief  Allocate one block of the data/operand cache.
00141
00142     This function allocate a block of the data/operand cache.
00143
00144     \param  src             The physical address to allocate.
00145     \param  value           The value written to first 4-byte.
00146 */
00147 static __always_inline void dcache_alloc_block(const void *src, uint32_t value) {
00148     __asm__ __volatile__ ("movca.l r0, @%0\n\t"
00149                          :
00150                          : "r" (src), "z" (value)
00151                          : "memory"
00152     );
00153 }
00154
00155
00156 __END_DECLS
00157
00158 #endif  /* __ARCH_CACHE_H */
```

9.138 kernel/arch/dreamcast/include/arch/exec.h File Reference

Program execution.

```
#include <sys/cdefs.h>
```

Functions

- void arch_exec_at (const void *image, uint32 length, uint32 address) __noreturn

  Replace the currently running binary.
- void arch_exec (const void *image, uint32 length) __noreturn

  Replace the currently running binary at the default address.

9.138.1 Detailed Description

Program execution.

This file contains functions that allow you to replace the currently running program with another binary that has already been loaded into memory. Doing so is expected to replace the currently running binary in its entirety, and these functions do not return to the calling function.

Author

Megan Potter

9.138.2 Function Documentation

arch_exec()

```
void arch_exec (
            const void * image,
            uint32 length )
```

Replace the currently running binary at the default address.

This is a convenience function for arch_exec_at() that assumes that the binary has been set up with its starting point at the standard location. In the case of the Dreamcast, this is 0xAC010000 (or 0x8C010000, in P1).

Parameters

image	The binary to run (already loaded into RAM).
length	The length of the binary.

arch_exec_at()

```
void arch_exec_at (
            const void * image,
```

```
            uint32 length,
            uint32 address )
```

Replace the currently running binary.

This function will replace the currently running binary with whatever is at the specified address. This function does not return.

Parameters

image	The binary to run (already loaded into RAM).
length	The length of the binary.
address	The address of the binary's starting point.

9.139 exec.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/arch/exec.h
00004    (c)2002 Megan Potter
00005
00006 */
00007
00008 /** \file   arch/exec.h
00009     \brief  Program execution.
00010
00011     This file contains functions that allow you to replace the currently running
00012     program with another binary that has already been loaded into memory. Doing
00013     so is expected to replace the currently running binary in its entirety, and
00014     these functions do not return to the calling function.
00015
00016     \author Megan Potter
00017 */
00018
00019 #ifndef __ARCH_EXEC_H
00020 #define __ARCH_EXEC_H
00021
00022 #include <sys/cdefs.h>
00023 __BEGIN_DECLS
00024
00025 /** \brief  Replace the currently running binary.
00026
00027     This function will replace the currently running binary with whatever is
00028     at the specified address. This function does not return.
00029
00030     \param  image           The binary to run (already loaded into RAM).
00031     \param  length          The length of the binary.
00032     \param  address         The address of the binary's starting point.
00033 */
00034 void arch_exec_at(const void *image, uint32 length, uint32 address) __noreturn;
00035
00036 /** \brief  Replace the currently running binary at the default address.
00037
00038     This is a convenience function for arch_exec_at() that assumes that the
00039     binary has been set up with its starting point at the standard location.
00040     In the case of the Dreamcast, this is 0xAC010000 (or 0x8C010000, in P1).
00041
00042     \param  image           The binary to run (already loaded into RAM).
00043     \param  length          The length of the binary.
00044 */
00045 void arch_exec(const void *image, uint32 length) __noreturn;
00046
00047 __END_DECLS
00048
00049 #endif  /* __ARCH_EXEC_H */
00050
```

9.140 kernel/arch/dreamcast/include/arch/gdb.h File Reference

GNU Debugger support.

```
#include <sys/cdefs.h>
```

Functions

- void gdb_init (void)
  Initialize the GDB stub.
- void gdb_breakpoint (void)
  Manually raise a GDB breakpoint.

9.140.1 Detailed Description

GNU Debugger support.

This file contains functions to set up and utilize GDB with KallistiOS.

Author

Megan Potter

9.140.2 Function Documentation

gdb_breakpoint()

```
void gdb_breakpoint (
            void  )
```

Manually raise a GDB breakpoint.

This function manually raises a GDB breakpoint at the current location in the code, allowing you to inspect things with GDB at the point where the function is called.

gdb_init()

```
void gdb_init (
            void  )
```

Initialize the GDB stub.

This function initializes GDB support. It should be the first thing you do in your program, when you wish to use GDB for debugging.

9.141 gdb.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/arch/gdb.h
00004    (c)2002 Megan Potter
00005
00006 */
00007
00008 /** \file   arch/gdb.h
00009     \brief  GNU Debugger support.
00010
00011     This file contains functions to set up and utilize GDB with KallistiOS.
00012
00013     \author Megan Potter
00014 */
00015
00016 #ifndef __ARCH_GDB_H
00017 #define __ARCH_GDB_H
00018
00019 #include <sys/cdefs.h>
00020 __BEGIN_DECLS
00021
00022 /** \brief  Initialize the GDB stub.
00023
00024     This function initializes GDB support. It should be the first thing you do
00025     in your program, when you wish to use GDB for debugging.
00026 */
00027 void gdb_init(void);
00028
00029 /** \brief  Manually raise a GDB breakpoint.
00030
00031     This function manually raises a GDB breakpoint at the current location in
00032     the code, allowing you to inspect things with GDB at the point where the
00033     function is called.
00034 */
00035 void gdb_breakpoint(void);
00036
00037 __END_DECLS
00038
00039 #endif  /* __ARCH_GDB_H */
00040
```

9.142 kernel/arch/dreamcast/include/arch/init_flags.h File Reference

Dreamcast-specific initialization-related flags and macros.

```
#include <kos/cdefs.h>
#include <kos/init_base.h>
```

Macros

- #define KOS_INIT_FLAGS_ARCH(flags)

Init Flags

Dreamcast-specific initialization flags.

These are the Dreamcast-specific flags that can be specified with KOS_INIT_FLAGS.

See also

Available flags for initialization

- #define INIT_DEFAULT_ARCH (INIT_MAPLE_ALL)
  Default init flags for the Dreamcast.
- #define INIT_CONTROLLER 0x00001000
  Enable Controller maple driver.
- #define INIT_KEYBOARD 0x00002000
  Enable Keyboard maple driver.
- #define INIT_MOUSE 0x00004000
  Enable Mouse maple driver.
- #define INIT_LIGHTGUN 0x00008000
  Enable Lightgun maple driver.
- #define INIT_VMU 0x00010000
  Enable VMU maple driver.
- #define INIT_PURUPURU 0x00020000
  Enable Puru Puru maple driver.
- #define INIT_SIP 0x00040000
  Enable Sound input maple driver.
- #define INIT_DREAMEYE 0x00080000
  Enable DreamEye maple driver.
- #define INIT_MAPLE_ALL 0x000ff000
  Enable all Maple drivers.
- #define INIT_OCRAM 0x10000000
  Use half of the dcache as RAM.
- #define INIT_NO_DCLOAD 0x20000000
  Disable dcload.

9.142.1 Detailed Description

Dreamcast-specific initialization-related flags and macros.

This file provides initialization-related flags that are specific to the Dreamcast architecture.

See also

kos/init.h

kos/init_base.h

Author

Lawrence Sebald

Megan Potter

Falco Girgis

9.142.2 Macro Definition Documentation

INIT_CONTROLLER

```
#define INIT_CONTROLLER 0x00001000
```

Enable Controller maple driver.

INIT_DEFAULT_ARCH

```
#define INIT_DEFAULT_ARCH (INIT_MAPLE_ALL)
```

Default init flags for the Dreamcast.

INIT_DREAMEYE

```
#define INIT_DREAMEYE 0x00080000
```

Enable DreamEye maple driver.

INIT_KEYBOARD

```
#define INIT_KEYBOARD 0x00002000
```

Enable Keyboard maple driver.

INIT_LIGHTGUN

```
#define INIT_LIGHTGUN 0x00008000
```

Enable Lightgun maple driver.

INIT_MAPLE_ALL

```
#define INIT_MAPLE_ALL 0x000ff000
```

Enable all Maple drivers.

INIT_MOUSE

```
#define INIT_MOUSE 0x00004000
```

Enable Mouse maple driver.

INIT_NO_DCLOAD

```
#define INIT_NO_DCLOAD 0x20000000
```

Disable dcload.

INIT_OCRAM

```
#define INIT_OCRAM 0x10000000
```

Use half of the dcache as RAM.

INIT_PURUPURU

```
#define INIT_PURUPURU 0x00020000
```

Enable Puru Puru maple driver.

INIT_SIP

```
#define INIT_SIP 0x00040000
```

Enable Sound input maple driver.

INIT_VMU

```
#define INIT_VMU 0x00010000
```

Enable VMU maple driver.

KOS_INIT_FLAGS_ARCH

```
#define KOS_INIT_FLAGS_ARCH(
              flags )
```

Value:

```
    KOS_INIT_FLAG(flags, INIT_CONTROLLER, cont_init); \
    KOS_INIT_FLAG(flags, INIT_CONTROLLER, cont_shutdown); \
    KOS_INIT_FLAG(flags, INIT_KEYBOARD, kbd_init); \
    KOS_INIT_FLAG(flags, INIT_KEYBOARD, kbd_shutdown); \
    KOS_INIT_FLAG(flags, INIT_MOUSE, mouse_init); \
    KOS_INIT_FLAG(flags, INIT_MOUSE, mouse_shutdown); \
    KOS_INIT_FLAG(flags, INIT_LIGHTGUN, lightgun_init); \
    KOS_INIT_FLAG(flags, INIT_LIGHTGUN, lightgun_shutdown); \
    KOS_INIT_FLAG(flags, INIT_VMU, vmu_init); \
    KOS_INIT_FLAG(flags, INIT_VMU, vmu_shutdown); \
    KOS_INIT_FLAG(flags, INIT_VMU, vmu_fs_init); \
    KOS_INIT_FLAG(flags, INIT_VMU, vmu_fs_shutdown); \
    KOS_INIT_FLAG(flags, INIT_PURUPURU, purupuru_init); \
    KOS_INIT_FLAG(flags, INIT_PURUPURU, purupuru_shutdown); \
    KOS_INIT_FLAG(flags, INIT_SIP, sip_init); \
    KOS_INIT_FLAG(flags, INIT_SIP, sip_shutdown); \
    KOS_INIT_FLAG(flags, INIT_DREAMEYE, dreameye_init); \
    KOS_INIT_FLAG(flags, INIT_DREAMEYE, dreameye_shutdown); \
    KOS_INIT_FLAG(flags, INIT_MAPLE_ALL, maple_wait_scan); \
    KOS_INIT_FLAG(flags, INIT_MAPLE_ALL, maple_init); \
    KOS_INIT_FLAG(flags, INIT_MAPLE_ALL, maple_shutdown)
```

9.143 init_flags.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/arch/init_flags.h
00004    Copyright (C) 2001 Megan Potter
00005    Copyright (C) 2023 Lawrence Sebald
00006    Copyright (C) 2023 Falco Girgis
00007
00008 */
00009
00010 /** \file   arch/init_flags.h
00011     \brief  Dreamcast-specific initialization-related flags and macros.
00012
00013     This file provides initialization-related flags that are specific to the
00014     Dreamcast architecture.
00015
00016     \sa    kos/init.h
00017     \sa    kos/init_base.h
00018
00019     \author Lawrence Sebald
00020     \author Megan Potter
00021     \author Falco Girgis
00022 */
00023
00024 #ifndef __ARCH_INIT_FLAGS_H
00025 #define __ARCH_INIT_FLAGS_H
00026
00027 #include <kos/cdefs.h>
00028 #include <kos/init_base.h>
00029 __BEGIN_DECLS
00030
00031 /** \defgroup dreamcast_initflags   Dreamcast-specific initialization flags.
00032
00033     \brief Dreamcast-specific KOS_INIT Exports
00034
00035     This macro contains a list of all of the possible DC-specific
00036     exported functions based on their associated initialization flags.
00037
00038     \note
00039     This is not typically used directly and is instead included within
00040     the top-level architecture-independent KOS_INIT_FLAGS() macro.
00041
00042     \param flags    Parts of KOS to initialize.
00043
00044     \sa KOS_INIT_FLAGS()
00045 */
00046 #define KOS_INIT_FLAGS_ARCH(flags) \
00047     KOS_INIT_FLAG(flags, INIT_CONTROLLER, cont_init); \
00048     KOS_INIT_FLAG(flags, INIT_CONTROLLER, cont_shutdown); \
00049     KOS_INIT_FLAG(flags, INIT_KEYBOARD, kbd_init); \
00050     KOS_INIT_FLAG(flags, INIT_KEYBOARD, kbd_shutdown); \
00051     KOS_INIT_FLAG(flags, INIT_MOUSE, mouse_init); \
00052     KOS_INIT_FLAG(flags, INIT_MOUSE, mouse_shutdown); \
00053     KOS_INIT_FLAG(flags, INIT_LIGHTGUN, lightgun_init); \
00054     KOS_INIT_FLAG(flags, INIT_LIGHTGUN, lightgun_shutdown); \
00055     KOS_INIT_FLAG(flags, INIT_VMU, vmu_init); \
00056     KOS_INIT_FLAG(flags, INIT_VMU, vmu_shutdown); \
00057     KOS_INIT_FLAG(flags, INIT_VMU, vmu_fs_init); \
00058     KOS_INIT_FLAG(flags, INIT_VMU, vmu_fs_shutdown); \
00059     KOS_INIT_FLAG(flags, INIT_PURUPURU, purupuru_init); \
00060     KOS_INIT_FLAG(flags, INIT_PURUPURU, purupuru_shutdown); \
00061     KOS_INIT_FLAG(flags, INIT_SIP, sip_init); \
00062     KOS_INIT_FLAG(flags, INIT_SIP, sip_shutdown); \
00063     KOS_INIT_FLAG(flags, INIT_DREAMEYE, dreameye_init); \
00064     KOS_INIT_FLAG(flags, INIT_DREAMEYE, dreameye_shutdown); \
00065     KOS_INIT_FLAG(flags, INIT_MAPLE_ALL, maple_wait_scan); \
00066     KOS_INIT_FLAG(flags, INIT_MAPLE_ALL, maple_init); \
00067     KOS_INIT_FLAG(flags, INIT_MAPLE_ALL, maple_shutdown)
00068
00069
00070 /** \name  Init Flags
00071     \brief Dreamcast-specific initialization flags.
00072
00073     These are the Dreamcast-specific flags that can be specified with
00074     KOS_INIT_FLAGS.
00075
00076     \see    kos_initflags
```

```
00077     @{
00078 */
00079
00080 /** \brief Default init flags for the Dreamcast. */
00081 #define INIT_DEFAULT_ARCH   (INIT_MAPLE_ALL)
00082
00083 #define INIT_CONTROLLER     0x00001000  /**< \brief Enable Controller maple driver */
00084 #define INIT_KEYBOARD       0x00002000  /**< \brief Enable Keyboard maple driver */
00085 #define INIT_MOUSE          0x00004000  /**< \brief Enable Mouse maple driver */
00086 #define INIT_LIGHTGUN       0x00008000  /**< \brief Enable Lightgun maple driver */
00087 #define INIT_VMU            0x00010000  /**< \brief Enable VMU maple driver */
00088 #define INIT_PURUPURU       0x00020000  /**< \brief Enable Puru Puru maple driver */
00089 #define INIT_SIP            0x00040000  /**< \brief Enable Sound input maple driver */
00090 #define INIT_DREAMEYE       0x00080000  /**< \brief Enable DreamEye maple driver */
00091 #define INIT_MAPLE_ALL      0x000ff000  /**< \brief Enable all Maple drivers */
00092
00093 #define INIT_OCRAM          0x10000000  /**< \brief Use half of the dcache as RAM */
00094 #define INIT_NO_DCLOAD      0x20000000  /**< \brief Disable dcload */
00095
00096 /** @} */
00097
00098 __END_DECLS
00099
00100 #endif /* !__ARCH_INIT_FLAGS_H */
```

9.144 kernel/arch/dreamcast/include/arch/irq.h File Reference

Interrupt and exception handling.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Data Structures

- struct irq_context_t

  Architecture-specific structure for holding the processor state.

Macros

- #define REG_BYTE_CNT 256 /* Currently really 228 */

  The number of bytes required to save thread context.
- #define CONTEXT_PC(c) ((c).pc)

  Fetch the program counter from an irq_context_t.
- #define CONTEXT_FP(c) ((c).r[14])

  Fetch the frame pointer from an irq_context_t.
- #define CONTEXT_SP(c) ((c).r[15])

  Fetch the stack pointer from an irq_context_t.
- #define CONTEXT_RET(c) ((c).r[0])

  Fetch the return value from an irq_context_t.
- #define EXC_RESET_POWERON 0x0000

  Power-on reset.
- #define EXC_RESET_MANUAL 0x0020

  Manual reset.
- #define EXC_RESET_UDI 0x0000

Hitachi UDI reset.

- #define EXC_ITLB_MULTIPLE 0x0140

  Instruction TLB multiple hit.
- #define EXC_DTLB_MULTIPLE 0x0140

  Data TLB multiple hit.
- #define EXC_USER_BREAK_PRE 0x01e0

  User break before instruction.
- #define EXC_INSTR_ADDRESS 0x00e0

  Instruction address.
- #define EXC_ITLB_MISS 0x0040

  Instruction TLB miss.
- #define EXC_ITLB_PV 0x00a0

  Instruction TLB protection violation.
- #define EXC_ILLEGAL_INSTR 0x0180

  Illegal instruction.
- #define EXC_SLOT_ILLEGAL_INSTR 0x01a0

  Slot illegal instruction.
- #define EXC_GENERAL_FPU 0x0800

  General FPU exception.
- #define EXC_SLOT_FPU 0x0820

  Slot FPU exception.
- #define EXC_DATA_ADDRESS_READ 0x00e0

  Data address (read)
- #define EXC_DATA_ADDRESS_WRITE 0x0100

  Data address (write)
- #define EXC_DTLB_MISS_READ 0x0040

  Data TLB miss (read)
- #define EXC_DTLB_MISS_WRITE 0x0060

  Data TLB miss (write)
- #define EXC_DTLB_PV_READ 0x00a0

  Data TLB protection violation (read)
- #define EXC_DTLB_PV_WRITE 0x00c0

  Data TLB protection violation (write)
- #define EXC_FPU 0x0120

  FPU exception.
- #define EXC_INITIAL_PAGE_WRITE 0x0080

  Initial page write exception.
- #define EXC_TRAPA 0x0160

  Unconditional trap (trapa)
- #define EXC_USER_BREAK_POST 0x01e0

  User break after instruction.
- #define EXC_NMI 0x01c0

  Nonmaskable interrupt.
- #define EXC_IRQ0 0x0200

  External IRQ request (level 0)
- #define EXC_IRQ1 0x0220

  External IRQ request (level 1)

- #define EXC_IRQ2 0x0240

  External IRQ request (level 2)
- #define EXC_IRQ3 0x0260

  External IRQ request (level 3)
- #define EXC_IRQ4 0x0280

  External IRQ request (level 4)
- #define EXC_IRQ5 0x02a0

  External IRQ request (level 5)
- #define EXC_IRQ6 0x02c0

  External IRQ request (level 6)
- #define EXC_IRQ7 0x02e0

  External IRQ request (level 7)
- #define EXC_IRQ8 0x0300

  External IRQ request (level 8)
- #define EXC_IRQ9 0x0320

  External IRQ request (level 9)
- #define EXC_IRQA 0x0340

  External IRQ request (level 10)
- #define EXC_IRQB 0x0360

  External IRQ request (level 11)
- #define EXC_IRQC 0x0380

  External IRQ request (level 12)
- #define EXC_IRQD 0x03a0

  External IRQ request (level 13)
- #define EXC_IRQE 0x03c0

  External IRQ request (level 14)
- #define EXC_TMU0_TUNI0 0x0400

  TMU0 underflow.
- #define EXC_TMU1_TUNI1 0x0420

  TMU1 underflow.
- #define EXC_TMU2_TUNI2 0x0440

  TMU2 underflow.
- #define EXC_TMU2_TICPI2 0x0460

  TMU2 input capture.
- #define EXC_RTC_ATI 0x0480

  RTC alarm interrupt.
- #define EXC_RTC_PRI 0x04a0

  RTC periodic interrupt.
- #define EXC_RTC_CUI 0x04c0

  RTC carry interrupt.
- #define EXC_SCI_ERI 0x04e0

  SCI Error receive.
- #define EXC_SCI_RXI 0x0500

  SCI Receive ready.
- #define EXC_SCI_TXI 0x0520

  SCI Transmit ready.
- #define EXC_SCI_TEI 0x0540

SCI Transmit error.

- #define EXC_WDT_ITI 0x0560

  Watchdog timer.
- #define EXC_REF_RCMI 0x0580

  Memory refresh compare-match interrupt.
- #define EXC_REF_ROVI 0x05a0

  Memory refresh counter overflow interrupt.
- #define EXC_UDI 0x0600

  Hitachi UDI.
- #define EXC_GPIO_GPIOI 0x0620

  I/O port interrupt.
- #define EXC_DMAC_DMTE0 0x0640

  DMAC transfer end (channel 0)
- #define EXC_DMAC_DMTE1 0x0660

  DMAC transfer end (channel 1)
- #define EXC_DMAC_DMTE2 0x0680

  DMAC transfer end (channel 2)
- #define EXC_DMAC_DMTE3 0x06a0

  DMAC transfer end (channel 3)
- #define EXC_DMA_DMAE 0x06c0

  DMAC address error.
- #define EXC_SCIF_ERI 0x0700

  SCIF Error receive.
- #define EXC_SCIF_RXI 0x0720

  SCIF Receive ready.
- #define EXC_SCIF_BRI 0x0740

  SCIF break.
- #define EXC_SCIF_TXI 0x0760

  SCIF Transmit ready.
- #define EXC_DOUBLE_FAULT 0x0ff0

  Double fault.
- #define EXC_UNHANDLED_EXC 0x0fe0

  Unhandled exception.
- #define TIMER_IRQ EXC_TMU0_TUNI0

  The value of the timer IRQ.

- #define EXC_OFFSET_000 0

  irq_type_offsets Exception type offsets
- #define EXC_OFFSET_100 1

  Offset 0x100.
- #define EXC_OFFSET_400 2

  Offset 0x400.
- #define EXC_OFFSET_600 3

  Offset 0x600.

Typedefs

- typedef uint32 irq_t

  The type of an interrupt identifier.
- typedef void(∗ irq_handler) (irq_t source, irq_context_t ∗context)

  The type of an IRQ handler.

Functions

- int irq_inside_int (void)

  Are we inside an interrupt handler?
- void irq_force_return (void)

  Pretend like we just came in from an interrupt and force a context switch back to the "current" context.
- int irq_set_handler (irq_t source, irq_handler hnd)

  Set or remove an IRQ handler.
- irq_handler irq_get_handler (irq_t source)

  Get the address of the current handler for the IRQ type.
- int trapa_set_handler (irq_t code, irq_handler hnd)

  Set or remove a handler for a trapa code.
- int irq_set_global_handler (irq_handler hnd)

  Set a global exception handler.
- irq_handler irq_get_global_handler (void)

  Get the global exception handler.
- void irq_set_context (irq_context_t ∗regbank)

  Switch out contexts (for interrupt return).
- irq_context_t ∗ irq_get_context (void)

  Get the current IRQ context.
- void irq_create_context (irq_context_t ∗context, uint32 stack_pointer, uint32 routine, uint32 ∗args, int usermode)

  Fill a newly allocated context block for usage with supervisor or user mode.
- int irq_disable (void)

  Disable interrupts.
- void irq_enable (void)

  Enable all interrupts.
- void irq_restore (int v)

  Restore IRQ state.
- int irq_init (void)

  Initialize interrupts.
- void irq_shutdown (void)

  Shutdown interrupts, restoring the state to how it was before irq_init() was called.

9.144.1 Detailed Description

Interrupt and exception handling.

This file contains various definitions and declarations related to handling interrupts and exceptions on the Dreamcast. This level deals with IRQs and exceptions generated on the SH4, versus the asic layer which deals with actually differentiating "external" interrupts.

Author

Megan Potter

See also

dc/asic.h

9.144.2 Macro Definition Documentation

CONTEXT_FP

```
#define CONTEXT_FP(
              c ) ((c).r[14])
```

Fetch the frame pointer from an irq_context_t.

Parameters

c	The context to read from.

Returns

The frame pointer value.

CONTEXT_PC

```
#define CONTEXT_PC(
              c ) ((c).pc)
```

Fetch the program counter from an irq_context_t.

Parameters

c	The context to read from.

Returns

The program counter value.

CONTEXT_RET

```
#define CONTEXT_RET(
              c ) ((c).r[0])
```

Fetch the return value from an irq_context_t.

Parameters

c	The context to read from.

Returns

The return value.

CONTEXT_SP

```
#define CONTEXT_SP(
              c ) ((c).r[15])
```

Fetch the stack pointer from an irq_context_t.

Parameters

c	The context to read from.

Returns

The stack pointer value.

EXC_OFFSET_000

```
#define EXC_OFFSET_000 0
```

irq_type_offsets Exception type offsets

The following are a table of "type offsets" (see the Hitachi PDF). These are the 0x000, 0x100, 0x400, and 0x600 offsets.

Offset 0x000

EXC_OFFSET_100

```
#define EXC_OFFSET_100 1
```

Offset 0x100.

EXC_OFFSET_400

```
#define EXC_OFFSET_400 2
```

Offset 0x400.

EXC_OFFSET_600

```
#define EXC_OFFSET_600 3
```

Offset 0x600.

REG_BYTE_CNT

```
#define REG_BYTE_CNT 256 /* Currently really 228 */
```

The number of bytes required to save thread context.

This should include all general CPU registers, FP registers, and status regs (even if not all of these are actually used).

On the Dreamcast, we need 228 bytes for all of that, but we round it up to a nicer number for sanity.

TIMER_IRQ

```
#define TIMER_IRQ EXC_TMU0_TUNI0
```

The value of the timer IRQ.

9.144.3 Typedef Documentation

irq_handler

```
typedef void(* irq_handler) (irq_t source, irq_context_t *context)
```

The type of an IRQ handler.

Parameters

source	The IRQ that caused the handler to be called.
context	The CPU's context.

irq_t

```
typedef uint32 irq_t
```

The type of an interrupt identifier.

9.144.4 Function Documentation

irq_create_context()

```
void irq_create_context (
            irq_context_t * context,
            uint32 stack_pointer,
            uint32 routine,
            uint32 * args,
            int usermode )
```

Fill a newly allocated context block for usage with supervisor or user mode.

The given parameters will be passed to the called routine (up to the architecture maximum). For the Dreamcast, this maximum is 4.

Parameters

context	The IRQ context to fill in.
stack_pointer	The value to set in the stack pointer.
routine	The address of the program counter for the context.
args	Any arguments to set in the registers. This cannot be NULL, and must have enough values to fill in up to the architecture maximum.
usermode	1 to run the routine in user mode, 0 for supervisor.

irq_disable()

```
int irq_disable (
            void  )
```

Disable interrupts.

This function will disable interrupts, but will leave exceptions enabled.

Returns

The state of IRQs before calling the function. This can be used to restore this state later on with irq_restore().

Referenced by g2_lock().

irq_enable()

```
void irq_enable (
            void  )
```

Enable all interrupts.

This function will enable ALL interrupts, including external ones.

irq_force_return()

```
void irq_force_return (
            void  )
```

Pretend like we just came in from an interrupt and force a context switch back to the "current" context.

Make sure you've called irq_set_context() before doing this!

irq_get_context()

```
irq_context_t * irq_get_context (
            void  )
```

Get the current IRQ context.

This will fetch the processor context prior to the exception handling during an IRQ service routine.

Returns

The current IRQ context.

irq_get_global_handler()

```
irq_handler irq_get_global_handler (
            void  )
```

Get the global exception handler.

Returns

The global exception handler set with irq_set_global_handler(), or NULL if none is set.

irq_get_handler()

```
irq_handler irq_get_handler (
            irq_t source )
```

Get the address of the current handler for the IRQ type.

Parameters

source	The IRQ type to look up.

Returns

A pointer to the procedure to handle the exception.

irq_init()

```
int irq_init (
            void  )
```

Initialize interrupts.

Return values

0	On success (no error conditions defined).

irq_inside_int()

```
int irq_inside_int (
            void  )
```

Are we inside an interrupt handler?

Return values

1	If interrupt handling is in progress.
0	If normal processing is in progress.

irq_restore()

```
void irq_restore (
            int v )
```

Restore IRQ state.

This function will restore the interrupt state to the value specified. This should correspond to a value returned by irq_disable().

Parameters

v	The IRQ state to restore. This should be a value returned by irq_disable().

Referenced by g2_unlock().

irq_set_context()

```
void irq_set_context (
            irq_context_t * regbank )
```

Switch out contexts (for interrupt return).

This function will set the processor state that will be restored when the exception returns.

Parameters

regbank	The values of all registers to be restored.

irq_set_global_handler()

```
int irq_set_global_handler (
            irq_handler hnd )
```

Set a global exception handler.

This function sets a global catch-all handler for all exception types.

Parameters

hnd	A pointer to the procedure to handle the exception.

Return values

0	On success (no error conditions defined).

Note

The specific handler will still be called for the exception if one is set. If not, setting one of these will stop the unhandled exception error.

irq_set_handler()

```
int irq_set_handler (
```

```
            irq_t source,
            irq_handler hnd )
```

Set or remove an IRQ handler.

Passing a NULL value for hnd will remove the current handler, if any.

Parameters

source	The IRQ type to set the handler for (see SH4 exception codes).
hnd	A pointer to a procedure to handle the exception.

Return values

0	On success.
-1	If the source is invalid.

irq_shutdown()

```
void irq_shutdown (
            void  )
```

Shutdown interrupts, restoring the state to how it was before irq_init() was called.

trapa_set_handler()

```
int trapa_set_handler (
            irq_t code,
            irq_handler hnd )
```

Set or remove a handler for a trapa code.

Parameters

code	The value passed to the trapa opcode.
hnd	A pointer to the procedure to handle the trap.

Return values

0	On success.
-1	If the code is invalid (greater than 0xFF).

9.145 irq.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/irq.h
00004    Copyright (C) 2000-2001 Megan Potter
00005
00006 */
00007
00008 /** \file   arch/irq.h
00009     \brief  Interrupt and exception handling.
00010
00011     This file contains various definitions and declarations related to handling
00012     interrupts and exceptions on the Dreamcast. This level deals with IRQs and
00013     exceptions generated on the SH4, versus the asic layer which deals with
00014     actually differentiating "external" interrupts.
00015
00016     \author Megan Potter
00017     \see    dc/asic.h
00018 */
00019
00020 #ifndef __ARCH_IRQ_H
00021 #define __ARCH_IRQ_H
00022
00023 #include <sys/cdefs.h>
00024 __BEGIN_DECLS
00025
00026 #include <arch/types.h>
00027
00028 /** \brief  The number of bytes required to save thread context.
00029
00030     This should include all general CPU registers, FP registers, and status regs
00031     (even if not all of these are actually used).
00032
00033     On the Dreamcast, we need 228 bytes for all of that, but we round it up to a
00034     nicer number for sanity.
00035 */
00036 #define REG_BYTE_CNT 256            /* Currently really 228 */
00037
00038 /** \brief  Architecture-specific structure for holding the processor state.
00039
00040     This structure should hold register values and other important parts of the
00041     processor state. The size of this structure should be less than or equal
00042     to the REG_BYTE_CNT value.
00043
00044     \headerfile arch/irq.h
00045 */
00046 typedef struct irq_context {
00047     uint32  r[16];      /**< \brief 16 general purpose (integer) registers */
00048     uint32  pc;         /**< \brief Program counter */
00049     uint32  pr;         /**< \brief Procedure register (aka return address) */
00050     uint32  gbr;        /**< \brief Global base register */
00051     uint32  vbr;        /**< \brief Vector base register */
00052     uint32  mach;       /**< \brief Multiply-and-accumulate register (high) */
00053     uint32  macl;       /**< \brief Multiply-and-accumulate register (low) */
00054     uint32  sr;         /**< \brief Status register */
00055     uint32  frbank[16]; /**< \brief Secondary floating poing registers */
00056     uint32  fr[16];     /**< \brief Primary floating point registers */
00057     uint32  fpscr;      /**< \brief Floating-point status/control register */
00058     uint32  fpul;       /**< \brief Floatint-point communication register */
00059 } irq_context_t;
00060
00061 /* A couple of architecture independent access macros */
00062 /** \brief  Fetch the program counter from an irq_context_t.
00063     \param  c               The context to read from.
00064     \return                 The program counter value.
00065 */
00066 #define CONTEXT_PC(c)   ((c).pc)
00067
00068 /** \brief  Fetch the frame pointer from an irq_context_t.
00069     \param  c               The context to read from.
00070     \return                 The frame pointer value.
00071 */
00072 #define CONTEXT_FP(c)   ((c).r[14])
00073
00074 /** \brief  Fetch the stack pointer from an irq_context_t.
00075     \param  c               The context to read from.
00076     \return                 The stack pointer value.
```

```
00077 */
00078 #define CONTEXT_SP(c)   ((c).r[15])
00079
00080 /** \brief  Fetch the return value from an irq_context_t.
00081     \param  c               The context to read from.
00082     \return                 The return value.
00083 */
00084 #define CONTEXT_RET(c)  ((c).r[0])
00085
00086 /** \defgroup irq_exception_codes   SH4 exception codes
00087
00088     These are all of the exceptions that can be raised on the SH4, and their
00089     codes. They're divided into several logical groups.
00090
00091     @{
00092 */
00093 /* Dreamcast-specific exception codes.. use these when getting or setting an
00094    exception code value. */
00095
00096 /** \defgroup irq_reset_codes       Reset type
00097
00098     These are exceptions that essentially cause a reset of the system. They
00099     cannot actually be caught by normal means. They will all automatically cause
00100     a branch to address 0xA0000000. These are pretty much fatal.
00101
00102     @{
00103 */
00104 #define EXC_RESET_POWERON   0x0000  /**< \brief Power-on reset */
00105 #define EXC_RESET_MANUAL    0x0020  /**< \brief Manual reset */
00106 #define EXC_RESET_UDI       0x0000  /**< \brief Hitachi UDI reset */
00107 #define EXC_ITLB_MULTIPLE   0x0140  /**< \brief Instruction TLB multiple hit */
00108 #define EXC_DTLB_MULTIPLE   0x0140  /**< \brief Data TLB multiple hit */
00109 /** @} */
00110
00111 /** \defgroup irq_reexec_codes      Re-Execution type
00112
00113     These exceptions will stop the currently processing instruction, and
00114     transition into exception processing. After handling the exception (assuming
00115     that it can be handled by the code), the offending instruction will be
00116     re-executed from the start.
00117
00118     @{
00119 */
00120 #define EXC_USER_BREAK_PRE      0x01e0  /**< \brief User break before instruction */
00121 #define EXC_INSTR_ADDRESS       0x00e0  /**< \brief Instruction address */
00122 #define EXC_ITLB_MISS           0x0040  /**< \brief Instruction TLB miss */
00123 #define EXC_ITLB_PV             0x00a0  /**< \brief Instruction TLB protection violation */
00124 #define EXC_ILLEGAL_INSTR       0x0180  /**< \brief Illegal instruction */
00125 #define EXC_SLOT_ILLEGAL_INSTR  0x01a0  /**< \brief Slot illegal instruction */
00126 #define EXC_GENERAL_FPU         0x0800  /**< \brief General FPU exception */
00127 #define EXC_SLOT_FPU            0x0820  /**< \brief Slot FPU exception */
00128 #define EXC_DATA_ADDRESS_READ   0x00e0  /**< \brief Data address (read) */
00129 #define EXC_DATA_ADDRESS_WRITE  0x0100  /**< \brief Data address (write) */
00130 #define EXC_DTLB_MISS_READ      0x0040  /**< \brief Data TLB miss (read) */
00131 #define EXC_DTLB_MISS_WRITE     0x0060  /**< \brief Data TLB miss (write) */
00132 #define EXC_DTLB_PV_READ        0x00a0  /**< \brief Data TLB protection violation (read) */
00133 #define EXC_DTLB_PV_WRITE       0x00c0  /**< \brief Data TLB protection violation (write) */
00134 #define EXC_FPU                 0x0120  /**< \brief FPU exception */
00135 #define EXC_INITIAL_PAGE_WRITE  0x0080  /**< \brief Initial page write exception */
00136 /** @} */
00137
00138 /** \defgroup irq_completion_codes  Completion type
00139
00140     These exceptions are actually handled in-between instructions, allowing the
00141     instruction that causes them to finish completely. The saved PC thus is the
00142     value of the next instruction.
00143
00144     @{
00145 */
00146 #define EXC_TRAPA           0x0160  /**< \brief Unconditional trap (trapa) */
00147 #define EXC_USER_BREAK_POST 0x01e0  /**< \brief User break after instruction */
00148 /** @} */
00149
00150 /** \defgroup irq_interrupt_codes   Interrupt (completion type)
00151
00152     These exceptions are caused by interrupt requests. These generally are from
00153     peripheral devices, but NMIs, timer interrupts, and DMAC interrupts are also
00154     included here.
00155
00156     \note   Not all of these have any meaning on the Dreamcast. Those that have
00157             no meaning are only included for completeness.
```

```
00158     @{
00159 */
00160 #define EXC_NMI          0x01c0  /**< \brief Nonmaskable interrupt */
00161 #define EXC_IRQ0         0x0200  /**< \brief External IRQ request (level 0) */
00162 #define EXC_IRQ1         0x0220  /**< \brief External IRQ request (level 1) */
00163 #define EXC_IRQ2         0x0240  /**< \brief External IRQ request (level 2) */
00164 #define EXC_IRQ3         0x0260  /**< \brief External IRQ request (level 3) */
00165 #define EXC_IRQ4         0x0280  /**< \brief External IRQ request (level 4) */
00166 #define EXC_IRQ5         0x02a0  /**< \brief External IRQ request (level 5) */
00167 #define EXC_IRQ6         0x02c0  /**< \brief External IRQ request (level 6) */
00168 #define EXC_IRQ7         0x02e0  /**< \brief External IRQ request (level 7) */
00169 #define EXC_IRQ8         0x0300  /**< \brief External IRQ request (level 8) */
00170 #define EXC_IRQ9         0x0320  /**< \brief External IRQ request (level 9) */
00171 #define EXC_IRQA         0x0340  /**< \brief External IRQ request (level 10) */
00172 #define EXC_IRQB         0x0360  /**< \brief External IRQ request (level 11) */
00173 #define EXC_IRQC         0x0380  /**< \brief External IRQ request (level 12) */
00174 #define EXC_IRQD         0x03a0  /**< \brief External IRQ request (level 13) */
00175 #define EXC_IRQE         0x03c0  /**< \brief External IRQ request (level 14) */
00176 #define EXC_TMU0_TUNI0   0x0400  /**< \brief TMU0 underflow */
00177 #define EXC_TMU1_TUNI1   0x0420  /**< \brief TMU1 underflow */
00178 #define EXC_TMU2_TUNI2   0x0440  /**< \brief TMU2 underflow */
00179 #define EXC_TMU2_TICPI2  0x0460  /**< \brief TMU2 input capture */
00180 #define EXC_RTC_ATI      0x0480  /**< \brief RTC alarm interrupt */
00181 #define EXC_RTC_PRI      0x04a0  /**< \brief RTC periodic interrupt */
00182 #define EXC_RTC_CUI      0x04c0  /**< \brief RTC carry interrupt */
00183 #define EXC_SCI_ERI      0x04e0  /**< \brief SCI Error receive */
00184 #define EXC_SCI_RXI      0x0500  /**< \brief SCI Receive ready */
00185 #define EXC_SCI_TXI      0x0520  /**< \brief SCI Transmit ready */
00186 #define EXC_SCI_TEI      0x0540  /**< \brief SCI Transmit error */
00187 #define EXC_WDT_ITI      0x0560  /**< \brief Watchdog timer */
00188 #define EXC_REF_RCMI     0x0580  /**< \brief Memory refresh compare-match interrupt */
00189 #define EXC_REF_ROVI     0x05a0  /**< \brief Memory refresh counter overflow interrupt */
00190 #define EXC_UDI          0x0600  /**< \brief Hitachi UDI */
00191 #define EXC_GPIO_GPIOI   0x0620  /**< \brief I/O port interrupt */
00192 #define EXC_DMAC_DMTE0   0x0640  /**< \brief DMAC transfer end (channel 0) */
00193 #define EXC_DMAC_DMTE1   0x0660  /**< \brief DMAC transfer end (channel 1) */
00194 #define EXC_DMAC_DMTE2   0x0680  /**< \brief DMAC transfer end (channel 2) */
00195 #define EXC_DMAC_DMTE3   0x06a0  /**< \brief DMAC transfer end (channel 3) */
00196 #define EXC_DMA_DMAE     0x06c0  /**< \brief DMAC address error */
00197 #define EXC_SCIF_ERI     0x0700  /**< \brief SCIF Error receive */
00198 #define EXC_SCIF_RXI     0x0720  /**< \brief SCIF Receive ready */
00199 #define EXC_SCIF_BRI     0x0740  /**< \brief SCIF break */
00200 #define EXC_SCIF_TXI     0x0760  /**< \brief SCIF Transmit ready */
00201 /** @} */
00202
00203 /** \brief  Double fault
00204
00205     This exception is completely done in software (not represented on the CPU at
00206     all). Its used for when an exception occurs during an IRQ service routine.
00207 */
00208 #define EXC_DOUBLE_FAULT    0x0ff0
00209
00210 /** \brief  Unhandled exception
00211
00212     This exception is a software-generated exception for a generic unhandled
00213     exception.
00214 */
00215 #define EXC_UNHANDLED_EXC   0x0fe0
00216 /** @} */
00217
00218 /** \brief  irq_type_offsets        Exception type offsets
00219
00220     The following are a table of "type offsets" (see the Hitachi PDF). These are
00221     the 0x000, 0x100, 0x400, and 0x600 offsets.
00222
00223     @{
00224 */
00225 #define EXC_OFFSET_000  0   /**< \brief Offset 0x000 */
00226 #define EXC_OFFSET_100  1   /**< \brief Offset 0x100 */
00227 #define EXC_OFFSET_400  2   /**< \brief Offset 0x400 */
00228 #define EXC_OFFSET_600  3   /**< \brief Offset 0x600 */
00229 /** @} */
00230
00231 /** \brief  The value of the timer IRQ */
00232 #define TIMER_IRQ       EXC_TMU0_TUNI0
00233
00234 /** \brief  The type of an interrupt identifier */
00235 typedef uint32 irq_t;
00236
00237 /** \brief  The type of an IRQ handler
00238     \param  source          The IRQ that caused the handler to be called.
```

```
00239     \param  context         The CPU's context.
00240 */
00241 typedef void (*irq_handler)(irq_t source, irq_context_t *context);
00242
00243 /** \brief  Are we inside an interrupt handler?
00244     \retval 1               If interrupt handling is in progress.
00245     \retval 0               If normal processing is in progress.
00246 */
00247 int irq_inside_int(void);
00248
00249 /** \brief  Pretend like we just came in from an interrupt and force
00250             a context switch back to the "current" context.
00251
00252     Make sure you've called irq_set_context() before doing this!
00253 */
00254 void irq_force_return(void);
00255
00256 /** \brief  Set or remove an IRQ handler.
00257
00258     Passing a NULL value for hnd will remove the current handler, if any.
00259
00260     \param  source          The IRQ type to set the handler for
00261                             (see \ref irq_exception_codes).
00262     \param  hnd             A pointer to a procedure to handle the exception.
00263     \retval 0               On success.
00264     \retval -1              If the source is invalid.
00265 */
00266 int irq_set_handler(irq_t source, irq_handler hnd);
00267
00268 /** \brief  Get the address of the current handler for the IRQ type.
00269     \param  source          The IRQ type to look up.
00270     \return                 A pointer to the procedure to handle the exception.
00271 */
00272 irq_handler irq_get_handler(irq_t source);
00273
00274 /** \brief  Set or remove a handler for a trapa code.
00275     \param  code            The value passed to the trapa opcode.
00276     \param  hnd             A pointer to the procedure to handle the trap.
00277     \retval 0               On success.
00278     \retval -1              If the code is invalid (greater than 0xFF).
00279 */
00280 int trapa_set_handler(irq_t code, irq_handler hnd);
00281
00282 /** \brief  Set a global exception handler.
00283
00284     This function sets a global catch-all handler for all exception types.
00285
00286     \param  hnd             A pointer to the procedure to handle the exception.
00287     \retval 0               On success (no error conditions defined).
00288     \note                   The specific handler will still be called for the
00289                             exception if one is set. If not, setting one of
00290                             these will stop the unhandled exception error.
00291 */
00292 int irq_set_global_handler(irq_handler hnd);
00293
00294 /** \brief  Get the global exception handler.
00295
00296     \return                 The global exception handler set with
00297                             irq_set_global_handler(), or NULL if none is set.
00298 */
00299 irq_handler irq_get_global_handler(void);
00300
00301 /** \brief  Switch out contexts (for interrupt return).
00302
00303     This function will set the processor state that will be restored when the
00304     exception returns.
00305
00306     \param  regbank         The values of all registers to be restored.
00307 */
00308 void irq_set_context(irq_context_t *regbank);
00309
00310 /** \brief  Get the current IRQ context.
00311
00312     This will fetch the processor context prior to the exception handling during
00313     an IRQ service routine.
00314
00315     \return                 The current IRQ context.
00316 */
00317 irq_context_t *irq_get_context(void);
00318
00319 /** \brief  Fill a newly allocated context block for usage with supervisor
```

```
00320             or user mode.
00321
00322     The given parameters will be passed to the called routine (up to the
00323     architecture maximum). For the Dreamcast, this maximum is 4.
00324
00325     \param  context         The IRQ context to fill in.
00326     \param  stack_pointer   The value to set in the stack pointer.
00327     \param  routine         The address of the program counter for the context.
00328     \param  args            Any arguments to set in the registers. This cannot
00329                             be NULL, and must have enough values to fill in up
00330                             to the architecture maximum.
00331     \param  usermode        1 to run the routine in user mode, 0 for supervisor.
00332 */
00333 void irq_create_context(irq_context_t *context, uint32 stack_pointer,
00334                         uint32 routine, uint32 *args, int usermode);
00335
00336 /* Enable/Disable interrupts */
00337 /** \brief  Disable interrupts.
00338
00339     This function will disable interrupts, but will leave exceptions enabled.
00340
00341     \return                 The state of IRQs before calling the function. This
00342                             can be used to restore this state later on with
00343                             irq_restore().
00344 */
00345 int irq_disable(void);
00346
00347 /** \brief  Enable all interrupts.
00348
00349     This function will enable ALL interrupts, including external ones.
00350 */
00351 void irq_enable(void);
00352
00353 /** \brief  Restore IRQ state.
00354
00355     This function will restore the interrupt state to the value specified. This
00356     should correspond to a value returned by irq_disable().
00357
00358     \param  v               The IRQ state to restore. This should be a value
00359                             returned by irq_disable().
00360 */
00361 void irq_restore(int v);
00362
00363 /** \brief  Initialize interrupts.
00364     \retval 0               On success (no error conditions defined).
00365 */
00366 int irq_init(void);
00367
00368 /** \brief  Shutdown interrupts, restoring the state to how it was before
00369             irq_init() was called.
00370 */
00371 void irq_shutdown(void);
00372
00373 __END_DECLS
00374
00375 #endif  /* __ARCH_IRQ_H */
```

9.146 kernel/arch/dreamcast/include/arch/memory.h File Reference

Constants for areas of the system memory map.

```
#include <sys/cdefs.h>
```

Macros

- #define MEM_AREA_CACHE_MASK 0x1fffffff

  Mask a cache-agnostic address.
- #define MEM_AREA_U0_BASE 0x00000000

U0 memory region (cachable).

- #define MEM_AREA_P0_BASE 0x00000000

  P0 memory region (cachable).
- #define MEM_AREA_P1_BASE 0x80000000

  P1 memory region (cachable).
- #define MEM_AREA_P2_BASE 0xa0000000

  P2 memory region (non-cachable).
- #define MEM_AREA_P3_BASE 0xc0000000

  P3 memory region (cachable).
- #define MEM_AREA_P4_BASE 0xe0000000
- #define MEM_AREA_SQ_BASE 0xe0000000

  Store Queue (SQ) memory base.
- #define MEM_AREA_ICACHE_ADDRESS_ARRAY_BASE 0xf0000000

  Instruction cache address array base.
- #define MEM_AREA_ICACHE_DATA_ARRAY_BASE 0xf1000000

  Instruction cache data array base.
- #define MEM_AREA_ITLB_ADDRESS_ARRAY_BASE 0xf2000000

  Instruction TLB address array base.
- #define MEM_AREA_ITLB_DATA_ARRAY1_BASE 0xf3000000

  Instruction TLB data array 1 base.
- #define MEM_AREA_ITLB_DATA_ARRAY2_BASE 0xf3800000

  Instruction TLB data array 2 base.
- #define MEM_AREA_OCACHE_ADDRESS_ARRAY_BASE 0xf4000000

  Operand cache address array base.
- #define MEM_AREA_OCACHE_DATA_ARRAY_BASE 0xf5000000

  Instruction cache data array base.
- #define MEM_AREA_UTLB_ADDRESS_ARRAY_BASE 0xf6000000

  Unified TLB address array base.
- #define MEM_AREA_UTLB_DATA_ARRAY1_BASE 0xf7000000

  Unified TLB data array 1 base.
- #define MEM_AREA_UTLB_DATA_ARRAY2_BASE 0xf7800000

  Unified TLB data array 2 base.
- #define MEM_AREA_CTRL_REG_BASE 0xff000000

  Control Register base.

9.146.1 Detailed Description

Constants for areas of the system memory map.

Various addresses and masks that are set by the SH7750. None of the values here are Dreamcast-specific.

These values are drawn from the Hitatchi SH7750 Series Hardware Manual rev 6.0.

Author

Donald Haase

9.146.2 Macro Definition Documentation

MEM_AREA_P4_BASE

```
#define MEM_AREA_P4_BASE 0xe0000000
```

9.147 memory.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kernel/arch/dreamcast/include/arch/memory.h
00004    Copyright (C) 2023 Donald Haase
00005
00006 */
00007
00008 /** \file   arch/memory.h
00009     \brief  Constants for areas of the system memory map.
00010
00011     Various addresses and masks that are set by the SH7750. None of the values
00012     here are Dreamcast-specific.
00013
00014     These values are drawn from the Hitatchi SH7750 Series Hardware Manual rev 6.0.
00015
00016     \author Donald Haase
00017 */
00018
00019 #ifndef __ARCH_MEMORY_H
00020 #define __ARCH_MEMORY_H
00021
00022 #include <sys/cdefs.h>
00023 __BEGIN_DECLS
00024
00025 /** \defgroup memory Memory
00026     \brief    Basics of the SH4 Memory Map
00027
00028     The SH7750 Series physical address space is mapped onto a 29-bit external
00029     memory space, with the upper 3 bits of the address indicating which memory
00030     region will be used. The P0/U0 memory region spans a 2GB space with the
00031     bottom 512MB mirrored to the P1, P2, and P3 regions.
00032
00033 */
00034
00035 /** \brief Mask a cache-agnostic address.
00036     \ingroup memory
00037
00038     This masks out the upper 3 bits of an address. This is used when it is
00039     necssary to access memory with a specified caching mode. This is needed for
00040     DMA and SQ usage as well as various MMU functions.
00041
00042 */
00043 #define MEM_AREA_CACHE_MASK 0x1fffffff
00044
00045 /** \brief U0 memory region (cachable).
00046     \ingroup memory
00047
00048     This is the base user mode memory address. It is cacheable as determined
00049     by the WT bit of the cache control register. By default KOS sets this to
00050     copy-back mode.
00051
00052     KOS runs in privileged mode, so this is here merely for completeness.
00053
00054 */
00055 #define MEM_AREA_U0_BASE    0x00000000
00056
00057 /** \brief P0 memory region (cachable).
00058     \ingroup memory
00059
00060     This is the base privileged mode memory address. It is cacheable as determined
00061     by the WT bit of the cache control register. By default KOS sets this to
00062     copy-back mode.
00063
```

```
00064 */
00065 #define MEM_AREA_P0_BASE    0x00000000
00066
00067 /** \brief P1 memory region (cachable).
00068     \ingroup memory
00069
00070     This is a modularly cachable memory region. It is cacheable as determined by
00071     the CB bit of the cache control register. That allows it to function in a
00072     different caching mode (copy-back v write-through) than the U0, P0, and P3
00073     regions, whose cache mode are governed by the WT bit. By default KOS sets this
00074     to the same copy-back mode as the other cachable regions.
00075
00076 */
00077 #define MEM_AREA_P1_BASE    0x80000000
00078
00079 /** \brief P2 memory region (non-cachable).
00080     \ingroup memory
00081
00082     This is the non-cachable memory region. It is most frequently for DMA
00083     transactions to ensure reads are not cached.
00084
00085 */
00086 #define MEM_AREA_P2_BASE    0xa0000000
00087
00088 /** \brief P3 memory region (cachable).
00089     \ingroup memory
00090
00091     This functions as the lower 512MB of P0.
00092
00093 */
00094 #define MEM_AREA_P3_BASE    0xc0000000
00095
00096 /** \brief P4 SH-internal memory region (non-cachable).
00097     \defgroup p4mem P4 memory region
00098     \ingroup memory
00099
00100     This offset maps to on-chip I/O channels.
00101
00102 */
00103 #define MEM_AREA_P4_BASE    0xe0000000
00104
00105 /** \brief Store Queue (SQ) memory base.
00106     \ingroup p4mem
00107
00108     This offset maps to the SQ memory region. RW to addresses from
00109     0xe0000000-0xe3ffffff follow SQ rules.
00110
00111     \see dc\sq.h
00112
00113 */
00114 #define MEM_AREA_SQ_BASE    0xe0000000
00115
00116 /** \brief Instruction cache address array base.
00117     \ingroup p4mem
00118
00119     This offset is used for direct access to the instruction cache address array.
00120
00121 */
00122 #define MEM_AREA_ICACHE_ADDRESS_ARRAY_BASE    0xf0000000
00123
00124 /** \brief Instruction cache data array base.
00125     \ingroup p4mem
00126
00127     This offset is used for direct access to the instruction cache data array.
00128
00129 */
00130 #define MEM_AREA_ICACHE_DATA_ARRAY_BASE       0xf1000000
00131
00132 /** \brief Instruction TLB address array base.
00133     \ingroup p4mem
00134
00135     This offset is used for direct access to the instruction TLB address array.
00136
00137 */
00138 #define MEM_AREA_ITLB_ADDRESS_ARRAY_BASE      0xf2000000
00139
00140 /** \brief Instruction TLB data array 1 base.
00141     \ingroup p4mem
00142
00143     This offset is used for direct access to the instruction TLB data array 1.
00144
```

```
00145 */
00146 #define MEM_AREA_ITLB_DATA_ARRAY1_BASE        0xf3000000
00147
00148 /** \brief Instruction TLB data array 2 base.
00149     \ingroup p4mem
00150
00151     This offset is used for direct access to the instruction TLB data array 2.
00152
00153 */
00154 #define MEM_AREA_ITLB_DATA_ARRAY2_BASE        0xf3800000
00155
00156 /** \brief Operand cache address array base.
00157     \ingroup p4mem
00158
00159     This offset is used for direct access to the operand cache address array.
00160
00161 */
00162 #define MEM_AREA_OCACHE_ADDRESS_ARRAY_BASE    0xf4000000
00163
00164 /** \brief Instruction cache data array base.
00165     \ingroup p4mem
00166
00167     This offset is used for direct access to the operand cache data array.
00168
00169 */
00170 #define MEM_AREA_OCACHE_DATA_ARRAY_BASE       0xf5000000
00171
00172 /** \brief Unified TLB address array base.
00173     \ingroup p4mem
00174
00175     This offset is used for direct access to the unified TLB address array.
00176
00177 */
00178 #define MEM_AREA_UTLB_ADDRESS_ARRAY_BASE      0xf6000000
00179
00180 /** \brief Unified TLB data array 1 base.
00181     \ingroup p4mem
00182
00183     This offset is used for direct access to the unified TLB data array 1.
00184
00185 */
00186 #define MEM_AREA_UTLB_DATA_ARRAY1_BASE        0xf7000000
00187
00188 /** \brief Unified TLB data array 2 base.
00189     \ingroup p4mem
00190
00191     This offset is used for direct access to the unified TLB data array 2.
00192
00193 */
00194 #define MEM_AREA_UTLB_DATA_ARRAY2_BASE        0xf7800000
00195
00196 /** \brief Control Register base.
00197     \ingroup p4mem
00198
00199     This is the base address of all control registers
00200
00201 */
00202 #define MEM_AREA_CTRL_REG_BASE                0xff000000
00203
00204 __END_DECLS
00205
00206 #endif /* __ARCH_MEMORY_H */
```

9.148 kernel/arch/dreamcast/include/arch/mmu.h File Reference

Memory Management Unit and Translation Lookaside Buffer handling.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <sys/uio.h>
```

Data Structures

- struct mmupage_t

  MMU TLB entry for a single page.
- struct mmusubcontext_t

  MMU sub-context type.
- struct mmucontext_t

  MMU context type.

Macros

- #define MMU_TOP_SHIFT 21

  Top-level shift.
- #define MMU_TOP_BITS 10

  Top-level bits.
- #define MMU_TOP_MASK ((1 << MMU_TOP_BITS) - 1)

  Top-level mask.
- #define MMU_BOT_SHIFT 12

  Bottom shift.
- #define MMU_BOT_BITS 9

  Bottom bits.
- #define MMU_BOT_MASK ((1 << MMU_BOT_BITS) - 1)

  Bottom mask.
- #define MMU_IND_SHIFT 0

  Index shift.
- #define MMU_IND_BITS 12

  Index bits.
- #define MMU_IND_MASK ((1 << MMU_IND_BITS) - 1)

  Index mask.
- #define MMU_KERNEL_RDONLY 0

  No user access, kernel read-only.
- #define MMU_KERNEL_RDWR 1

  No user access, kernel full.
- #define MMU_ALL_RDONLY 2

  Read-only user and kernel.
- #define MMU_ALL_RDWR 3

  Full access, user and kernel.
- #define MMU_NO_CACHE 1

  Cache disabled.
- #define MMU_CACHE_BACK 2

  Write-back cacheing.
- #define MMU_CACHE_WT 3

  Write-through cacheing.
- #define MMU_CACHEABLE MMU_CACHE_BACK

  Default cacheing.
- #define MMU_SUB_PAGES 512

  The number of pages in a sub-context.
- #define MMU_PAGES 1024

  The number of sub-contexts in the main level context.

Typedefs

- typedef mmupage_t *(* mmu_mapfunc_t) (mmucontext_t *context, int virtpage)

  MMU mapping handler.

Functions

- void mmu_use_table (mmucontext_t *context)

  Set the "current" page tables for TLB handling.
- mmucontext_t * mmu_context_create (int asid)

  Allocate a new MMU context.
- void mmu_context_destroy (mmucontext_t *context)

  Destroy an MMU context when a process is being destroyed.
- int mmu_virt_to_phys (mmucontext_t *context, int virtpage)

  Using the given page tables, translate the virtual page ID to a physical page ID.
- int mmu_phys_to_virt (mmucontext_t *context, int physpage)

  Using the given page tables, translate the physical page ID to a virtual page ID.
- void mmu_switch_context (mmucontext_t *context)

  Switch to the given context.
- void mmu_page_map (mmucontext_t *context, int virtpage, int physpage, int count, int prot, int cache, int share, int dirty)

  Set the given virtual page to map to the given physical page.
- int mmu_copyin (mmucontext_t *context, uint32 srcaddr, uint32 srccnt, void *buffer)

  Copy a chunk of data from a process' address space into a kernel buffer, taking into account page mappings.
- int mmu_copyv (mmucontext_t *context1, struct iovec *iov1, int iovcnt1, mmucontext_t *context2, struct iovec *iov2, int iovcnt2)

  Copy a chunk of data from one process' address space to another process' address space, taking into account page mappings.
- mmu_mapfunc_t mmu_map_get_callback (void)

  Get the current mapping function.
- mmu_mapfunc_t mmu_map_set_callback (mmu_mapfunc_t newfunc)

  Set a new MMU mapping handler.
- int mmu_init (void)

  Initialize MMU support.
- void mmu_shutdown (void)

  Shutdown MMU support.
- void mmu_reset_itlb (void)

  Reset ITLB.

Variables

- mmucontext_t * mmu_cxt_current

  "Current" page tables (for TLB exception handling).

9.148.1 Detailed Description

Memory Management Unit and Translation Lookaside Buffer handling.

This file defines the interface to the Memory Management Unit (MMU) in the SH4. The MMU, while not used normally by KOS, is available for virtual memory use, if you so desire. While using this functionality is probably overkill for most homebrew, there are a few very interesting things that this functionality could be used for (like mapping large files into memory that wouldn't otherwise fit).

The whole system is set up as a normal paged memory virtual->physical address translation. KOS implements the page table as a sparse, two-level page table. By default, pages are 4KB in size. Each top-level page table entry has 512 2nd level entries (there are 1024 entries in the top-level entry). This works out to about 2KB of space needed for one top-level entry.

The SH4 itself has 4 TLB entries for instruction fetches, and 64 "unified" TLB entries (for combined instructions + data). Essentially, the UTLB acts both as the TLB for data accesses (from mov instructions) and as a cache for entries for the ITLB. If there is no entry in the ITLB for an instruction access, the UTLB will automatically be searched. If no entry is found still, an ITLB miss exception will be generated. Data accesses are handled similarly to this (although additional complications are involved due to write accesses, and of course the ITLB doesn't play into data accesses).

For more information about how the MMU works, refer to the Hitachi/Renesas SH4 programming manual. It has much more detailed information than what is in here, for obvious reasons.

This functionality was ported over to mainline KOS from the KOS-MMU project of Megan Potter. Unfortunately, KOS-MMU never reached a real phase of maturity and usefulness, but this piece can be quite useful on its own.

Author

Megan Potter

9.148.2 Macro Definition Documentation

MMU_PAGES

```
#define MMU_PAGES 1024
```

The number of sub-contexts in the main level context.

MMU_SUB_PAGES

```
#define MMU_SUB_PAGES 512
```

The number of pages in a sub-context.

9.148.3 Typedef Documentation

mmu_mapfunc_t

```
typedef mmupage_t *(* mmu_mapfunc_t) (mmucontext_t *context, int virtpage)
```

MMU mapping handler.

This type is used for functions that will take over the mapping for the kernel. In general, there shouldn't be much use for taking this over yourself, unless you want to change the size of the page table entries or something of the like.

Parameters

context	The context in use.
virtpage	The virtual page to map.

Returns

The page table entry, or NULL if none exists.

9.148.4 Function Documentation

mmu_context_create()

```
mmucontext_t * mmu_context_create (
            int asid )
```

Allocate a new MMU context.

Each process should have exactly one of these, and these should not exist without a process. Since KOS doesn't actually have a process model of its own, that means you will only ever have one of these, if any.

Parameters

asid	The address space ID of this process.

Returns

The newly created context or NULL on fail.

mmu_context_destroy()

```
void mmu_context_destroy (
            mmucontext_t * context )
```

Destroy an MMU context when a process is being destroyed.

This function cleans up a MMU context, deallocating any memory its using.

Parameters

context	The context to clean up after.

mmu_copyin()

```
int mmu_copyin (
             mmucontext_t * context,
             uint32 srcaddr,
             uint32 srccnt,
             void * buffer )
```

Copy a chunk of data from a process' address space into a kernel buffer, taking into account page mappings.

Parameters

context	The context to use.
srcaddr	Source, in the mapped memory space.
srccnt	The number of bytes to copy.
buffer	The kernel buffer to copy into (should be in P1).

Returns

The number of bytes copied (failure causes arch_panic).

mmu_copyv()

```
int mmu_copyv (
             mmucontext_t * context1,
             struct iovec * iov1,
             int iovcnt1,
             mmucontext_t * context2,
             struct iovec * iov2,
             int iovcnt2 )
```

Copy a chunk of data from one process' address space to another process' address space, taking into account page mappings.

Parameters

context1	The source's context.
iov1	The scatter/gather array to copy from.
iovcnt1	The number of entries in iov1.
context2	The destination's context.
iov2	The scatter/gather array to copy to.
iovcnt2	The number of entries in iov2.

Returns

The number of bytes copied (failure causes arch_panic).

mmu_init()

```
int mmu_init (
              void  )
```

Initialize MMU support.

Unlike most things in KOS, the MMU is not initialized by a normal startup. This is because for most homebrew, its not needed.

Return values

0	On success (no error conditions defined).

mmu_map_get_callback()

```
mmu_mapfunc_t mmu_map_get_callback (
              void  )
```

Get the current mapping function.

Returns

The current function that maps pages.

mmu_map_set_callback()

```
mmu_mapfunc_t mmu_map_set_callback (
              mmu_mapfunc_t newfunc )
```

Set a new MMU mapping handler.

This function will allow you to set a new function to handle mapping for memory pages. There's not much of a reason to do this unless you really do not like the way KOS handles the page mapping internally.

Parameters

newfunc	The new function to handle mapping.

Returns

The old function that did mapping.

mmu_page_map()

```
void mmu_page_map (
```

```
            mmucontext_t * context,
            int virtpage,
            int physpage,
            int count,
            int prot,
            int cache,
            int share,
            int dirty )
```

Set the given virtual page to map to the given physical page.

This implies turning on the "valid" bit. Also sets the other named attributes as specified.

Parameters

context	The context to modify.
virtpage	The first virtual page to map.
physpage	The first physical page to map.
count	The number of sequential pages to map.
prot	Memory protection for page (see MMU protection settings).
cache	Cache scheme for page (see MMU cacheability settings).
share	Set to 1 to share between processes (meaningless), otherwise set to 0.
dirty	Set to 1 to mark the page as dirty, otherwise set to 0.

mmu_phys_to_virt()

```
int mmu_phys_to_virt (
            mmucontext_t * context,
            int physpage )
```

Using the given page tables, translate the physical page ID to a virtual page ID.

Parameters

context	The context to look in.
physpage	The physical page number to look for.

Returns

The virtual page number, or -1 on failure.

See also

mmu_virt_to_phys()

mmu_reset_itlb()

```
void mmu_reset_itlb (
            void  )
```

Reset ITLB.

mmu_shutdown()

```
void mmu_shutdown (
            void  )
```

Shutdown MMU support.

Turn off the MMU after it was initialized. You should try to make sure this gets done if you initialize the MMU in your program, so as to play nice with loaders and the like (that will not expect that its on, in general).

mmu_switch_context()

```
void mmu_switch_context (
            mmucontext_t * context )
```

Switch to the given context.

This function switches to the given context's address space ID. The context should have already been made current with mmu_use_table(). You are responsible for invalidating any caches as neccessary, as well as invalidating any stale TLB entries.

Parameters

context	The context to make current.

mmu_use_table()

```
void mmu_use_table (
            mmucontext_t * context )
```

Set the "current" page tables for TLB handling.

This function is useful if you're trying to implement a process model or something of the like on top of KOS. Essentially, this allows you to completely boot the MMU context in use out and replace it with another. You will need to call the mmu_switch_context() function afterwards to set the address space id.

Parameters

context	The context to make current.

mmu_virt_to_phys()

```
int mmu_virt_to_phys (
            mmucontext_t * context,
            int virtpage )
```

Using the given page tables, translate the virtual page ID to a physical page ID.

Parameters

context	The context to look in.
virtpage	The virtual page number to look for.

Returns

The physical page number, or -1 on failure.

See also

mmu_phys_to_virt()

9.148.5 Variable Documentation

mmu_cxt_current

```
mmucontext_t* mmu_cxt_current  [extern]
```

"Current" page tables (for TLB exception handling).

You should not modify this directly, but rather use the functions provided to do so.

9.149 mmu.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/arch/mmu.h
00004    Copyright (C) 2001 Megan Potter
00005
00006 */
00007
00008 /** \file   arch/mmu.h
00009     \brief  Memory Management Unit and Translation Lookaside Buffer handling.
00010
00011     This file defines the interface to the Memory Management Unit (MMU) in the
00012     SH4. The MMU, while not used normally by KOS, is available for virtual
00013     memory use, if you so desire. While using this functionality is probably
00014     overkill for most homebrew, there are a few very interesting things that
00015     this functionality could be used for (like mapping large files into memory
00016     that wouldn't otherwise fit).
00017
00018     The whole system is set up as a normal paged memory virtual->physical
00019     address translation. KOS implements the page table as a sparse, two-level
00020     page table. By default, pages are 4KB in size. Each top-level page table
```

```
00021     entry has 512 2nd level entries (there are 1024 entries in the top-level
00022     entry). This works out to about 2KB of space needed for one top-level entry.
00023
00024     The SH4 itself has 4 TLB entries for instruction fetches, and 64 "unified"
00025     TLB entries (for combined instructions + data). Essentially, the UTLB acts
00026     both as the TLB for data accesses (from mov instructions) and as a cache for
00027     entries for the ITLB. If there is no entry in the ITLB for an instruction
00028     access, the UTLB will automatically be searched. If no entry is found still,
00029     an ITLB miss exception will be generated. Data accesses are handled
00030     similarly to this (although additional complications are involved due to
00031     write accesses, and of course the ITLB doesn't play into data accesses).
00032
00033     For more information about how the MMU works, refer to the Hitachi/Renesas
00034     SH4 programming manual. It has much more detailed information than what is
00035     in here, for obvious reasons.
00036
00037     This functionality was ported over to mainline KOS from the KOS-MMU project
00038     of Megan Potter. Unfortunately, KOS-MMU never reached a real phase of maturity
00039     and usefulness, but this piece can be quite useful on its own.
00040
00041     \author Megan Potter
00042 */
00043
00044 #ifndef __ARCH_MMU_H
00045 #define __ARCH_MMU_H
00046
00047 #include <sys/cdefs.h>
00048 __BEGIN_DECLS
00049
00050 #include <arch/types.h>
00051 #include <sys/uio.h>
00052
00053 /* Since the software has to handle TLB misses on the SH-4, we have freedom
00054    to use any page table format we want (and thus save space), but we must
00055    make it quick to access. The SH-4 can address a maximum of 512M of address
00056    space per "area", but we only care about one area, so this is the total
00057    maximum addressable space. With 4K pages, that works out to 2^17 pages
00058    that must be mappable, or 17 bits. We use 18 bits just to be sure (there
00059    are a few left over).
00060
00061    Page tables (per-process) are a sparse two-level array. The virtual address
00062    space is actually 2^30 bytes, or 2^(30-12)=2^18 pages, so there must be
00063    a possibility of having that many page entries per process space. A full
00064    page table for a process would be 1M, so this is obviously too big!! Thus
00065    the sparse array.
00066
00067    The bottom layer of the page tables consists of a sub-context array for
00068    512 pages, which translates into 2K of storage space. The process then
00069    has the possibility of using one or more of the 512 top-level slots. For
00070    a very small process (using one page for code/data and one for stack), it
00071    should be possible to achieve a page table footprint of one page. The tables
00072    can grow from there as neccessary.
00073
00074    Virtual addresses are broken up as follows:
00075
00076    Bits 31 - 22     10 bits top-level page directory
00077    Bits 21 - 13     9 bits bottom-level page entry
00078    Bits 11 - 0      Byte index into page
00079
00080    */
00081
00082 /** \defgroup mmu_bit_macros        MMU address bit definitions
00083
00084     The MMU code uses these to determine the page of a request.
00085
00086     @{
00087 */
00088 #define MMU_TOP_SHIFT 21                        /**< \brief Top-level shift */
00089 #define MMU_TOP_BITS 10                         /**< \brief Top-level bits */
00090 #define MMU_TOP_MASK ((1 << MMU_TOP_BITS) - 1)  /**< \brief Top-level mask */
00091 #define MMU_BOT_SHIFT 12                        /**< \brief Bottom shift */
00092 #define MMU_BOT_BITS 9                          /**< \brief Bottom bits */
00093 #define MMU_BOT_MASK ((1 << MMU_BOT_BITS) - 1)  /**< \brief Bottom mask */
00094 #define MMU_IND_SHIFT 0                         /**< \brief Index shift */
00095 #define MMU_IND_BITS 12                         /**< \brief Index bits */
00096 #define MMU_IND_MASK ((1 << MMU_IND_BITS) - 1)  /**< \brief Index mask */
00097 /** @} */
00098
00099 /** \defgroup mmu_prot_values       MMU protection settings
00100
00101     Each page mapped via the MMU can be protected in a couple of different ways,
```

```
00102     as specified here.
00103
00104     @{
00105 */
00106 #define MMU_KERNEL_RDONLY   0   /**< \brief No user access, kernel read-only */
00107 #define MMU_KERNEL_RDWR     1   /**< \brief No user access, kernel full */
00108 #define MMU_ALL_RDONLY      2   /**< \brief Read-only user and kernel */
00109 #define MMU_ALL_RDWR        3   /**< \brief Full access, user and kernel */
00110 /** @} */
00111
00112 /** \defgroup mmu_cache_values      MMU cacheability settings
00113
00114     Each page mapped via the MMU can have its cacheability set individually.
00115
00116     @{
00117 */
00118 #define MMU_NO_CACHE    1               /**< \brief Cache disabled */
00119 #define MMU_CACHE_BACK  2               /**< \brief Write-back cacheing */
00120 #define MMU_CACHE_WT    3               /**< \brief Write-through cacheing */
00121 #define MMU_CACHEABLE   MMU_CACHE_BACK  /**< \brief Default cacheing */
00122 /** @} */
00123
00124 /** \brief  MMU TLB entry for a single page.
00125
00126     The TLB entries on the SH4 are a single 32-bit dword in length. We store
00127     some other data here too for ease of use.
00128
00129     \headerfile arch/mmu.h
00130 */
00131 typedef struct mmupage {
00132     /* Explicit pieces, used for reference */
00133     /*uint32    virtual; */ /* implicit */
00134     uint32  physical: 18;   /**< \brief Physical page ID -- 18 bits */
00135     uint32  prkey: 2;       /**< \brief Protection key data -- 2 bits */
00136     uint32  valid: 1;       /**< \brief Valid mapping -- 1 bit */
00137     uint32  shared: 1;      /**< \brief Shared between procs -- 1 bit */
00138     uint32  cache: 1;       /**< \brief Cacheable -- 1 bit */
00139     uint32  dirty: 1;       /**< \brief Dirty -- 1 bit */
00140     uint32  wthru: 1;       /**< \brief Write-thru enable -- 1 bit */
00141     uint32  blank: 7;       /**< \brief Reserved -- 7 bits */
00142
00143     /* Pre-compiled pieces. These waste a bit of ram, but they also
00144        speed loading immensely at runtime. */
00145     uint32  pteh;           /**< \brief Pre-built PTEH value */
00146     uint32  ptel;           /**< \brief Pre-built PTEL value */
00147 } mmupage_t;
00148
00149 /** \brief  The number of pages in a sub-context. */
00150 #define MMU_SUB_PAGES   512
00151
00152 /** \brief  MMU sub-context type.
00153
00154     We have two-level page tables on SH4, and each sub-context contains 512
00155     entries.
00156
00157     \headerfile arch/mmu.h
00158 */
00159 typedef struct mmusubcontext {
00160     mmupage_t   page[MMU_SUB_PAGES];    /**< \brief 512 page entries */
00161 } mmusubcontext_t;
00162
00163 /** \brief  The number of sub-contexts in the main level context. */
00164 #define MMU_PAGES   1024
00165
00166 /** \brief  MMU context type.
00167
00168     This type is the top-level context that makes up the page table. There is
00169     one of these, with 1024 sub-contexts.
00170
00171     \headerfile arch/mmu.h
00172 */
00173 typedef struct mmucontext {
00174     mmusubcontext_t *sub[MMU_PAGES];    /**< \brief 1024 sub-contexts */
00175     int             asid;               /**< \brief Address Space ID */
00176 } mmucontext_t;
00177
00178 /** \brief  "Current" page tables (for TLB exception handling).
00179
00180     You should not modify this directly, but rather use the functions provided
00181     to do so.
00182 */
```

```
00183 extern mmucontext_t *mmu_cxt_current;
00184
00185 /** \brief  Set the "current" page tables for TLB handling.
00186
00187     This function is useful if you're trying to implement a process model or
00188     something of the like on top of KOS. Essentially, this allows you to
00189     completely boot the MMU context in use out and replace it with another. You
00190     will need to call the mmu_switch_context() function afterwards to set the
00191     address space id.
00192
00193     \param  context         The context to make current.
00194 */
00195 void mmu_use_table(mmucontext_t *context);
00196
00197 /** \brief  Allocate a new MMU context.
00198
00199     Each process should have exactly one of these, and these should not exist
00200     without a process. Since KOS doesn't actually have a process model of its
00201     own, that means you will only ever have one of these, if any.
00202
00203     \param  asid            The address space ID of this process.
00204     \return                 The newly created context or NULL on fail.
00205 */
00206 mmucontext_t *mmu_context_create(int asid);
00207
00208 /** \brief  Destroy an MMU context when a process is being destroyed.
00209
00210     This function cleans up a MMU context, deallocating any memory its using.
00211
00212     \param  context         The context to clean up after.
00213 */
00214 void mmu_context_destroy(mmucontext_t *context);
00215
00216 /** \brief  Using the given page tables, translate the virtual page ID to a
00217             physical page ID.
00218
00219     \param  context         The context to look in.
00220     \param  virtpage        The virtual page number to look for.
00221     \return                 The physical page number, or -1 on failure.
00222     \see    mmu_phys_to_virt()
00223 */
00224 int mmu_virt_to_phys(mmucontext_t *context, int virtpage);
00225
00226 /** \brief  Using the given page tables, translate the physical page ID to a
00227             virtual page ID.
00228
00229     \param  context         The context to look in.
00230     \param  physpage        The physical page number to look for.
00231     \return                 The virtual page number, or -1 on failure.
00232     \see    mmu_virt_to_phys()
00233 */
00234 int mmu_phys_to_virt(mmucontext_t *context, int physpage);
00235
00236 /** \brief  Switch to the given context.
00237
00238     This function switches to the given context's address space ID. The context
00239     should have already been made current with mmu_use_table().
00240     You are responsible for invalidating any caches as neccessary, as well as
00241     invalidating any stale TLB entries.
00242
00243     \param  context         The context to make current.
00244 */
00245 void mmu_switch_context(mmucontext_t *context);
00246
00247 /** \brief  Set the given virtual page to map to the given physical page.
00248
00249     This implies turning on the "valid" bit. Also sets the other named
00250     attributes as specified.
00251
00252     \param  context         The context to modify.
00253     \param  virtpage        The first virtual page to map.
00254     \param  physpage        The first physical page to map.
00255     \param  count           The number of sequential pages to map.
00256     \param  prot            Memory protection for page (see
00257                             \ref mmu_prot_values).
00258     \param  cache           Cache scheme for page (see \ref mmu_cache_values).
00259     \param  share           Set to 1 to share between processes (meaningless),
00260                             otherwise set to 0.
00261     \param  dirty           Set to 1 to mark the page as dirty, otherwise set to
00262                             0.
00263 */
```

```
00264 void mmu_page_map(mmucontext_t *context, int virtpage, int physpage,
00265                   int count, int prot, int cache, int share, int dirty);
00266
00267 /** \brief  Copy a chunk of data from a process' address space into a kernel
00268             buffer, taking into account page mappings.
00269
00270     \param  context         The context to use.
00271     \param  srcaddr         Source, in the mapped memory space.
00272     \param  srccnt          The number of bytes to copy.
00273     \param  buffer          The kernel buffer to copy into (should be in P1).
00274     \return                 The number of bytes copied (failure causes arch_panic).
00275 */
00276 int mmu_copyin(mmucontext_t *context, uint32 srcaddr, uint32 srccnt,
00277                void *buffer);
00278
00279 /** \brief  Copy a chunk of data from one process' address space to another
00280             process' address space, taking into account page mappings.
00281
00282     \param  context1        The source's context.
00283     \param  iov1            The scatter/gather array to copy from.
00284     \param  iovcnt1         The number of entries in iov1.
00285     \param  context2        The destination's context.
00286     \param  iov2            The scatter/gather array to copy to.
00287     \param  iovcnt2         The number of entries in iov2.
00288     \return                 The number of bytes copied (failure causes arch_panic).
00289 */
00290 int mmu_copyv(mmucontext_t *context1, struct iovec *iov1, int iovcnt1,
00291               mmucontext_t *context2, struct iovec *iov2, int iovcnt2);
00292
00293 /** \brief  MMU mapping handler.
00294
00295     This type is used for functions that will take over the mapping for the
00296     kernel. In general, there shouldn't be much use for taking this over
00297     yourself, unless you want to change the size of the page table entries or
00298     something of the like.
00299
00300     \param  context         The context in use.
00301     \param  virtpage        The virtual page to map.
00302     \return                 The page table entry, or NULL if none exists.
00303 */
00304 typedef mmupage_t * (*mmu_mapfunc_t)(mmucontext_t * context, int virtpage);
00305
00306 /** \brief  Get the current mapping function.
00307     \return                 The current function that maps pages.
00308 */
00309 mmu_mapfunc_t mmu_map_get_callback(void);
00310
00311 /** \brief  Set a new MMU mapping handler.
00312
00313     This function will allow you to set a new function to handle mapping for
00314     memory pages. There's not much of a reason to do this unless you really do
00315     not like the way KOS handles the page mapping internally.
00316
00317     \param  newfunc         The new function to handle mapping.
00318     \return                 The old function that did mapping.
00319 */
00320 mmu_mapfunc_t mmu_map_set_callback(mmu_mapfunc_t newfunc);
00321
00322 /** \brief  Initialize MMU support.
00323
00324     Unlike most things in KOS, the MMU is not initialized by a normal startup.
00325     This is because for most homebrew, its not needed.
00326
00327     \retval 0               On success (no error conditions defined).
00328 */
00329 int mmu_init(void);
00330
00331 /** \brief  Shutdown MMU support.
00332
00333     Turn off the MMU after it was initialized. You should try to make sure this
00334     gets done if you initialize the MMU in your program, so as to play nice with
00335     loaders and the like (that will not expect that its on, in general).
00336 */
00337 void mmu_shutdown(void);
00338
00339 /** \brief  Reset ITLB. */
00340 void mmu_reset_itlb(void);
00341
00342 __END_DECLS
00343
00344 #endif  /* __ARCH_MMU_H */
```

9.150 kernel/arch/dreamcast/include/arch/rtc.h File Reference

Low-level real-time clock functionality.

```
#include <sys/cdefs.h>
#include <time.h>
```

Macros

- #define RTC_TIMESTAMP_HIGH_ADDR 0xa0710000
  High 16-bit timestamp value.
- #define RTC_TIMESTAMP_LOW_ADDR 0xa0710004
  Low 16-bit timestamp value.
- #define RTC_CTRL_ADDR 0xa0710008
  Timestamp control register.
- #define RTC_CTRL_WRITE_EN (1 << 0)
  Timestamp write enable.

Functions

- time_t rtc_unix_secs (void)
  Get the current date/time.
- int rtc_set_unix_secs (time_t time)
  Get the current date/time.
- time_t rtc_boot_time (void)
  Get the time since the sytem was booted.

9.150.1 Detailed Description

Low-level real-time clock functionality.

This file contains functions for interacting with the real-time clock in the Dreamcast. Generally, you should prefer interacting with the higher level standard C functions, like time(), rather than these when simply needing to fetch the current system time.

Author

Megan Potter

Falco Girgis

9.150.2 Macro Definition Documentation

RTC_CTRL_WRITE_EN

```
#define RTC_CTRL_WRITE_EN (1 << 0)
```

Timestamp write enable.

RTC_CTRL_ADDR value to be written in order to unlock writing to the timestamp registers.

9.151 rtc.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/rtc.h
00004    Copyright (C) 2000-2001 Megan Potter
00005    Copyright (C) 2023 Falco Girgis
00006
00007 */
00008
00009 /** \file    arch/rtc.h
00010     \brief   Low-level real-time clock functionality.
00011     \ingroup rtc
00012
00013     This file contains functions for interacting with the real-time clock in the
00014     Dreamcast. Generally, you should prefer interacting with the higher level
00015     standard C functions, like time(), rather than these when simply needing
00016     to fetch the current system time.
00017
00018     \author Megan Potter
00019     \author Falco Girgis
00020 */
00021
00022 #ifndef __ARCH_RTC_H
00023 #define __ARCH_RTC_H
00024
00025 #include <sys/cdefs.h>
00026 __BEGIN_DECLS
00027
00028 #include <time.h>
00029
00030 /** \defgroup rtc Real-Time Clock
00031     \brief Real-Time Clock (RTC) Management
00032
00033     Provides an API for fetching and managing the date/time using
00034     the Dreamcast's real-time clock. All timestamps are in standard
00035     Unix format, with an epoch of January 1, 1970. Due to the fact
00036     that there is no time zone data on the RTC, all times are expected
00037     to be in the local time zone.
00038
00039     \note
00040     The RTC that is used by the DC is located on the AICA rather than SH4,
00041     presumably for power-efficiency reasons. Because of this, accessing
00042     it requires a trip over the G2 BUS, which is notoriously slow.
00043
00044     \note
00045     For reading the current date/time, you should favor the standard C,
00046     C++, or POSIX functions, as they are platform-indpendent and are
00047     calculating current time based on a cached boot time plus a delta
00048     that is maintained by the timer subsystem, rather than actually
00049     having to requery the RTC over the G2 BUS, so they are faster.
00050
00051     \warning
00052     Internally, the RTC's date/time is maintained using a 32-bit counter
00053     with an epoch of January 1, 1950 00:00. Because of this, the Dreamcast's
00054     Y2K and the last timestamp it can represent before rolling over is
00055     February 06 2086 06:28:15.
00056 */
00057
00058 /** \defgroup rtc_regs Registers
00059     \brief    RTC registers
00060     \ingroup  rtc
00061
00062     All registers are located on the G2 BUS and must be read and
00063     written to as full 32-byte values.
00064 @{*/
00065
00066 /** \brief High 16-bit timestamp value
00067
00068     32-bit register containing the upper 16-bits of
00069     the 32-bit timestamp in seconds. Only the lower 16-bits
00070     are valid.
00071
00072     \note Writing to this register will lock the timestamp registers.
00073  */
00074 #define RTC_TIMESTAMP_HIGH_ADDR   0xa0710000
00075
00076 /** \brief Low 16-bit timestamp value
```

```
00077
00078     32-bit register containing the lower 16-bits of
00079     the 32-bit timestamp in seconds. Only the lower 16-bits
00080     are valid.
00081  */
00082 #define RTC_TIMESTAMP_LOW_ADDR    0xa0710004
00083
00084 /** \brief Timestamp control register
00085
00086     All fields are reserved except for #RTC_CTRL_WRITE_EN,
00087     which is write-only.
00088  */
00089 #define RTC_CTRL_ADDR             0xa0710008
00090 /**
00091 @} */
00092
00093 /** \brief Timestamp write enable
00094
00095     #RTC_CTRL_ADDR value to be written in order to unlock
00096     writing to the timestamp registers.
00097 */
00098 #define RTC_CTRL_WRITE_EN         (1 « 0)
00099
00100 /** \brief   Get the current date/time.
00101     \ingroup rtc
00102
00103     This function retrieves the current RTC value as a standard UNIX timestamp
00104     (with an epoch of January 1, 1970 00:00). This is assumed to be in the
00105     timezone of the user (as the RTC does not support timezones).
00106
00107     \return                 The current UNIX-style timestamp (local time).
00108
00109     \sa rtc_set_unix_secs(), rtc_boot_time()
00110 */
00111 time_t rtc_unix_secs(void);
00112
00113 /** \brief   Get the current date/time.
00114     \ingroup rtc
00115
00116     This function sets the current RTC value as a standard UNIX timestamp
00117     (with an epoch of January 1, 1970 00:00). This is assumed to be in the
00118     timezone of the user (as the RTC does not support timezones).
00119
00120     \param time             Unix timestamp to set the current time to
00121
00122     \return                 0 for success or -1 for failure
00123
00124     \sa rtc_unix_secs()
00125 */
00126 int rtc_set_unix_secs(time_t time);
00127
00128 /** \brief   Get the time since the sytem was booted.
00129     \ingroup rtc
00130
00131     This function retrieves the cached RTC value from when KallistiOS was started. As
00132     with rtc_unix_secs(), this is a UNIX-style timestamp in local time.
00133
00134     \return                 The boot time as a UNIX-style timestamp.
00135
00136     \sa rtc_unix_secs()
00137 */
00138 time_t rtc_boot_time(void);
00139
00140 /* \cond */
00141 /* Internally called Init / Shutdown */
00142 int rtc_init(void);
00143 void rtc_shutdown(void);
00144 /* \endcond */
00145
00146 __END_DECLS
00147
00148 #endif  /* __ARCH_RTC_H */
00149
```

9.152 kernel/arch/dreamcast/include/arch/spinlock.h File Reference

Simple locking.

```
#include <sys/cdefs.h>
#include <kos/thread.h>
```

Macros

- #define SPINLOCK_INITIALIZER 0

  Spinlock initializer.
- #define spinlock_init(A) *(A) = SPINLOCK_INITIALIZER

  Initialize a spinlock.
- #define spinlock_lock(A)

  Spin on a lock.
- #define spinlock_trylock(A)

  Try to lock, without spinning.
- #define spinlock_unlock(A)

  Free a lock.
- #define spinlock_is_locked(A) (*(A) != 0)

  Determine if a lock is locked.

Typedefs

- typedef volatile int spinlock_t

  Spinlock data type.

9.152.1 Detailed Description

Simple locking.

This file contains definitions for very simple locks. Most of the time, you will probably not use such low-level locking, but will opt for something more fully featured like mutexes, semaphores, reader-writer semaphores, or recursive locks.

Author

Megan Potter

See also

kos/sem.h

kos/mutex.h

kos/rwsem.h

kos/recursive_lock.h

9.152.2 Macro Definition Documentation

spinlock_init

```
#define spinlock_init(
              A ) *(A) = SPINLOCK_INITIALIZER
```

Initialize a spinlock.

This function-like macro abstracts initializing a spinlock, in case the initializer is not applicable to what you are doing.

Parameters

A	A pointer to the spinlock to be initialized.

SPINLOCK_INITIALIZER

```
#define SPINLOCK_INITIALIZER 0
```

Spinlock initializer.

All created spinlocks should be initialized with this initializer so that they are in a sane state, ready to be used.

spinlock_is_locked

```
#define spinlock_is_locked(
              A ) ( *(A) != 0 )
```

Determine if a lock is locked.

This macro will return whether or not the lock specified is actually locked when it is called. This is NOT a thread-safe way of determining if a lock will be locked when you get around to locking it!

Parameters

A	A pointer to the spinlock to be checked.

spinlock_lock

```
#define spinlock_lock(
              A )
```

Value:

```
        do { \
        spinlock_t * __lock = A; \
        int __gotlock = 0; \
        while(1) { \
            __asm__ __volatile__("tas.b @%1\n\t" \
                                 "movt %0\n\t" \
                                 : "=r" (__gotlock) \
                                 : "r" (__lock) \
                                 : "t", "memory"); \
            if(!__gotlock) \
                thd_pass(); \
            else break; \
        } \
    } while(0)
```

Spin on a lock.

This macro will spin on the lock, and will not return until the lock has been obtained for the calling thread.

Parameters

A	A pointer to the spinlock to be locked.

spinlock_trylock

```
#define spinlock_trylock(
              A )
```

Value:

```
        ({ \
        int __gotlock = 0; \
        do { \
            spinlock_t *__lock = A; \
            __asm__ __volatile__("tas.b @%1\n\t" \
                                 "movt %0\n\t" \
                                 : "=r" (__gotlock) \
                                 : "r" (__lock) \
                                 : "t", "memory"); \
        } while(0); \
        __gotlock; \
    })
```

Try to lock, without spinning.

This macro will attempt to lock the lock, but will not spin. Instead, it will return whether the lock was obtained or not.

Parameters

A	A pointer to the spinlock to be locked.

Returns

0 if the lock is held by another thread. Non-zero if the lock was successfully obtained.

spinlock_unlock

```
#define spinlock_unlock(
              A )
```

Value:

```
        do { \
        *(A) = 0; \
    } while(0)
```

Free a lock.

This macro will unlock the lock that is currently held by the calling thread. Do not use this macro unless you actually hold the lock!

Parameters

A	A pointer to the spinlock to be unlocked.

9.152.3 Typedef Documentation

spinlock_t

```
typedef volatile int spinlock_t
```

Spinlock data type.

9.153 spinlock.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/spinlock.h
00004    Copyright (C) 2001 Megan Potter
00005
00006 */
00007
00008 /** \file   arch/spinlock.h
00009     \brief  Simple locking.
00010
00011     This file contains definitions for very simple locks. Most of the time, you
00012     will probably not use such low-level locking, but will opt for something
00013     more fully featured like mutexes, semaphores, reader-writer semaphores, or
00014     recursive locks.
00015
00016     \author Megan Potter
00017
00018     \see    kos/sem.h
00019     \see    kos/mutex.h
00020     \see    kos/rwsem.h
00021     \see    kos/recursive_lock.h
00022 */
00023
00024 #ifndef __ARCH_SPINLOCK_H
00025 #define __ARCH_SPINLOCK_H
00026
00027 /* Defines processor specific spinlocks */
00028
00029 #include <sys/cdefs.h>
00030 __BEGIN_DECLS
00031
00032 /* DC implementation uses threads most of the time */
00033 #include <kos/thread.h>
00034
00035 /** \brief  Spinlock data type. */
00036 typedef volatile int spinlock_t;
00037
00038 /** \brief  Spinlock initializer.
00039
00040     All created spinlocks should be initialized with this initializer so that
00041     they are in a sane state, ready to be used.
00042 */
00043 #define SPINLOCK_INITIALIZER 0
00044
00045 /** \brief  Initialize a spinlock.
00046
00047     This function-like macro abstracts initializing a spinlock, in case the
00048     initializer is not applicable to what you are doing.
00049
00050     \param  A               A pointer to the spinlock to be initialized.
00051 */
```

```
00052 #define spinlock_init(A) *(A) = SPINLOCK_INITIALIZER
00053
00054 /* Note here that even if threads aren't enabled, we'll still set the
00055    lock so that it can be used for anti-IRQ protection (e.g., malloc) */
00056
00057 /** \brief  Spin on a lock.
00058
00059     This macro will spin on the lock, and will not return until the lock has
00060     been obtained for the calling thread.
00061
00062     \param  A               A pointer to the spinlock to be locked.
00063 */
00064 #define spinlock_lock(A) do { \
00065         spinlock_t * __lock = A; \
00066         int __gotlock = 0; \
00067         while(1) { \
00068             __asm__ __volatile__("tas.b @%1\n\t" \
00069                                  "movt %0\n\t" \
00070                                  : "=r" (__gotlock) \
00071                                  : "r" (__lock) \
00072                                  : "t", "memory"); \
00073             if(!__gotlock) \
00074                 thd_pass(); \
00075             else break; \
00076         } \
00077     } while(0)
00078
00079 /** \brief  Try to lock, without spinning.
00080
00081     This macro will attempt to lock the lock, but will not spin. Instead, it
00082     will return whether the lock was obtained or not.
00083
00084     \param  A               A pointer to the spinlock to be locked.
00085     \return                 0 if the lock is held by another thread. Non-zero if
00086                             the lock was successfully obtained.
00087 */
00088 #define spinlock_trylock(A) ({ \
00089         int __gotlock = 0; \
00090         do { \
00091             spinlock_t *__lock = A; \
00092             __asm__ __volatile__("tas.b @%1\n\t" \
00093                                  "movt %0\n\t" \
00094                                  : "=r" (__gotlock) \
00095                                  : "r" (__lock) \
00096                                  : "t", "memory"); \
00097         } while(0); \
00098         __gotlock; \
00099     })
00100
00101 /** \brief  Free a lock.
00102
00103     This macro will unlock the lock that is currently held by the calling
00104     thread. Do not use this macro unless you actually hold the lock!
00105
00106     \param  A               A pointer to the spinlock to be unlocked.
00107 */
00108 #define spinlock_unlock(A) do { \
00109         *(A) = 0; \
00110     } while(0)
00111
00112 /** \brief  Determine if a lock is locked.
00113
00114     This macro will return whether or not the lock specified is actually locked
00115     when it is called. This is NOT a thread-safe way of determining if a lock
00116     will be locked when you get around to locking it!
00117
00118     \param  A               A pointer to the spinlock to be checked.
00119 */
00120 #define spinlock_is_locked(A) ( *(A) != 0 )
00121
00122 __END_DECLS
00123
00124 #endif  /* __ARCH_SPINLOCK_H */
```

9.154 kernel/arch/dreamcast/include/arch/stack.h File Reference

Stack traces.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Functions

- void arch_stk_trace (int n)

  Do a stack trace from the current function.
- void arch_stk_trace_at (uint32 fp, int n)

  Do a stack trace from the current function.

9.154.1 Detailed Description

Stack traces.

The functions in this file deal with doing stack traces. These functions will do a stack trace, as specified, printing it out to stdout (usually a dcload terminal). These functions only work if frame pointers have been enabled at compile time (-DFRAME_POINTERS and no -fomit-frame-pointer flag).

Author

Megan Potter

9.154.2 Function Documentation

arch_stk_trace()

```
void arch_stk_trace (
            int n )
```

Do a stack trace from the current function.

This function does a stack trace from the current function, printing the results to stdout. This is used, for instance, when an assertion fails in assert().

Parameters

n	The number of frames to leave off. Each frame is a jump to subroutine or branch to subroutine. assert() leaves off 2 frames, for reference.

arch_stk_trace_at()

```
void arch_stk_trace_at (
            uint32 fp,
            int n )
```

Do a stack trace from the current function.

This function does a stack trace from the the specified frame pointer, printing the results to stdout. This could be used for doing something like stack tracing a main thread from inside an IRQ handler.

Parameters

fp	The frame pointer to start from.
n	The number of frames to leave off.

9.155 stack.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/arch/stack.h
00004    (c)2002 Megan Potter
00005
00006 */
00007
00008 /** \file   arch/stack.h
00009     \brief  Stack traces.
00010
00011     The functions in this file deal with doing stack traces. These functions
00012     will do a stack trace, as specified, printing it out to stdout (usually a
00013     dcload terminal). These functions only work if frame pointers have been
00014     enabled at compile time (-DFRAME_POINTERS and no -fomit-frame-pointer flag).
00015
00016     \author Megan Potter
00017 */
00018
00019 #ifndef __ARCH_STACK_H
00020 #define __ARCH_STACK_H
00021
00022 #include <sys/cdefs.h>
00023 __BEGIN_DECLS
00024
00025 #include <arch/types.h>
00026
00027 /** \brief  Do a stack trace from the current function.
00028
00029     This function does a stack trace from the current function, printing the
00030     results to stdout. This is used, for instance, when an assertion fails in
00031     assert().
00032
00033     \param  n               The number of frames to leave off. Each frame is a
00034                             jump to subroutine or branch to subroutine. assert()
00035                             leaves off 2 frames, for reference.
00036 */
00037 void arch_stk_trace(int n);
00038
00039 /** \brief  Do a stack trace from the current function.
00040
00041     This function does a stack trace from the the specified frame pointer,
00042     printing the results to stdout. This could be used for doing something like
00043     stack tracing a main thread from inside an IRQ handler.
00044
00045     \param  fp              The frame pointer to start from.
00046     \param  n               The number of frames to leave off.
00047 */
00048 void arch_stk_trace_at(uint32 fp, int n);
00049
00050 __END_DECLS
00051
00052 #endif  /* __ARCH_EXEC_H */
00053
```

9.156 kernel/arch/dreamcast/include/arch/timer.h File Reference

Low-level timer functionality.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <arch/irq.h>
```

Macros

- #define TMU0 0

  SH4 Timer 0.
- #define TMU1 1

  SH4 Timer 1.
- #define TMU2 2

  SH4 Timer 2.
- #define TIMER_ID TMU0

  Which timer does the thread system use?
- #define PRFC0 0

  SH4 Performance Counter.
- #define PRFC1 1

  SH4 Performance Counter.
- #define PMCR_COUNT_CPU_CYCLES 0

  CPU Cycles Count Type.
- #define PMCR_COUNT_RATIO_CYCLES 1

  Ratio Cycles Count Type.
- #define PMCR_INIT_NO_MODE 0x00

  None; Just here to be complete.
- #define PMCR_OPERAND_READ_ACCESS_MODE 0x01

  Quantity; With cache.
- #define PMCR_OPERAND_WRITE_ACCESS_MODE 0x02

  Quantity; With cache.
- #define PMCR_UTLB_MISS_MODE 0x03

  Quantity.
- #define PMCR_OPERAND_CACHE_READ_MISS_MODE 0x04

  Quantity.
- #define PMCR_OPERAND_CACHE_WRITE_MISS_MODE 0x05

  Quantity.
- #define PMCR_INSTRUCTION_FETCH_MODE 0x06

  Quantity; With cache.
- #define PMCR_INSTRUCTION_TLB_MISS_MODE 0x07

  Quantity.
- #define PMCR_INSTRUCTION_CACHE_MISS_MODE 0x08

  Quantity.
- #define PMCR_ALL_OPERAND_ACCESS_MODE 0x09

  Quantity.

- #define PMCR_ALL_INSTRUCTION_FETCH_MODE 0x0a

  Quantity.
- #define PMCR_ON_CHIP_RAM_OPERAND_ACCESS_MODE 0x0b

  Quantity.
- #define PMCR_ON_CHIP_IO_ACCESS_MODE 0x0d

  Quantity.
- #define PMCR_OPERAND_ACCESS_MODE 0x0e

  Quantity; With cache, counts both reads and writes.
- #define PMCR_OPERAND_CACHE_MISS_MODE 0x0f

  Quantity.
- #define PMCR_BRANCH_ISSUED_MODE 0x10

  Quantity; Not the same as branch taken!
- #define PMCR_BRANCH_TAKEN_MODE 0x11

  Quantity.
- #define PMCR_SUBROUTINE_ISSUED_MODE 0x12

  Quantity; Issued a BSR, BSRF, JSR, JSR/N.
- #define PMCR_INSTRUCTION_ISSUED_MODE 0x13

  Quantity.
- #define PMCR_PARALLEL_INSTRUCTION_ISSUED_MODE 0x14

  Quantity.
- #define PMCR_FPU_INSTRUCTION_ISSUED_MODE 0x15

  Quantity.
- #define PMCR_INTERRUPT_COUNTER_MODE 0x16

  Quantity.
- #define PMCR_NMI_COUNTER_MODE 0x17

  Quantity.
- #define PMCR_TRAPA_INSTRUCTION_COUNTER_MODE 0x18

  Quantity.
- #define PMCR_UBC_A_MATCH_MODE 0x19

  Quantity.
- #define PMCR_UBC_B_MATCH_MODE 0x1a

  Quantity.
- #define PMCR_INSTRUCTION_CACHE_FILL_MODE 0x21

  Cycles.
- #define PMCR_OPERAND_CACHE_FILL_MODE 0x22

  Cycles.
- #define PMCR_ELAPSED_TIME_MODE 0x23

  Cycles; For 200MHz CPU: 5ns per count in 1 cycle = 1 count mode, or around 417.715ps per count (increments by 12) in CPU/bus ratio mode.
- #define PMCR_PIPELINE_FREEZE_BY_ICACHE_MISS_MODE 0x24

  Cycles.
- #define PMCR_PIPELINE_FREEZE_BY_DCACHE_MISS_MODE 0x25

  Cycles.
- #define PMCR_PIPELINE_FREEZE_BY_BRANCH_MODE 0x27

  Cycles.
- #define PMCR_PIPELINE_FREEZE_BY_CPU_REGISTER_MODE 0x28

  Cycles.
- #define PMCR_PIPELINE_FREEZE_BY_FPU_MODE 0x29

  Cycles.

Typedefs

- typedef void(∗ timer_primary_callback_t) (irq_context_t ∗)

  Primary timer callback type.

Functions

- int timer_prime (int which, uint32 speed, int interrupts)

  Pre-initialize a timer, but do not start it.
- int timer_start (int which)

  Start a timer.
- int timer_stop (int which)

  Stop a timer.
- uint32 timer_count (int which)

  Obtain the count of a timer.
- int timer_clear (int which)

  Clear the underflow bit of a timer.
- void timer_spin_sleep (int ms)

  Spin-loop sleep function.
- void timer_enable_ints (int which)

  Enable high-priority timer interrupts.
- void timer_disable_ints (int which)

  Disable timer interrupts.
- int timer_ints_enabled (int which)

  Check whether interrupts are enabled on a timer.
- void timer_ms_enable (void)

  Enable the millisecond timer.
- void timer_ms_disable (void)

  Disable the millisecond timer.
- void timer_ms_gettime (uint32 ∗secs, uint32 ∗msecs)

  Get the current uptime of the system.
- uint64 timer_ms_gettime64 (void)

  Get the current uptime of the system (in milliseconds).
- uint64 timer_us_gettime64 (void)

  Get the current uptime of the system (in microseconds).
- timer_primary_callback_t timer_primary_set_callback (timer_primary_callback_t callback)

  Set the primary timer callback.
- void timer_primary_wakeup (uint32 millis)

  Request a primary timer wakeup.
- uint16 perf_cntr_get_config (int which)

  Get a performance counter's settings.
- int perf_cntr_start (int which, int mode, int count_type)

  Start a performance counter.
- int perf_cntr_stop (int which)

  Stop a performance counter.
- int perf_cntr_clear (int which)

  Clear a performance counter.

- uint64 perf_cntr_count (int which)

  Obtain the count of a performance counter.
- void timer_ns_enable (void)

  Enable the nanosecond timer.
- void timer_ns_disable (void)

  Disable the nanosecond timer.
- uint64 timer_ns_gettime64 (void)

  Get the current uptime of the system (in nanoseconds).

9.156.1 Detailed Description

Low-level timer functionality.

This file contains functions for interacting with the timer sources on the SH4. Many of these functions may interfere with thread operation or other such things, and should thus be used with caution. Basically, the only functionality that you might use in practice in here in normal programs is the gettime functions.

Author

Megan Potter

9.156.2 Macro Definition Documentation

TIMER_ID

```
#define TIMER_ID TMU0
```

Which timer does the thread system use?

TMU0

```
#define TMU0 0
```

SH4 Timer 0.

This timer is used for thread operation, and thus is off limits if you want that to work properly.

TMU1

```
#define TMU1 1
```

SH4 Timer 1.

This timer is used for the timer_spin_sleep() function.

TMU2

```
#define TMU2 2
```

SH4 Timer 2.

This timer is used by the various gettime functions in this header.

9.156.3 Typedef Documentation

timer_primary_callback_t

```
typedef void(* timer_primary_callback_t) (irq_context_t *)
```

Primary timer callback type.

9.156.4 Function Documentation

timer_clear()

```
int timer_clear (
            int which )
```

Clear the underflow bit of a timer.

This function clears the underflow bit of a timer if it was set.

Parameters

which	The timer to inspect (i.e, TMU0).

Return values

0	If the underflow bit was clear (prior to calling).
1	If the underflow bit was set (prior to calling).

timer_count()

```
uint32 timer_count (
            int which )
```

Obtain the count of a timer.

This function simply returns the count of the timer.

Parameters

which	The timer to inspect (i.e, TMU0).

Returns

The timer's count.

timer_disable_ints()

```
void timer_disable_ints (
            int which )
```

Disable timer interrupts.

This function disables interrupts on the specified timer.

Parameters

which	The timer to disable interrupts on (i.e, TMU0).

timer_enable_ints()

```
void timer_enable_ints (
            int which )
```

Enable high-priority timer interrupts.

This function enables interrupts on the specified timer.

Parameters

which	The timer to enable interrupts on (i.e, TMU0).

timer_ints_enabled()

```
int timer_ints_enabled (
            int which )
```

Check whether interrupts are enabled on a timer.

This function checks whether or not interrupts are enabled on the specified timer.

Parameters

which	The timer to inspect (i.e, TMU0).

Return values

0	If interrupts are disabled on the timer.
1	If interrupts are enabled on the timer.

timer_ms_disable()

```
void timer_ms_disable (
            void  )
```

Disable the millisecond timer.

This function disables the timer used for the gettime functions. Generally, you will not want to do this, unless you have some need to use the timer TMU2 for something else.

timer_ms_enable()

```
void timer_ms_enable (
            void  )
```

Enable the millisecond timer.

This function enables the timer used for the gettime functions. This is on by default. These functions use TMU2 to do their work.

timer_ms_gettime()

```
void timer_ms_gettime (
            uint32 * secs,
            uint32 * msecs )
```

Get the current uptime of the system.

This function retrieves the number of seconds and milliseconds since KOS was started.

Parameters

secs	A pointer to store the number of seconds since boot into.
msecs	A pointer to store the number of milliseconds past a second since boot.

Note

To get the total number of milliseconds since boot, calculate (*secs * 1000) + *msecs, or use the timer_ms_gettime64() function.

timer_ms_gettime64()

```
uint64 timer_ms_gettime64 (
            void  )
```

Get the current uptime of the system (in milliseconds).

This function retrieves the number of milliseconds since KOS was started. It is equivalent to calling timer_ms_gettime() and combining the number of seconds and milliseconds into one 64-bit value.

Returns

The number of milliseconds since KOS started.

timer_primary_set_callback()

```
timer_primary_callback_t timer_primary_set_callback (
            timer_primary_callback_t callback )
```

Set the primary timer callback.

This function sets the primary timer callback to the specified function pointer. Generally, you should not do this, as the threading system relies on the primary timer to work.

Parameters

callback	The new timer callback (set to NULL to disable).

Returns

The old timer callback.

timer_primary_wakeup()

```
void timer_primary_wakeup (
            uint32 millis )
```

Request a primary timer wakeup.

This function will wake the caller (by calling the primary timer callback) in approximately the number of milliseconds specified. You can only have one timer wakeup scheduled at a time. Any subsequently scheduled wakeups will replace any existing one.

Parameters

millis	The number of milliseconds to schedule for.

timer_prime()

```
int timer_prime (
            int which,
            uint32 speed,
            int interrupts )
```

Pre-initialize a timer, but do not start it.

This function sets up a timer for use, but does not start it.

Parameters

which	The timer to set up (i.e, TMU0).
speed	The number of ticks per second.
interrupts	Set to 1 to receive interrupts when the timer ticks.

Return values

0	On success.

timer_spin_sleep()

```
void timer_spin_sleep (
            int ms )
```

Spin-loop sleep function.

This function is meant as a very accurate delay function, even if threading and interrupts are disabled. It uses TMU1 to sleep.

Parameters

ms	The number of milliseconds to sleep.

timer_start()

```
int timer_start (
            int which )
```

Start a timer.

This function starts a timer that has been initialized with timer_prime(), starting raising interrupts if applicable.

Parameters

which	The timer to start (i.e, TMU0).

Return values

0	On success.

timer_stop()

```
int timer_stop (
            int which )
```

Stop a timer.

This function stops a timer that was started with timer_start(), and as a result stops interrupts coming in from the timer.

Parameters

which	The timer to stop (i.e, TMU0).

Return values

0	On success.

timer_us_gettime64()

```
uint64 timer_us_gettime64 (
            void  )
```

Get the current uptime of the system (in microseconds).

This function retrieves the number of microseconds since KOS was started. It should be more precise, in theory, than timer_ms_gettime64(), but the exact amount of preciseness is undetermined.

Returns

The number of microseconds since KOS started.

9.157 timer.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/timer.h
00004    (c)2000-2001 Megan Potter
00005
00006 */
00007
00008 /** \file   arch/timer.h
00009     \brief  Low-level timer functionality.
00010
00011     This file contains functions for interacting with the timer sources on the
00012     SH4. Many of these functions may interfere with thread operation or other
00013     such things, and should thus be used with caution. Basically, the only
00014     functionality that you might use in practice in here in normal programs is
00015     the gettime functions.
00016
00017     \author Megan Potter
00018 */
00019
00020 #ifndef __ARCH_TIMER_H
00021 #define __ARCH_TIMER_H
00022
00023 #include <sys/cdefs.h>
00024 __BEGIN_DECLS
00025
00026 #include <arch/types.h>
00027 #include <arch/irq.h>
00028
00029 /* Timer sources -- we get four on the SH4 */
00030
00031 /** \brief  SH4 Timer 0.
00032
00033     This timer is used for thread operation, and thus is off limits if you want
00034     that to work properly.
00035 */
00036 #define TMU0    0
00037
00038 /** \brief  SH4 Timer 1.
00039
00040     This timer is used for the timer_spin_sleep() function.
00041 */
00042 #define TMU1    1
00043
00044 /** \brief  SH4 Timer 2.
00045
00046     This timer is used by the various gettime functions in this header.
00047 */
00048 #define TMU2    2
00049
00050 /** \brief  Which timer does the thread system use? */
00051 #define TIMER_ID TMU0
00052
00053 /** \brief  Pre-initialize a timer, but do not start it.
00054
00055     This function sets up a timer for use, but does not start it.
00056
00057     \param  which           The timer to set up (i.e, \ref TMU0).
00058     \param  speed           The number of ticks per second.
00059     \param  interrupts      Set to 1 to receive interrupts when the timer ticks.
00060     \retval 0               On success.
00061 */
00062 int timer_prime(int which, uint32 speed, int interrupts);
00063
00064 /** \brief  Start a timer.
00065
00066     This function starts a timer that has been initialized with timer_prime(),
00067     starting raising interrupts if applicable.
00068
00069     \param  which           The timer to start (i.e, \ref TMU0).
00070     \retval 0               On success.
00071 */
00072 int timer_start(int which);
00073
00074 /** \brief  Stop a timer.
00075
00076     This function stops a timer that was started with timer_start(), and as a
```

```
00077     result stops interrupts coming in from the timer.
00078
00079     \param  which           The timer to stop (i.e, \ref TMU0).
00080     \retval 0               On success.
00081 */
00082 int timer_stop(int which);
00083
00084 /** \brief  Obtain the count of a timer.
00085
00086     This function simply returns the count of the timer.
00087
00088     \param  which           The timer to inspect (i.e, \ref TMU0).
00089     \return                 The timer's count.
00090 */
00091 uint32 timer_count(int which);
00092
00093 /** \brief  Clear the underflow bit of a timer.
00094
00095     This function clears the underflow bit of a timer if it was set.
00096
00097     \param  which           The timer to inspect (i.e, \ref TMU0).
00098     \retval 0               If the underflow bit was clear (prior to calling).
00099     \retval 1               If the underflow bit was set (prior to calling).
00100 */
00101 int timer_clear(int which);
00102
00103 /** \brief  Spin-loop sleep function.
00104
00105     This function is meant as a very accurate delay function, even if threading
00106     and interrupts are disabled. It uses \ref TMU1 to sleep.
00107
00108     \param  ms              The number of milliseconds to sleep.
00109 */
00110 void timer_spin_sleep(int ms);
00111
00112 /** \brief  Enable high-priority timer interrupts.
00113
00114     This function enables interrupts on the specified timer.
00115
00116     \param  which           The timer to enable interrupts on (i.e, \ref TMU0).
00117 */
00118 void timer_enable_ints(int which);
00119
00120 /** \brief  Disable timer interrupts.
00121
00122     This function disables interrupts on the specified timer.
00123
00124     \param  which           The timer to disable interrupts on (i.e, \ref TMU0).
00125 */
00126 void timer_disable_ints(int which);
00127
00128 /** \brief  Check whether interrupts are enabled on a timer.
00129
00130     This function checks whether or not interrupts are enabled on the specified
00131     timer.
00132
00133     \param  which           The timer to inspect (i.e, \ref TMU0).
00134     \retval 0               If interrupts are disabled on the timer.
00135     \retval 1               If interrupts are enabled on the timer.
00136 */
00137 int timer_ints_enabled(int which);
00138
00139 /** \brief  Enable the millisecond timer.
00140
00141     This function enables the timer used for the gettime functions. This is on
00142     by default. These functions use \ref TMU2 to do their work.
00143 */
00144 void timer_ms_enable(void);
00145
00146 /** \brief  Disable the millisecond timer.
00147
00148     This function disables the timer used for the gettime functions. Generally,
00149     you will not want to do this, unless you have some need to use the timer
00150     \ref TMU2 for something else.
00151 */
00152 void timer_ms_disable(void);
00153
00154 /** \brief  Get the current uptime of the system.
00155
00156     This function retrieves the number of seconds and milliseconds since KOS was
00157     started.
```

```
00158
00159     \param  secs            A pointer to store the number of seconds since boot
00160                             into.
00161     \param  msecs           A pointer to store the number of milliseconds past
00162                             a second since boot.
00163     \note                   To get the total number of milliseconds since boot,
00164                             calculate (*secs * 1000) + *msecs, or use the
00165                             timer_ms_gettime64() function.
00166 */
00167 void timer_ms_gettime(uint32 *secs, uint32 *msecs);
00168
00169 /** \brief  Get the current uptime of the system (in milliseconds).
00170
00171     This function retrieves the number of milliseconds since KOS was started. It
00172     is equivalent to calling timer_ms_gettime() and combining the number of
00173     seconds and milliseconds into one 64-bit value.
00174
00175     \return                 The number of milliseconds since KOS started.
00176 */
00177 uint64 timer_ms_gettime64(void);
00178
00179 /** \brief  Get the current uptime of the system (in microseconds).
00180
00181     This function retrieves the number of microseconds since KOS was started. It
00182     should be more precise, in theory, than timer_ms_gettime64(), but the exact
00183     amount of preciseness is undetermined.
00184
00185     \return                 The number of microseconds since KOS started.
00186 */
00187 uint64 timer_us_gettime64(void);
00188
00189 /** \brief  Primary timer callback type. */
00190 typedef void (*timer_primary_callback_t)(irq_context_t *);
00191
00192 /** \brief  Set the primary timer callback.
00193
00194     This function sets the primary timer callback to the specified function
00195     pointer. Generally, you should not do this, as the threading system relies
00196     on the primary timer to work.
00197
00198     \param  callback        The new timer callback (set to NULL to disable).
00199     \return                 The old timer callback.
00200 */
00201 timer_primary_callback_t timer_primary_set_callback(timer_primary_callback_t callback);
00202
00203 /** \brief  Request a primary timer wakeup.
00204
00205     This function will wake the caller (by calling the primary timer callback)
00206     in approximately the number of milliseconds specified. You can only have one
00207     timer wakeup scheduled at a time. Any subsequently scheduled wakeups will
00208     replace any existing one.
00209
00210     \param  millis          The number of milliseconds to schedule for.
00211 */
00212 void timer_primary_wakeup(uint32 millis);
00213
00214 /* \cond */
00215 /* Init function */
00216 int timer_init(void);
00217
00218 /* Shutdown */
00219 void timer_shutdown(void);
00220 /* \endcond */
00221
00222 /** \defgroup   perf_counters Performance Counters
00223     The performance counter API exposes the SH4's hardware profiling registers,
00224     which consist of two different sets of independently operable 64-bit
00225     counters.
00226 */
00227
00228 /** \brief  SH4 Performance Counter.
00229     \ingroup perf_counters
00230
00231     This counter is used by the ns_gettime function in this header.
00232 */
00233 #define PRFC0   0
00234
00235 /** \brief  SH4 Performance Counter.
00236     \ingroup perf_counters
00237
00238     A counter that is not used by KOS.
```

```
00239 */
00240 #define PRFC1   1
00241
00242 /** \brief  CPU Cycles Count Type.
00243     \ingroup perf_counters
00244
00245     Count cycles. At 5 ns increments, a 48-bit cycle counter can
00246     run continuously for 16.33 days.
00247 */
00248 #define PMCR_COUNT_CPU_CYCLES 0
00249
00250 /** \brief  Ratio Cycles Count Type.
00251     \ingroup perf_counters
00252
00253     CPU/bus ratio mode where cycles (where T = C x B / 24 and T is time,
00254     C is count, and B is time of one bus cycle).
00255 */
00256 #define PMCR_COUNT_RATIO_CYCLES 1
00257
00258 /** \defgroup   perf_counters_modes Performance Counter Modes
00259     This is the list of modes that are allowed to be passed into the perf_cntr_start()
00260     function, representing different things you want to count.
00261     \ingroup perf_counters
00262     @{
00263 */
00264 /*                MODE DEFINITION                  VALUE   MEASURMENT TYPE & NOTES */
00265 #define PMCR_INIT_NO_MODE                          0x00 /**< \brief None; Just here to be complete */
00266 #define PMCR_OPERAND_READ_ACCESS_MODE              0x01 /**< \brief Quantity; With cache */
00267 #define PMCR_OPERAND_WRITE_ACCESS_MODE             0x02 /**< \brief Quantity; With cache */
00268 #define PMCR_UTLB_MISS_MODE                        0x03 /**< \brief Quantity */
00269 #define PMCR_OPERAND_CACHE_READ_MISS_MODE          0x04 /**< \brief Quantity */
00270 #define PMCR_OPERAND_CACHE_WRITE_MISS_MODE         0x05 /**< \brief Quantity */
00271 #define PMCR_INSTRUCTION_FETCH_MODE                0x06 /**< \brief Quantity; With cache */
00272 #define PMCR_INSTRUCTION_TLB_MISS_MODE             0x07 /**< \brief Quantity */
00273 #define PMCR_INSTRUCTION_CACHE_MISS_MODE           0x08 /**< \brief Quantity */
00274 #define PMCR_ALL_OPERAND_ACCESS_MODE               0x09 /**< \brief Quantity */
00275 #define PMCR_ALL_INSTRUCTION_FETCH_MODE            0x0a /**< \brief Quantity */
00276 #define PMCR_ON_CHIP_RAM_OPERAND_ACCESS_MODE       0x0b /**< \brief Quantity */
00277 /* No 0x0c */
00278 #define PMCR_ON_CHIP_IO_ACCESS_MODE                0x0d /**< \brief Quantity */
00279 #define PMCR_OPERAND_ACCESS_MODE                   0x0e /**< \brief Quantity; With cache, counts both
      reads and writes */
00280 #define PMCR_OPERAND_CACHE_MISS_MODE               0x0f /**< \brief Quantity */
00281 #define PMCR_BRANCH_ISSUED_MODE                    0x10 /**< \brief Quantity; Not the same as branch
      taken! */
00282 #define PMCR_BRANCH_TAKEN_MODE                     0x11 /**< \brief Quantity */
00283 #define PMCR_SUBROUTINE_ISSUED_MODE                0x12 /**< \brief Quantity; Issued a BSR, BSRF, JSR,
      JSR/N */
00284 #define PMCR_INSTRUCTION_ISSUED_MODE               0x13 /**< \brief Quantity */
00285 #define PMCR_PARALLEL_INSTRUCTION_ISSUED_MODE      0x14 /**< \brief Quantity */
00286 #define PMCR_FPU_INSTRUCTION_ISSUED_MODE           0x15 /**< \brief Quantity */
00287 #define PMCR_INTERRUPT_COUNTER_MODE                0x16 /**< \brief Quantity */
00288 #define PMCR_NMI_COUNTER_MODE                      0x17 /**< \brief Quantity */
00289 #define PMCR_TRAPA_INSTRUCTION_COUNTER_MODE        0x18 /**< \brief Quantity */
00290 #define PMCR_UBC_A_MATCH_MODE                      0x19 /**< \brief Quantity */
00291 #define PMCR_UBC_B_MATCH_MODE                      0x1a /**< \brief Quantity */
00292 /* No 0x1b-0x20 */
00293 #define PMCR_INSTRUCTION_CACHE_FILL_MODE           0x21 /**< \brief Cycles */
00294 #define PMCR_OPERAND_CACHE_FILL_MODE               0x22 /**< \brief Cycles */
00295 #define PMCR_ELAPSED_TIME_MODE                     0x23 /**< \brief Cycles; For 200MHz CPU: 5ns per count
      in 1 cycle = 1 count mode, or around 417.715ps per count (increments by 12) in CPU/bus ratio mode */
00296 #define PMCR_PIPELINE_FREEZE_BY_ICACHE_MISS_MODE   0x24 /**< \brief Cycles */
00297 #define PMCR_PIPELINE_FREEZE_BY_DCACHE_MISS_MODE   0x25 /**< \brief Cycles */
00298 /* No 0x26 */
00299 #define PMCR_PIPELINE_FREEZE_BY_BRANCH_MODE        0x27 /**< \brief Cycles */
00300 #define PMCR_PIPELINE_FREEZE_BY_CPU_REGISTER_MODE  0x28 /**< \brief Cycles */
00301 #define PMCR_PIPELINE_FREEZE_BY_FPU_MODE           0x29 /**< \brief Cycles */
00302 /** @} */
00303
00304
00305 /** \brief  Get a performance counter's settings.
00306     \ingroup perf_counters
00307
00308     This function returns a performance counter's settings.
00309
00310     \param  which           The performance counter (i.e, \ref PRFC0 or PRFC1).
00311     \retval 0               On success.
00312 */
00313 uint16 perf_cntr_get_config(int which);
00314
00315 /** \brief  Start a performance counter.
```

```
00316     \ingroup perf_counters
00317
00318     This function starts a performance counter
00319
00320     \param  which           The counter to start (i.e, \ref PRFC0 or PRFC1).
00321     \param  mode            Use one of the 33 modes listed above.
00322     \param  count_type      PMCR_COUNT_CPU_CYCLES or PMCR_COUNT_RATIO_CYCLES.
00323     \retval 0               On success.
00324 */
00325 int perf_cntr_start(int which, int mode, int count_type);
00326
00327 /** \brief  Stop a performance counter.
00328     \ingroup perf_counters
00329
00330     This function stops a performance counter that was started with perf_cntr_start().
00331     Stopping a counter retains its count. To clear the count use perf_cntr_clear().
00332
00333     \param  which           The counter to stop (i.e, \ref PRFC0 or PRFC1).
00334     \retval 0               On success.
00335 */
00336 int perf_cntr_stop(int which);
00337
00338 /** \brief  Clear a performance counter.
00339     \ingroup perf_counters
00340
00341     This function clears a performance counter. It resets its count to zero.
00342     This function stops the counter before clearing it because you cant clear
00343     a running counter.
00344
00345     \param  which           The counter to clear (i.e, \ref PRFC0 or PRFC1).
00346     \retval 0               On success.
00347 */
00348 int perf_cntr_clear(int which);
00349
00350 /** \brief  Obtain the count of a performance counter.
00351     \ingroup perf_counters
00352
00353     This function simply returns the count of the counter.
00354
00355     \param  which           The counter to read (i.e, \ref PRFC0 or PRFC1).
00356     \return                 The counter's count.
00357 */
00358 uint64 perf_cntr_count(int which);
00359
00360 /** \brief  Enable the nanosecond timer.
00361     \ingroup perf_counters
00362
00363     This function enables the performance counter used for the timer_ns_gettime64()
00364     function. This is on by default. The function uses \ref PRFC0 to do the work.
00365 */
00366 void timer_ns_enable(void);
00367
00368 /** \brief  Disable the nanosecond timer.
00369     \ingroup perf_counters
00370
00371     This function disables the performance counter used for the timer_ns_gettime64()
00372     function. Generally, you will not want to do this, unless you have some need to use
00373     the counter \ref PRFC0 for something else.
00374 */
00375 void timer_ns_disable(void);
00376
00377 /** \brief  Get the current uptime of the system (in nanoseconds).
00378     \ingroup perf_counters
00379
00380     This function retrieves the number of nanoseconds since KOS was started.
00381
00382     \return                 The number of nanoseconds since KOS started.
00383 */
00384 uint64 timer_ns_gettime64(void);
00385
00386 __END_DECLS
00387
00388 #endif  /* __ARCH_TIMER_H */
00389
```

9.158 kernel/arch/dreamcast/include/arch/types.h File Reference

Common integer types.

```
#include <sys/cdefs.h>
#include <stddef.h>
#include <sys/_types.h>
```

Macros

- #define BYTE_ORDER LITTLE_ENDIAN

  Endianness definition – Little Endian.

Typedefs

- typedef unsigned long long uint64

  64-bit unsigned integer
- typedef unsigned long uint32

  32-bit unsigned integer
- typedef unsigned short uint16

  16-bit unsigned integer
- typedef unsigned char uint8

  8-bit unsigned integer
- typedef long long int64

  64-bit signed integer
- typedef long int32

  32-bit signed integer
- typedef short int16

  16-bit signed integer
- typedef char int8

  8-bit signed integer
- typedef volatile uint64 vuint64

  64-bit volatile unsigned type
- typedef volatile uint32 vuint32

  32-bit volatile unsigned type
- typedef volatile uint16 vuint16

  16-bit volatile unsigned type
- typedef volatile uint8 vuint8

  8-bit volatile unsigned type
- typedef volatile int64 vint64

  64-bit volatile signed type
- typedef volatile int32 vint32

  32-bit volatile signed type
- typedef volatile int16 vint16

  16-bit volatile signed type
- typedef volatile int8 vint8

  8-bit volatile signed type
- typedef uint32 ptr_t

  Pointer arithmetic type.

- typedef unsigned char u_char
  BSD-style unsigned char.
- typedef unsigned short u_short
  BSD-style unsigned short.
- typedef unsigned int u_int
  BSD-style unsigned integer.
- typedef unsigned long u_long
  BSD-style unsigned long.
- typedef unsigned short ushort
  BSD-style unsigned short.
- typedef unsigned int uint
  BSD-style unsigned integer.
- typedef int handle_t
  Generic "handle" type.
- typedef handle_t tid_t
  Thread ID type.
- typedef handle_t prio_t
  Priority value type.

9.158.1 Detailed Description

Common integer types.

This file contains typedefs for some common/useful integer types. These types include ones that tell you exactly how long they are, as well as some BSD-isms.

Author

Megan Potter

9.158.2 Macro Definition Documentation

BYTE_ORDER

```
#define BYTE_ORDER LITTLE_ENDIAN
```

Endianness definition – Little Endian.

9.158.3 Typedef Documentation

handle_t

```
typedef int handle_t
```

Generic "handle" type.

int16

```
typedef short int16
```

16-bit signed integer

int32

```
typedef long int32
```

32-bit signed integer

int64

```
typedef long long int64
```

64-bit signed integer

int8

```
typedef char int8
```

8-bit signed integer

prio_t

```
typedef handle_t prio_t
```

Priority value type.

ptr_t

```
typedef uint32 ptr_t
```

Pointer arithmetic type.

tid_t

```
typedef handle_t tid_t
```

Thread ID type.

u_char

```
typedef unsigned char u_char
```

BSD-style unsigned char.

u_int

```
typedef unsigned int u_int
```

BSD-style unsigned integer.

u_long

```
typedef unsigned long u_long
```

BSD-style unsigned long.

u_short

```
typedef unsigned short u_short
```

BSD-style unsigned short.

uint

```
typedef unsigned int uint
```

BSD-style unsigned integer.

uint16

```
typedef unsigned short uint16
```

16-bit unsigned integer

uint32

```
typedef unsigned long uint32
```

32-bit unsigned integer

uint64

```
typedef unsigned long long uint64
```

64-bit unsigned integer

uint8

```
typedef unsigned char uint8
```

8-bit unsigned integer

ushort

```
typedef unsigned short ushort
```

BSD-style unsigned short.

vint16

```
typedef volatile int16 vint16
```

16-bit volatile signed type

vint32

```
typedef volatile int32 vint32
```

32-bit volatile signed type

vint64

```
typedef volatile int64 vint64
```

64-bit volatile signed type

vint8

```
typedef volatile int8 vint8
```

8-bit volatile signed type

vuint16

```
typedef volatile uint16 vuint16
```

16-bit volatile unsigned type

vuint32

```
typedef volatile uint32 vuint32
```

32-bit volatile unsigned type

vuint64

```
typedef volatile uint64 vuint64
```

64-bit volatile unsigned type

vuint8

```
typedef volatile uint8 vuint8
```

8-bit volatile unsigned type

9.159 types.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/types.h
00004    (c)2000-2001 Megan Potter
00005
00006 */
00007
00008 /** \file   arch/types.h
00009     \brief  Common integer types.
00010
00011     This file contains typedefs for some common/useful integer types. These
00012     types include ones that tell you exactly how long they are, as well as some
00013     BSD-isms.
00014
00015     \author Megan Potter
00016 */
00017
00018 #ifndef __ARCH_TYPES_H
00019 #define __ARCH_TYPES_H
00020
00021 #include <sys/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 #include <stddef.h>
00025
00026 /* Generic types */
00027 typedef unsigned long long uint64;  /**< \brief 64-bit unsigned integer */
00028 typedef unsigned long uint32;       /**< \brief 32-bit unsigned integer */
00029 typedef unsigned short uint16;      /**< \brief 16-bit unsigned integer */
00030 typedef unsigned char uint8;        /**< \brief 8-bit unsigned integer */
```

```
00031 typedef long long int64;            /**< \brief 64-bit signed integer */
00032 typedef long int32;                 /**< \brief 32-bit signed integer */
00033 typedef short int16;                /**< \brief 16-bit signed integer */
00034 typedef char int8;                  /**< \brief 8-bit signed integer */
00035
00036 /* Volatile types */
00037 typedef volatile uint64 vuint64;    /**< \brief 64-bit volatile unsigned type */
00038 typedef volatile uint32 vuint32;    /**< \brief 32-bit volatile unsigned type */
00039 typedef volatile uint16 vuint16;    /**< \brief 16-bit volatile unsigned type */
00040 typedef volatile uint8 vuint8;      /**< \brief 8-bit volatile unsigned type */
00041 typedef volatile int64 vint64;      /**< \brief 64-bit volatile signed type */
00042 typedef volatile int32 vint32;      /**< \brief 32-bit volatile signed type */
00043 typedef volatile int16 vint16;      /**< \brief 16-bit volatile signed type */
00044 typedef volatile int8 vint8;        /**< \brief 8-bit volatile signed type */
00045
00046 /* Pointer arithmetic types */
00047 typedef uint32 ptr_t;               /**< \brief Pointer arithmetic type */
00048
00049 /* another format for type names */
00050 #ifndef _BSDTYPES_DEFINED
00051
00052 /* \cond */
00053 #define __u_char_defined
00054 #define __u_short_defined
00055 #define __u_int_defined
00056 #define __u_long_defined
00057 #define __ushort_defined
00058 #define __uint_defined
00059 /* \endcond */
00060
00061 typedef unsigned char   u_char;     /**< \brief BSD-style unsigned char */
00062 typedef unsigned short  u_short;    /**< \brief BSD-style unsigned short */
00063 typedef unsigned int    u_int;      /**< \brief BSD-style unsigned integer */
00064 typedef unsigned long   u_long;     /**< \brief BSD-style unsigned long */
00065 typedef unsigned short  ushort;     /**< \brief BSD-style unsigned short */
00066 typedef unsigned int    uint;       /**< \brief BSD-style unsigned integer */
00067
00068 /* \cond */
00069 #define _BSDTYPES_DEFINED
00070 /* \endcond */
00071
00072 #endif  /* _BSDTYPES_DEFINED */
00073
00074 /* This type may be used for any generic handle type that is allowed
00075    to be negative (for errors) and has no specific bit count
00076    restraints. */
00077 typedef int handle_t;               /**< \brief Generic "handle" type */
00078
00079 /* Thread and priority types */
00080 typedef handle_t tid_t;             /**< \brief Thread ID type */
00081 typedef handle_t prio_t;            /**< \brief Priority value type */
00082
00083 #ifndef BYTE_ORDER
00084 /* Make sure to pull in the base endianness defines... */
00085 #ifndef LITTLE_ENDIAN
00086 #include <sys/_types.h>
00087 #endif
00088
00089 /** \brief  Endianness definition -- Little Endian */
00090 #define BYTE_ORDER  LITTLE_ENDIAN
00091 #endif
00092
00093 __END_DECLS
00094
00095 #endif  /* __ARCH_TYPES_H */
```

9.160 kernel/arch/dreamcast/include/arch/wdt.h File Reference

Watchdog Timer API.

```
#include <sys/cdefs.h>
#include <stdint.h>
```

Typedefs

- typedef void(∗ wdt_callback) (void ∗user_data)

Enumerations

- enum WDT_CLK_DIV {
  WDT_CLK_DIV_32 , WDT_CLK_DIV_64 , WDT_CLK_DIV_128 , WDT_CLK_DIV_256 ,
  WDT_CLK_DIV_512 , WDT_CLK_DIV_1024 , WDT_CLK_DIV_2048 , WDT_CLK_DIV_4096 }

  Clock divider settings.
- enum WDT_RST { WDT_RST_POWER_ON , WDT_RST_MANUAL }

  Reset signal type.

Functions

- void wdt_enable_timer (uint8_t initial_count, uint32_t microsec_period, uint8_t irq_prio, wdt_callback callback, void ∗user_data)

  Enables the WDT as an interval timer.
- void wdt_enable_watchdog (uint8_t initial_count, WDT_CLK_DIV clk_config, WDT_RST reset_select)

  Enables the WDT in watchdog mode.
- uint8_t wdt_get_counter (void)

  Fetches the counter value.
- void wdt_set_counter (uint8_t value)

  Sets the counter value.
- void wdt_pet (void)

  Resets the counter value.
- void wdt_disable (void)

  Disables the WDT.
- int wdt_is_enabled (void)

  Returns whether the WDT is enabled.

9.160.1 Detailed Description

Watchdog Timer API.

This file provides an API built around utilizing the SH4's watchdog timer. There are two different modes of operation which are supported:

- watchdog mode: counter overflow causes a reset interrupt
- interval timer mode: counter overflow invokes a callback function

To start the WDT in watchdog mode, use wdt_enable_watchdog(). To use the WDT as a general-purpose interval timer, use wdt_enable_timer().

The timer can be stopped in either mode by calling wdt_disable_timer().

Warning

Once the WDT has been enabled, special care must be taken to disable it when exiting from the application. If left enabled, the WDT will continue running beyond the lifetime of the application, causing either a reset or an unhandled exception (depending on which mode was used), preventing you from gracefully returning to a DC-Load session when testing.

See also

timer.h, rtc.h

Author

Falco Girgis

9.160.2 Typedef Documentation

wdt_callback

```
typedef void(* wdt_callback) (void *user_data)
```

9.160.3 Enumeration Type Documentation

WDT_CLK_DIV

```
enum WDT_CLK_DIV
```

Clock divider settings.

Denominators used to set the frequency divider for the input clock to the WDT.

Enumerator

WDT_CLK_DIV_32	
WDT_CLK_DIV_64	Period: 41us.
WDT_CLK_DIV_128	Period: 82us.
WDT_CLK_DIV_256	Period: 164us.
WDT_CLK_DIV_512	Period: 328us.
WDT_CLK_DIV_1024	Period: 656us.
WDT_CLK_DIV_2048	Period: 1.31ms.
WDT_CLK_DIV_4096	Period: 2.62ms. Period: 5.25ms

WDT_RST

```
enum WDT_RST
```

Reset signal type.

Specifies the kind of reset to be performed when the WDT overflows in watchdog mode.

Enumerator

WDT_RST_POWER_ON	
WDT_RST_MANUAL	Power-On Reset. Manual Reset

9.160.4 Function Documentation

wdt_disable()

```
void wdt_disable (
             void  )
```

Disables the WDT.

Disables the WDT if it was previously enabled, otherwise does nothing.

See also

wdt_enable_timer(), wdt_enable_watchdog()

wdt_enable_timer()

```
void wdt_enable_timer (
             uint8_t initial_count,
             uint32_t microsec_period,
             uint8_t irq_prio,
             wdt_callback callback,
             void * user_data )
```

Enables the WDT as an interval timer.

Stops the WDT if it was previously running and reconfigures it to be used as a generic interval timer, calling the given callback periodically at the requested interval (or as close to it as possible without calling it prematurely).

Note

The internal resolution for each tick of the WDT in this mode is 41us, meaning a requested `microsec_period` of 100us will result in an actual callback interval of 123us.

Warning

`callback` is invoked within an interrupt context, meaning that special care should be taken to not perform any logic requiring additional interrupts. Data that is accessed from both within and outside of the callback should be atomic or protected by a lock.

Parameters

initial_count	Initial value of the WDT counter (Normally 0).
microsec_period	Timer callback interval in microseconds
irq_prio	Priority for the interval timer IRQ (1-15)
callback	User function to invoke periodically
user_data	Arbitrary user-provided data for the callback

See also

wdt_disable()

wdt_enable_watchdog()

```
void wdt_enable_watchdog (
            uint8_t initial_count,
            WDT_CLK_DIV clk_config,
            WDT_RST reset_select )
```

Enables the WDT in watchdog mode.

Stops the WDT if it was previously running and reconfigures it to be used as a typical watchdog timer, generating a resest interrupt upon counter overflow. To prevent this from happening, the user should be periodically resetting the counter.

Note

Keep in mind the speed of the WDT. With a range of 41us to 5.2ms, the WDT will overflow before a single frame in a typical game.

Parameters

initial_count	Initial value of the WDT counter (Normally 0)
clk_config	Clock divider to set watchdog period
reset_select	The type of reset generated upon overflow

See also

wdt_disable()

wdt_get_counter()

```
uint8_t wdt_get_counter (
            void  )
```

Fetches the counter value.

Returns the current 8-bit value of the WDT counter.

Returns

Current counter value

See also

wdt_set_counter()

wdt_is_enabled()

```
int wdt_is_enabled (
             void  )
```

Returns whether the WDT is enabled.

Checks to see whether the WDT has been enabled.

Returns

1 if enabled, 0 if disabled

wdt_pet()

```
void wdt_pet (
             void  )
```

Resets the counter value.

"Petting" or "kicking" the WDT is the same thing as resetting its counter value to 0.

See also

wdt_set_counter()

wdt_set_counter()

```
void wdt_set_counter (
             uint8_t value )
```

Sets the counter value.

Sets the current 8-bit value of the WDT counter.

Parameters

value	New value for the counter

See also

wdt_get_counter(), wdt_pet()

9.161 wdt.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    arch/dreamcast/include/wdt.h
00004    Copyright (c) 2023 Falco Girgis
00005
00006 */
00007
00008 /** \file   arch/wdt.h
00009     \brief  Watchdog Timer API
00010
00011     This file provides an API built around utilizing the SH4's watchdog timer.
00012     There are two different modes of operation which are supported:
00013         - watchdog mode: counter overflow causes a reset interrupt
00014         - interval timer mode: counter overflow invokes a callback function
00015
00016     To start the WDT in watchdog mode, use wdt_enable_watchdog(). To use the
00017     WDT as a general-purpose interval timer, use wdt_enable_timer().
00018
00019     The timer can be stopped in either mode by calling wdt_disable_timer().
00020
00021     \warning
00022     Once the WDT has been enabled, special care must be taken to disable it
00023     when exiting from the application. If left enabled, the WDT will continue
00024     running beyond the lifetime of the application, causing either a reset or
00025     an unhandled exception (depending on which mode was used), preventing you
00026     from gracefully returning to a DC-Load session when testing.
00027
00028     \sa timer.h, rtc.h
00029
00030     \author Falco Girgis
00031 */
00032
00033 #ifndef __ARCH_WDT_H
00034 #define __ARCH_WDT_H
00035
00036 #include <sys/cdefs.h>
00037 __BEGIN_DECLS
00038
00039 #include <stdint.h>
00040
00041 /** \brief Clock divider settings
00042
00043     Denominators used to set the frequency divider
00044     for the input clock to the WDT.
00045  */
00046 typedef enum WDT_CLK_DIV {
00047     WDT_CLK_DIV_32,      /** \brief Period: 41us */
00048     WDT_CLK_DIV_64,      /** \brief Period: 82us */
00049     WDT_CLK_DIV_128,     /** \brief Period: 164us */
00050     WDT_CLK_DIV_256,     /** \brief Period: 328us */
00051     WDT_CLK_DIV_512,     /** \brief Period: 656us */
00052     WDT_CLK_DIV_1024,    /** \brief Period: 1.31ms */
00053     WDT_CLK_DIV_2048,    /** \brief Period: 2.62ms */
00054     WDT_CLK_DIV_4096     /** \brief Period: 5.25ms */
00055 } WDT_CLK_DIV;
00056
00057 /** \brief Reset signal type
00058
00059     Specifies the kind of reset to be performed when the WDT
00060     overflows in watchdog mode.
00061 */
00062 typedef enum WDT_RST {
00063     WDT_RST_POWER_ON,    /** \brief Power-On Reset */
00064     WDT_RST_MANUAL       /** \brief Manual Reset */
00065 } WDT_RST;
00066
00067 /* \brief WDT interval timer callback function type */
00068 typedef void (*wdt_callback)(void *user_data);
00069
```

```
00070 /** \brief  Enables the WDT as an interval timer
00071
00072     Stops the WDT if it was previously running and reconfigures it
00073     to be used as a generic interval timer, calling the given callback
00074     periodically at the requested interval (or as close to it as possible
00075     without calling it prematurely).
00076
00077     \note
00078     The internal resolution for each tick of the WDT in this mode is
00079     41us, meaning a requested \p microsec_period of 100us will result
00080     in an actual callback interval of 123us.
00081
00082     \warning
00083     \p callback is invoked within an interrupt context, meaning that
00084     special care should be taken to not perform any logic requiring
00085     additional interrupts. Data that is accessed from both within
00086     and outside of the callback should be atomic or protected by a
00087     lock.
00088
00089     \param  initial_count   Initial value of the WDT counter (Normally 0).
00090     \param  microsec_period Timer callback interval in microseconds
00091     \param  irq_prio        Priority for the interval timer IRQ (1-15)
00092     \param  callback        User function to invoke periodically
00093     \param  user_data       Arbitrary user-provided data for the callback
00094
00095     \sa wdt_disable()
00096 */
00097 void wdt_enable_timer(uint8_t initial_count,
00098                       uint32_t microsec_period,
00099                       uint8_t irq_prio,
00100                       wdt_callback callback,
00101                       void *user_data);
00102
00103 /** \brief  Enables the WDT in watchdog mode
00104
00105     Stops the WDT if it was previously running and reconfigures it
00106     to be used as a typical watchdog timer, generating a resest
00107     interrupt upon counter overflow. To prevent this from happening,
00108     the user should be periodically resetting the counter.
00109
00110     \note
00111     Keep in mind the speed of the WDT. With a range of 41us to 5.2ms,
00112     the WDT will overflow before a single frame in a typical game.
00113
00114     \param  initial_count   Initial value of the WDT counter (Normally 0)
00115     \param  clk_config      Clock divider to set watchdog period
00116     \param  reset_select    The type of reset generated upon overflow
00117
00118     \sa wdt_disable()
00119 */
00120 void wdt_enable_watchdog(uint8_t initial_count,
00121                          WDT_CLK_DIV clk_config,
00122                          WDT_RST reset_select);
00123
00124 /** \brief  Fetches the counter value
00125
00126     Returns the current 8-bit value of the WDT counter.
00127
00128     \return     Current counter value
00129
00130     \sa wdt_set_counter()
00131 */
00132 uint8_t wdt_get_counter(void);
00133
00134 /** \brief  Sets the counter value
00135
00136     Sets the current 8-bit value of the WDT counter.
00137
00138     \param  value       New value for the counter
00139
00140     \sa wdt_get_counter(), wdt_pet()
00141 */
00142 void wdt_set_counter(uint8_t value);
00143
00144 /** \brief  Resets the counter value
00145
00146     "Petting" or "kicking" the WDT is the same thing as
00147     resetting its counter value to 0.
00148
00149     \sa wdt_set_counter()
00150 */
```

```
00151 void wdt_pet(void);
00152
00153 /** \brief Disables the WDT
00154
00155     Disables the WDT if it was previously enabled,
00156     otherwise does nothing.
00157
00158     \sa wdt_enable_timer(), wdt_enable_watchdog()
00159 */
00160 void wdt_disable(void);
00161
00162 /** \brief  Returns whether the WDT is enabled
00163
00164     Checks to see whether the WDT has been enabled.
00165
00166     \return     1 if enabled, 0 if disabled
00167 */
00168 int wdt_is_enabled(void);
00169
00170 __END_DECLS
00171
00172 #endif  /* __ARCH_WDT_H */
```

9.162 kernel/arch/dreamcast/include/dc/asic.h File Reference

Dreamcast ASIC event handling support.

```
#include <sys/cdefs.h>
#include <stdint.h>
```

Macros

- #define ASIC_EVT_PVR_RENDERDONE_VIDEO 0x0000

  Video render stage completed.
- #define ASIC_EVT_PVR_RENDERDONE_ISP 0x0001

  ISP render stage completed.
- #define ASIC_EVT_PVR_RENDERDONE_TSP 0x0002

  TSP render stage completed.
- #define ASIC_EVT_PVR_VBLANK_BEGIN 0x0003

  VBLANK begin interrupt.
- #define ASIC_EVT_PVR_VBLANK_END 0x0004

  VBLANK end interrupt.
- #define ASIC_EVT_PVR_HBLANK_BEGIN 0x0005

  HBLANK begin interrupt.
- #define ASIC_EVT_PVR_YUV_DONE 0x0007

  YUV completed.
- #define ASIC_EVT_PVR_OPAQUEDONE 0x0007

  Opaque list completed.
- #define ASIC_EVT_PVR_OPAQUEMODDONE 0x0008

  Opaque modifiers completed.
- #define ASIC_EVT_PVR_TRANSDONE 0x0009

  Transparent list completed.
- #define ASIC_EVT_PVR_TRANSMODDONE 0x000a

  Transparent modifiers completed.

- #define ASIC_EVT_PVR_DMA 0x0013

  PVR DMA complete.
- #define ASIC_EVT_PVR_PTDONE 0x0015

  Punch-thrus completed.
- #define ASIC_EVT_PVR_ISP_OUTOFMEM 0x0200

  ISP out of memory.
- #define ASIC_EVT_PVR_STRIP_HALT 0x0201

  Halt due to strip buffer error.
- #define ASIC_EVT_PVR_PARAM_OUTOFMEM 0x0202

  Param out of memory.
- #define ASIC_EVT_PVR_OPB_OUTOFMEM 0x0203

  OPB went past PVR_TA_OPB_END.
- #define ASIC_EVT_PVR_TA_INPUT_ERR 0x0204

  Vertex input error.
- #define ASIC_EVT_PVR_TA_INPUT_OVERFLOW 0x0205

  Vertex input overflowed queue.
- #define ASIC_EVT_GD_COMMAND 0x0100

  GD-Rom Command Status.
- #define ASIC_EVT_GD_DMA 0x000e

  GD-Rom DMA complete.
- #define ASIC_EVT_GD_DMA_OVERRUN 0x020d

  GD-Rom DMA overrun.
- #define ASIC_EVT_GD_DMA_ILLADDR 0x020c

  GD-Rom DMA illegal address.
- #define ASIC_EVT_MAPLE_DMA 0x000c

  Maple DMA complete.
- #define ASIC_EVT_MAPLE_ERROR 0x000d

  Maple error (?)
- #define ASIC_EVT_SPU_DMA 0x000f

  SPU (G2 channel 0) DMA complete.
- #define ASIC_EVT_SPU_IRQ 0x0101

  SPU interrupt.
- #define ASIC_EVT_G2_DMA0 0x000f

  G2 DMA channel 0 complete.
- #define ASIC_EVT_G2_DMA1 0x0010

  G2 DMA channel 1 complete.
- #define ASIC_EVT_G2_DMA2 0x0011

  G2 DMA channel 2 complete.
- #define ASIC_EVT_G2_DMA3 0x0012

  G2 DMA channel 3 complete.
- #define ASIC_EVT_EXP_8BIT 0x0102

  Modem / Lan Adapter.
- #define ASIC_EVT_EXP_PCI 0x0103

  BBA IRQ.
- #define ASIC_ACK_A 0xa05f6900

  IRQD ACK register.
- #define ASIC_ACK_B 0xa05f6904

IRQB ACK register.

- #define ASIC_ACK_C 0xa05f6908

  IRQ9 ACK register.

- #define ASIC_IRQD_A 0xa05f6910

  IRQD first register.

- #define ASIC_IRQD_B 0xa05f6914

  IRQD second register.

- #define ASIC_IRQD_C 0xa05f6918

  IRQD third register.

- #define ASIC_IRQB_A 0xa05f6920

  IRQB first register.

- #define ASIC_IRQB_B 0xa05f6924

  IRQB second register.

- #define ASIC_IRQB_C 0xa05f6928

  IRQB third register.

- #define ASIC_IRQ9_A 0xa05f6930

  IRQ9 first register.

- #define ASIC_IRQ9_B 0xa05f6934

  IRQ9 second register.

- #define ASIC_IRQ9_C 0xa05f6938

  IRQ9 third register.

- #define ASIC_IRQ9 0

  IRQ level 9.

- #define ASIC_IRQB 1

  IRQ level B (11)

- #define ASIC_IRQD 2

  IRQ level D (13)

- #define ASIC_IRQ_MAX 3

  Don't take irqs from here up.

- #define ASIC_IRQ_DEFAULT ASIC_IRQ9

  Pick an IRQ level for me!

Typedefs

- typedef void(∗ asic_evt_handler) (uint32_t code)

  ASIC event handler type.

Functions

- void asic_evt_set_handler (uint16_t code, asic_evt_handler handler)

  Set or remove an ASIC handler.

- void asic_evt_disable_all (void)

  Disable all ASIC events.

- void asic_evt_disable (uint16_t code, uint8_t irqlevel)

  Disable one ASIC event.

- void asic_evt_enable (uint16_t code, uint8_t irqlevel)

Enable an ASIC event.

- void asic_init (void)

  Init ASIC events.

- void asic_shutdown (void)

  Shutdown ASIC events, disabling all hooks.

9.162.1 Detailed Description

Dreamcast ASIC event handling support.

This file provides definitions of the events that the ASIC (a part of the PVR) in the Dreamcast can trigger as IRQs, and ways to set responders for those events. Pretty much, this covers all IRQs that aren't generated internally in the SH4 (SCIF and the SH4 DMAC can generate their own IRQs, as well as the trapa instruction, and various exceptions – those are not dealt with here).

Author

Megan Potter

9.162.2 Typedef Documentation

asic_evt_handler

```
typedef void(* asic_evt_handler) (uint32_t code)
```

ASIC event handler type.

Any event handlers registered must be of this type. These will be run in an interrupt context, so don't try anything funny.

Parameters

code	The ASIC event code that generated this event.

See also

ASIC event codes

9.162.3 Function Documentation

asic_evt_disable()

```
void asic_evt_disable (
            uint16_t code,
            uint8_t irqlevel )
```

Disable one ASIC event.

This function will disable the hook for a specified code that was registered at the given IRQ level. Generally, you will never have to do this yourself unless you're adding in some new functionality.

Parameters

code	The ASIC event code to unhook (see ASIC event codes).
irqlevel	The IRQ level it was hooked on (see ASIC IRQ levels).

asic_evt_disable_all()

```
void asic_evt_disable_all (
            void  )
```

Disable all ASIC events.

This function will disable hooks for every event that has been hooked. In order to reinstate them, you must individually re-enable them. Not a very good idea to be doing this normally.

asic_evt_enable()

```
void asic_evt_enable (
            uint16_t code,
            uint8_t irqlevel )
```

Enable an ASIC event.

This function will enable the hook for a specified code and register it at the given IRQ level. You should only register each event at a max of one IRQ level (this will not check that for you), and this does not actually set the hook function for the event, you must do that separately with asic_evt_set_handler(). Generally, unless you're adding in new functionality, you'll never have to do this.

Parameters

code	The ASIC event code to hook (see ASIC event codes).
irqlevel	The IRQ level to hook on (see ASIC IRQ levels).

asic_evt_set_handler()

```
void asic_evt_set_handler (
            uint16_t code,
            asic_evt_handler handler )
```

Set or remove an ASIC handler.

This function will register an event handler for a given event code, or if the handler is NULL, unregister any that is currently registered.

Parameters

code	The ASIC event code to hook (see ASIC event codes).
handler	The function to call when the event happens.

asic_init()

```
void asic_init (
            void  )
```

Init ASIC events.

asic_shutdown()

```
void asic_shutdown (
            void  )
```

Shutdown ASIC events, disabling all hooks.

9.163 asic.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/asic.h
00004    Copyright (C) 2001-2002 Megan Potter
00005
00006 */
00007
00008 /** \file   dc/asic.h
00009     \brief  Dreamcast ASIC event handling support.
00010
00011     This file provides definitions of the events that the ASIC (a part of the
00012     PVR) in the Dreamcast can trigger as IRQs, and ways to set responders for
00013     those events. Pretty much, this covers all IRQs that aren't generated
00014     internally in the SH4 (SCIF and the SH4 DMAC can generate their own IRQs,
00015     as well as the trapa instruction, and various exceptions -- those are not
00016     dealt with here).
00017
00018     \author Megan Potter
00019 */
00020
00021 #ifndef __DC_ASIC_H
00022 #define __DC_ASIC_H
00023
00024 #include <sys/cdefs.h>
00025 __BEGIN_DECLS
00026
00027 #include <stdint.h>
00028
00029 /* All event codes are two 8-bit integers; the top integer is the event code
00030    register to look in to check the event (and to acknolwedge it). The
00031    register to check is 0xa05f6900+4*regnum. The bottom integer is the
00032    bit index within that register. */
00033
00034 /** \defgroup asic_events           ASIC event codes
00035     @{
00036 */
00037
00038 /** \defgroup asic_pvr_evts         Event codes for the PVR chip
```

```
00039
00040     These are events that the PVR itself generates that can be hooked.
00041     @{
00042 */
00043
00044 #define ASIC_EVT_PVR_RENDERDONE_VIDEO     0x0000  /**< \brief Video render stage completed */
00045 #define ASIC_EVT_PVR_RENDERDONE_ISP       0x0001  /**< \brief ISP render stage completed */
00046 #define ASIC_EVT_PVR_RENDERDONE_TSP       0x0002  /**< \brief TSP render stage completed */
00047 #define ASIC_EVT_PVR_VBLANK_BEGIN         0x0003  /**< \brief VBLANK begin interrupt */
00048 #define ASIC_EVT_PVR_VBLANK_END           0x0004  /**< \brief VBLANK end interrupt */
00049 #define ASIC_EVT_PVR_HBLANK_BEGIN         0x0005  /**< \brief HBLANK begin interrupt */
00050
00051 #define ASIC_EVT_PVR_YUV_DONE             0x0007  /**< \brief YUV completed */
00052 #define ASIC_EVT_PVR_OPAQUEDONE           0x0007  /**< \brief Opaque list completed */
00053 #define ASIC_EVT_PVR_OPAQUEMODDONE        0x0008  /**< \brief Opaque modifiers completed */
00054 #define ASIC_EVT_PVR_TRANSDONE            0x0009  /**< \brief Transparent list completed */
00055 #define ASIC_EVT_PVR_TRANSMODDONE         0x000a  /**< \brief Transparent modifiers completed */
00056
00057 #define ASIC_EVT_PVR_DMA                  0x0013  /**< \brief PVR DMA complete */
00058 #define ASIC_EVT_PVR_PTDONE               0x0015  /**< \brief Punch-thrus completed */
00059
00060 #define ASIC_EVT_PVR_ISP_OUTOFMEM         0x0200  /**< \brief ISP out of memory */
00061 #define ASIC_EVT_PVR_STRIP_HALT           0x0201  /**< \brief Halt due to strip buffer error */
00062 #define ASIC_EVT_PVR_PARAM_OUTOFMEM       0x0202  /**< \brief Param out of memory */
00063 #define ASIC_EVT_PVR_OPB_OUTOFMEM         0x0203  /**< \brief OPB went past PVR_TA_OPB_END */
00064 #define ASIC_EVT_PVR_TA_INPUT_ERR         0x0204  /**< \brief Vertex input error */
00065 #define ASIC_EVT_PVR_TA_INPUT_OVERFLOW    0x0205  /**< \brief Vertex input overflowed queue */
00066
00067 /** @} */
00068
00069 /** \defgroup asic_gd_evts          Event codes for the GD controller
00070
00071     These are events that the GD-ROM drive generates that can be hooked.
00072     @{
00073 */
00074 #define ASIC_EVT_GD_COMMAND         0x0100  /**< \brief GD-Rom Command Status */
00075 #define ASIC_EVT_GD_DMA             0x000e  /**< \brief GD-Rom DMA complete */
00076 #define ASIC_EVT_GD_DMA_OVERRUN     0x020d  /**< \brief GD-Rom DMA overrun */
00077 #define ASIC_EVT_GD_DMA_ILLADDR     0x020c  /**< \brief GD-Rom DMA illegal address */
00078 /** @} */
00079
00080 /** \defgroup asic_maple_evts       Event codes for the Maple controller
00081
00082     These are events that Maple generates that can be hooked.
00083     @{
00084 */
00085 #define ASIC_EVT_MAPLE_DMA          0x000c  /**< \brief Maple DMA complete */
00086 #define ASIC_EVT_MAPLE_ERROR        0x000d  /**< \brief Maple error (?) */
00087 /** @} */
00088
00089 /** \defgroup asic_spu_evts         Event codes for the SPU
00090
00091     These are events that the SPU (AICA) generates that can be hooked.
00092     @{
00093 */
00094 #define ASIC_EVT_SPU_DMA            0x000f  /**< \brief SPU (G2 channel 0) DMA complete */
00095 #define ASIC_EVT_SPU_IRQ            0x0101  /**< \brief SPU interrupt */
00096 /** @} */
00097
00098 /** \defgroup asic_g2dma_evts       Event codes for G2 bus DMA
00099
00100     These are events that G2 bus DMA generates that can be hooked.
00101     @{
00102 */
00103 #define ASIC_EVT_G2_DMA0            0x000f  /**< \brief G2 DMA channel 0 complete */
00104 #define ASIC_EVT_G2_DMA1            0x0010  /**< \brief G2 DMA channel 1 complete */
00105 #define ASIC_EVT_G2_DMA2            0x0011  /**< \brief G2 DMA channel 2 complete */
00106 #define ASIC_EVT_G2_DMA3            0x0012  /**< \brief G2 DMA channel 3 complete */
00107 /** @} */
00108
00109 /** \defgroup asic_ext_evts         Event codes for the external port
00110
00111     These are events that external devices generate that can be hooked.
00112     @{
00113 */
00114 #define ASIC_EVT_EXP_8BIT           0x0102  /**< \brief Modem / Lan Adapter */
00115 #define ASIC_EVT_EXP_PCI            0x0103  /**< \brief BBA IRQ */
00116 /** @} */
00117 /** @} */
00118
00119 /** \defgroup asic_regs             ASIC registers
```

```
00120
00121     These are the locations in memory where the ASIC registers sit.
00122     @{
00123 */
00124 #define ASIC_ACK_A            0xa05f6900  /**< \brief IRQD ACK register */
00125 #define ASIC_ACK_B            0xa05f6904  /**< \brief IRQB ACK register */
00126 #define ASIC_ACK_C            0xa05f6908  /**< \brief IRQ9 ACK register */
00127
00128 #define ASIC_IRQD_A            0xa05f6910  /**< \brief IRQD first register */
00129 #define ASIC_IRQD_B            0xa05f6914  /**< \brief IRQD second register */
00130 #define ASIC_IRQD_C            0xa05f6918  /**< \brief IRQD third register */
00131 #define ASIC_IRQB_A            0xa05f6920  /**< \brief IRQB first register */
00132 #define ASIC_IRQB_B            0xa05f6924  /**< \brief IRQB second register */
00133 #define ASIC_IRQB_C            0xa05f6928  /**< \brief IRQB third register */
00134 #define ASIC_IRQ9_A            0xa05f6930  /**< \brief IRQ9 first register */
00135 #define ASIC_IRQ9_B            0xa05f6934  /**< \brief IRQ9 second register */
00136 #define ASIC_IRQ9_C            0xa05f6938  /**< \brief IRQ9 third register */
00137 /** @} */
00138
00139 /** \defgroup asic_irq_lv           ASIC IRQ levels
00140
00141     You can pick one at hook time, or don't choose anything and the default will
00142     be used instead.
00143     @{
00144 */
00145 #define ASIC_IRQ9           0  /**< \brief IRQ level 9 */
00146 #define ASIC_IRQB           1  /**< \brief IRQ level B (11) */
00147 #define ASIC_IRQD           2  /**< \brief IRQ level D (13) */
00148
00149 #define ASIC_IRQ_MAX        3  /**< \brief Don't take irqs from here up */
00150 #define ASIC_IRQ_DEFAULT    ASIC_IRQ9  /**< \brief Pick an IRQ level for me! */
00151 /** @} */
00152
00153 /** \brief  ASIC event handler type.
00154
00155     Any event handlers registered must be of this type. These will be run in an
00156     interrupt context, so don't try anything funny.
00157
00158     \param  code            The ASIC event code that generated this event.
00159     \see    asic_events
00160 */
00161 typedef void (*asic_evt_handler)(uint32_t code);
00162
00163 /** \brief  Set or remove an ASIC handler.
00164
00165     This function will register an event handler for a given event code, or if
00166     the handler is NULL, unregister any that is currently registered.
00167
00168     \param  code            The ASIC event code to hook (see \ref asic_events).
00169     \param  handler         The function to call when the event happens.
00170
00171 */
00172 void asic_evt_set_handler(uint16_t code, asic_evt_handler handler);
00173
00174 /** \brief  Disable all ASIC events.
00175
00176     This function will disable hooks for every event that has been hooked. In
00177     order to reinstate them, you must individually re-enable them. Not a very
00178     good idea to be doing this normally.
00179 */
00180 void asic_evt_disable_all(void);
00181
00182 /** \brief  Disable one ASIC event.
00183
00184     This function will disable the hook for a specified code that was registered
00185     at the given IRQ level. Generally, you will never have to do this yourself
00186     unless you're adding in some new functionality.
00187
00188     \param  code            The ASIC event code to unhook (see
00189                             \ref asic_events).
00190     \param  irqlevel        The IRQ level it was hooked on (see
00191                             \ref asic_irq_lv).
00192 */
00193 void asic_evt_disable(uint16_t code, uint8_t irqlevel);
00194
00195 /** \brief  Enable an ASIC event.
00196
00197     This function will enable the hook for a specified code and register it at
00198     the given IRQ level. You should only register each event at a max of one
00199     IRQ level (this will not check that for you), and this does not actually set
00200     the hook function for the event, you must do that separately with
```

```
00201     asic_evt_set_handler(). Generally, unless you're adding in new
00202     functionality, you'll never have to do this.
00203
00204     \param  code            The ASIC event code to hook (see \ref asic_events).
00205     \param  irqlevel        The IRQ level to hook on (see \ref asic_irq_lv).
00206  */
00207 void asic_evt_enable(uint16_t code, uint8_t irqlevel);
00208
00209 /** \brief  Init ASIC events. */
00210 void asic_init(void);
00211
00212 /** \brief  Shutdown ASIC events, disabling all hooks. */
00213 void asic_shutdown(void);
00214
00215 __END_DECLS
00216
00217 #endif  /* __DC_ASIC_H */
```

9.164 kernel/arch/dreamcast/include/dc/biosfont.h File Reference

BIOS font drawing functions.

```
#include <kos/cdefs.h>
#include <arch/types.h>
```

Macros

- #define BFONT_THIN_WIDTH 12

  Width of Thin Font (ISO8859_1, half-JP)
- #define BFONT_WIDE_WIDTH BFONT_THIN_WIDTH ∗ 2

  Width of Wide Font (full-JP)
- #define BFONT_HEIGHT 24

  Height of All Fonts.
- #define JISX_0208_ROW_SIZE 94
- #define BFONT_NARROW_START 0

  Start of Narrow Characters in Font Block.
- #define BFONT_OVERBAR BFONT_NARROW_START
- #define BFONT_ISO_8859_1_33_126 BFONT_NARROW_START+(1∗BFONT_THIN_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_YEN BFONT_NARROW_START+(95∗BFONT_THIN_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_ISO_8859_1_160_255 BFONT_NARROW_START+(96∗BFONT_THIN_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_WIDE_START (288∗BFONT_THIN_WIDTH∗BFONT_HEIGHT/8)

  Start of Wide Characters in Font Block.
- #define BFONT_JISX_0208_ROW1 BFONT_WIDE_START

  Start of JISX-0208 Rows 1-7 in Font Block.
- #define BFONT_JISX_0208_ROW16 BFONT_WIDE_START+(658∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)

  Start of JISX-0208 Row 16-47 (Start of Level 1) in Font Block.
- #define BFONT_JISX_0208_ROW48 BFONT_JISX_0208_ROW16+((32∗JISX_0208_ROW_SIZE)∗BFONT_WIDE_WIDTH∗BFON

  JISX-0208 Row 48-84 (Start of Level 2) in Font Block.
- #define BFONT_DREAMCAST_SPECIFIC BFONT_WIDE_START+(7056∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)

  Start of DC Specific Characters in Font Block.
- #define BFONT_CIRCLECOPYRIGHT BFONT_DREAMCAST_SPECIFIC+(0∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_CIRCLER BFONT_DREAMCAST_SPECIFIC+(1∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)

- #define BFONT_TRADEMARK BFONT_DREAMCAST_SPECIFIC+(2∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_UPARROW BFONT_DREAMCAST_SPECIFIC+(3∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_DOWNARROW BFONT_DREAMCAST_SPECIFIC+(4∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_LEFTARROW BFONT_DREAMCAST_SPECIFIC+(5∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_RIGHTARROW BFONT_DREAMCAST_SPECIFIC+(6∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_UPRIGHTARROW BFONT_DREAMCAST_SPECIFIC+(7∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_DOWNRIGHTARROW BFONT_DREAMCAST_SPECIFIC+(8∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_DOWNLEFTARROW BFONT_DREAMCAST_SPECIFIC+(9∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_UPLEFTARROW BFONT_DREAMCAST_SPECIFIC+(10∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_ABUTTON BFONT_DREAMCAST_SPECIFIC+(11∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_BBUTTON BFONT_DREAMCAST_SPECIFIC+(12∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_CBUTTON BFONT_DREAMCAST_SPECIFIC+(13∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_DBUTTON BFONT_DREAMCAST_SPECIFIC+(14∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_XBUTTON BFONT_DREAMCAST_SPECIFIC+(15∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_YBUTTON BFONT_DREAMCAST_SPECIFIC+(16∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_ZBUTTON BFONT_DREAMCAST_SPECIFIC+(17∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_LTRIGGER BFONT_DREAMCAST_SPECIFIC+(18∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_RTRIGGER BFONT_DREAMCAST_SPECIFIC+(19∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_STARTBUTTON BFONT_DREAMCAST_SPECIFIC+(20∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_VMUICON BFONT_DREAMCAST_SPECIFIC+(21∗BFONT_WIDE_WIDTH∗BFONT_HEIGHT/8)
- #define BFONT_ICON_DIMEN 32

  Dimension of vmu icons.
- #define BFONT_VMU_DREAMCAST_SPECIFIC BFONT_DREAMCAST_SPECIFIC+(22∗BFONT_WIDE_WIDTH∗BFONT_HEIGH
- #define BFONT_ICON_INVALID_VMU 0x00
- #define BFONT_ICON_HOURGLASS_ONE 0x01
- #define BFONT_ICON_HOURGLASS_TWO 0x02
- #define BFONT_ICON_HOURGLASS_THREE 0x03
- #define BFONT_ICON_HOURGLASS_FOUR 0x04
- #define BFONT_ICON_VMUICON 0x05
- #define BFONT_ICON_EARTH 0x06
- #define BFONT_ICON_SATURN 0x07
- #define BFONT_ICON_QUARTER_MOON 0x08
- #define BFONT_ICON_LAUGHING_FACE 0x09
- #define BFONT_ICON_SMILING_FACE 0x0A
- #define BFONT_ICON_CASUAL_FACE 0x0B
- #define BFONT_ICON_ANGRY_FACE 0x0C
- #define BFONT_ICON_COW 0x0D
- #define BFONT_ICON_HORSE 0x0E
- #define BFONT_ICON_RABBIT 0x0F
- #define BFONT_ICON_CAT 0x10
- #define BFONT_ICON_CHICK 0x11
- #define BFONT_ICON_LION 0x12
- #define BFONT_ICON_MONKEY 0x13
- #define BFONT_ICON_PANDA 0x14
- #define BFONT_ICON_BEAR 0x15
- #define BFONT_ICON_PIG 0x16
- #define BFONT_ICON_DOG 0x17
- #define BFONT_ICON_FISH 0x18
- #define BFONT_ICON_OCTOPUS 0x19
- #define BFONT_ICON_SQUID 0x1A

- #define BFONT_ICON_WHALE 0x1B
- #define BFONT_ICON_CRAB 0x1C
- #define BFONT_ICON_BUTTERFLY 0x1D
- #define BFONT_ICON_LADYBUG 0x1E
- #define BFONT_ICON_ANGLER_FISH 0x1F
- #define BFONT_ICON_PENGUIN 0x20
- #define BFONT_ICON_CHERRIES 0x21
- #define BFONT_ICON_TULIP 0x22
- #define BFONT_ICON_LEAF 0x23
- #define BFONT_ICON_SAKURA 0x24
- #define BFONT_ICON_APPLE 0x25
- #define BFONT_ICON_ICECREAM 0x26
- #define BFONT_ICON_CACTUS 0x27
- #define BFONT_ICON_PIANO 0x28
- #define BFONT_ICON_GUITAR 0x29
- #define BFONT_ICON_EIGHTH_NOTE 0x2A
- #define BFONT_ICON_TREBLE_CLEF 0x2B
- #define BFONT_ICON_BOAT 0x2C
- #define BFONT_ICON_CAR 0x2D
- #define BFONT_ICON_HELMET 0x2E
- #define BFONT_ICON_MOTORCYCLE 0x2F
- #define BFONT_ICON_VAN 0x30
- #define BFONT_ICON_TRUCK 0x31
- #define BFONT_ICON_CLOCK 0x32
- #define BFONT_ICON_TELEPHONE 0x33
- #define BFONT_ICON_PENCIL 0x34
- #define BFONT_ICON_CUP 0x35
- #define BFONT_ICON_SILVERWARE 0x36
- #define BFONT_ICON_HOUSE 0x37
- #define BFONT_ICON_BELL 0x38
- #define BFONT_ICON_CROWN 0x39
- #define BFONT_ICON_SOCK 0x3A
- #define BFONT_ICON_CAKE 0x3B
- #define BFONT_ICON_KEY 0x3C
- #define BFONT_ICON_BOOK 0x3D
- #define BFONT_ICON_BASEBALL 0x3E
- #define BFONT_ICON_SOCCER 0x3F
- #define BFONT_ICON_BULB 0x40
- #define BFONT_ICON_TEDDY_BEAR 0x41
- #define BFONT_ICON_BOW_TIE 0x42
- #define BFONT_ICON_BOW_ARROW 0x43
- #define BFONT_ICON_SNOWMAN 0x44
- #define BFONT_ICON_LIGHTNING 0x45
- #define BFONT_ICON_SUN 0x46
- #define BFONT_ICON_CLOUD 0x47
- #define BFONT_ICON_UMBRELLA 0x48
- #define BFONT_ICON_ONE_STAR 0x49
- #define BFONT_ICON_TWO_STARS 0x4A
- #define BFONT_ICON_THREE_STARS 0x4B
- #define BFONT_ICON_FOUR_STARS 0x4C
- #define BFONT_ICON_HEART 0x4D

- #define BFONT_ICON_DIAMOND 0x4E
- #define BFONT_ICON_SPADE 0x4F
- #define BFONT_ICON_CLUB 0x50
- #define BFONT_ICON_JACK 0x51
- #define BFONT_ICON_QUEEN 0x52
- #define BFONT_ICON_KING 0x53
- #define BFONT_ICON_JOKER 0x54
- #define BFONT_ICON_ISLAND 0x55
- #define BFONT_ICON_0 0x56
- #define BFONT_ICON_1 0x57
- #define BFONT_ICON_2 0x58
- #define BFONT_ICON_3 0x59
- #define BFONT_ICON_4 0x5A
- #define BFONT_ICON_5 0x5B
- #define BFONT_ICON_6 0x5C
- #define BFONT_ICON_7 0x5D
- #define BFONT_ICON_8 0x5E
- #define BFONT_ICON_9 0x5F
- #define BFONT_ICON_A 0x60
- #define BFONT_ICON_B 0x61
- #define BFONT_ICON_C 0x62
- #define BFONT_ICON_D 0x63
- #define BFONT_ICON_E 0x64
- #define BFONT_ICON_F 0x65
- #define BFONT_ICON_G 0x66
- #define BFONT_ICON_H 0x67
- #define BFONT_ICON_I 0x68
- #define BFONT_ICON_J 0x69
- #define BFONT_ICON_K 0x6A
- #define BFONT_ICON_L 0x6B
- #define BFONT_ICON_M 0x6C
- #define BFONT_ICON_N 0x6D
- #define BFONT_ICON_O 0x6E
- #define BFONT_ICON_P 0x6F
- #define BFONT_ICON_Q 0x70
- #define BFONT_ICON_R 0x71
- #define BFONT_ICON_S 0x72
- #define BFONT_ICON_T 0x73
- #define BFONT_ICON_U 0x74
- #define BFONT_ICON_V 0x75
- #define BFONT_ICON_W 0x76
- #define BFONT_ICON_X 0x77
- #define BFONT_ICON_Y 0x78
- #define BFONT_ICON_Z 0x79
- #define BFONT_ICON_CHECKER_BOARD 0x7A
- #define BFONT_ICON_GRID 0x7B
- #define BFONT_ICON_LIGHT_GRAY 0x7C
- #define BFONT_ICON_DIAG_GRID 0x7D
- #define BFONT_ICON_PACMAN_GRID 0x7E
- #define BFONT_ICON_DARK_GRAY 0x7F
- #define BFONT_ICON_EMBROIDERY 0x80

- #define BFONT_CODE_ISO8859_1 0

  ISO-8859-1 (western) charset.
- #define BFONT_CODE_EUC 1

  EUC-JP charset.
- #define BFONT_CODE_SJIS 2

  Shift-JIS charset.
- #define BFONT_CODE_RAW 3

  Raw indexing to the BFONT.

Functions

- uint32 bfont_set_foreground_color (uint32 c)

  Set the font foreground color.
- uint32 bfont_set_background_color (uint32 c)

  Set the font background color.
- int bfont_set_32bit_mode (int on) __depr("Please use the bpp function of the the bfont_draw_ex functions")

  Set the font to draw 32-bit color.
- void bfont_set_encoding (uint8 enc)

  Set the font encoding.
- uint8 * bfont_find_char (uint32 ch)

  Find an ISO-8859-1 character in the font.
- uint8 * bfont_find_char_jp (uint32 ch)

  Find an full-width Japanese character in the font.
- uint8 * bfont_find_char_jp_half (uint32 ch)

  Find an half-width Japanese character in the font.
- unsigned char bfont_draw_ex (uint8 *buffer, uint32 bufwidth, uint32 fg, uint32 bg, uint8 bpp, uint8 opaque, uint32 c, uint8 wide, uint8 iskana)

  Draw a single character of any sort to the buffer.
- unsigned char bfont_draw (void *buffer, uint32 bufwidth, uint8 opaque, uint32 c)

  Draw a single character to a buffer.
- unsigned char bfont_draw_thin (void *buffer, uint32 bufwidth, uint8 opaque, uint32 c, uint8 iskana)

  Draw a single thin character to a buffer.
- unsigned char bfont_draw_wide (void *buffer, uint32 bufwidth, uint8 opaque, uint32 c)

  Draw a single wide character to a buffer.
- void bfont_draw_str_ex (void *b, uint32 width, uint32 fg, uint32 bg, uint8 bpp, uint8 opaque, const char *str)

  Draw a full string to any sort of buffer.
- void bfont_draw_str (void *b, uint32 width, uint8 opaque, const char *str)

  Draw a full string to a buffer.
- uint8 * bfont_find_icon (uint8 icon)

  Find a VMU icon.

9.164.1 Detailed Description

BIOS font drawing functions.

This file provides support for utilizing the font built into the Dreamcast's BIOS. These functions allow access to both the western character set and Japanese characters.

Author

Megan Potter

Kazuaki Matsumoto

Donald Haase

9.164.2 Macro Definition Documentation

BFONT_CODE_EUC

```
#define BFONT_CODE_EUC 1
```

EUC-JP charset.

BFONT_CODE_ISO8859_1

```
#define BFONT_CODE_ISO8859_1 0
```

ISO-8859-1 (western) charset.

BFONT_CODE_RAW

```
#define BFONT_CODE_RAW 3
```

Raw indexing to the BFONT.

BFONT_CODE_SJIS

```
#define BFONT_CODE_SJIS 2
```

Shift-JIS charset.

JISX_0208_ROW_SIZE

```
#define JISX_0208_ROW_SIZE 94
```

9.164.3 Function Documentation

bfont_draw()

```
unsigned char bfont_draw (
            void * buffer,
            uint32 bufwidth,
            uint8 opaque,
            uint32 c )
```

Draw a single character to a buffer.

This function draws a single character in the set encoding to the given buffer. Calling this is equivalent to calling bfont_draw_thin() with 0 for the final parameter.

Parameters

buffer	The buffer to draw to (at least 12 x 24 pixels)
bufwidth	The width of the buffer in pixels
opaque	If non-zero, overwrite blank areas with black, otherwise do not change them from what they are
c	The character to draw

Returns

Amount of width covered in bytes.

bfont_draw_ex()

```
unsigned char bfont_draw_ex (
            uint8 * buffer,
            uint32 bufwidth,
            uint32 fg,
            uint32 bg,
            uint8 bpp,
            uint8 opaque,
            uint32 c,
            uint8 wide,
            uint8 iskana )
```

Draw a single character of any sort to the buffer.

This function draws a single character in the set encoding to the given buffer. This function sits under draw, draw_↩ thin, and draw_wide, while exposing the colors and bitdepths desired. This will allow the writing of bfont characters to palletted textures.

Parameters

buffer	The buffer to draw to.
bufwidth	The width of the buffer in pixels.

Parameters

fg	The foreground color to use.
bg	The background color to use.
bpp	The number of bits per pixel in the buffer.
opaque	If non-zero, overwrite background areas with black, otherwise do not change them from what they are.
c	The character to draw.
wide	Draw a wide character.
iskana	Draw a half-width kana character.

Returns

Amount of width covered in bytes.

bfont_draw_str()

```
void bfont_draw_str (
            void * b,
            uint32 width,
            uint8 opaque,
            const char * str )
```

Draw a full string to a buffer.

This function draws a NUL-terminated string in the set encoding to the given buffer. This will automatically handle mixed half and full-width characters if the encoding is set to one of the Japanese encodings. Draws pre-set 16-bit colors.

Parameters

b	The buffer to draw to.
width	The width of the buffer in pixels.
opaque	If one, overwrite blank areas with bfont_bgcolor, otherwise do not change them from what they are.
str	The string to draw.

bfont_draw_str_ex()

```
void bfont_draw_str_ex (
            void * b,
            uint32 width,
            uint32 fg,
            uint32 bg,
            uint8 bpp,
            uint8 opaque,
            const char * str )
```

Draw a full string to any sort of buffer.

This function draws a NUL-terminated string in the set encoding to the given buffer. This will automatically handle mixed half and full-width characters if the encoding is set to one of the Japanese encodings. Colors and bitdepth can be set.

Parameters

b	The buffer to draw to.
width	The width of the buffer in pixels.
fg	The foreground color to use.
bg	The background color to use.
bpp	The number of bits per pixel in the buffer.
opaque	If non-zero, overwrite background areas with black, otherwise do not change them from what they are.
str	The string to draw.

bfont_draw_thin()

```
unsigned char bfont_draw_thin (
            void * buffer,
            uint32 bufwidth,
            uint8 opaque,
            uint32 c,
            uint8 iskana )
```

Draw a single thin character to a buffer.

This function draws a single character in the set encoding to the given buffer. This only works with ISO-8859-1 characters and half-width kana.

Parameters

buffer	The buffer to draw to (at least 12 x 24 pixels)
bufwidth	The width of the buffer in pixels
opaque	If non-zero, overwrite blank areas with black, otherwise do not change them from what they are
c	The character to draw
iskana	Set to 1 if the character is a kana, 0 if ISO-8859-1

Returns

Amount of width covered in bytes.

bfont_draw_wide()

```
unsigned char bfont_draw_wide (
            void * buffer,
            uint32 bufwidth,
```

```
            uint8 opaque,
            uint32 c )
```

Draw a single wide character to a buffer.

This function draws a single character in the set encoding to the given buffer. This only works with full-width kana and kanji.

Parameters

buffer	The buffer to draw to (at least 24 x 24 pixels)
bufwidth	The width of the buffer in pixels
opaque	If non-zero, overwrite blank areas with black, otherwise do not change them from what they are
c	The character to draw

Returns

Amount of width covered in bytes.

bfont_find_char()

```
uint8 * bfont_find_char (
            uint32 ch )
```

Find an ISO-8859-1 character in the font.

This function retrieves a pointer to the font data for the specified character in the font, if its available. Generally, you will not have to use this function, use one of the bfont_draw_* functions instead.

Parameters

ch	The character to look up

Returns

A pointer to the raw character data

bfont_find_char_jp()

```
uint8 * bfont_find_char_jp (
            uint32 ch )
```

Find an full-width Japanese character in the font.

This function retrieves a pointer to the font data for the specified character in the font, if its available. Generally, you will not have to use this function, use one of the bfont_draw_* functions instead.

This function deals with full-width kana and kanji.

Parameters

ch	The character to look up

Returns

A pointer to the raw character data

bfont_find_char_jp_half()

```
uint8 * bfont_find_char_jp_half (
            uint32 ch )
```

Find an half-width Japanese character in the font.

This function retrieves a pointer to the font data for the specified character in the font, if its available. Generally, you will not have to use this function, use one of the bfont_draw_* functions instead.

This function deals with half-width kana only.

Parameters

ch	The character to look up

Returns

A pointer to the raw character data

bfont_find_icon()

```
uint8 * bfont_find_icon (
            uint8 icon )
```

Find a VMU icon.

This function retrieves a pointer to the icon data for the specified VMU icon in the bios, if its available. The icon data is flipped both vertically and horizontally. Each vmu icon has dimensions 32x32 pixels and is 128 bytes long.

Parameters

icon	The icon to look up. Use BFONT_ICON_* values starting with BFONT_ICON_INVALID_VMU.

Returns

A pointer to the raw icon data or NULL if icon value is incorrect.

bfont_set_32bit_mode()

```
int bfont_set_32bit_mode (
            int on )
```

Set the font to draw 32-bit color.

This function changes whether the font draws colors as 32-bit or 16-bit. The default is to use 16-bit.

Parameters

on	Set to 0 to use 16-bit color, 32-bit otherwise.

Returns

The old state (1 = 32-bit, 0 = 16-bit).

bfont_set_background_color()

```
uint32 bfont_set_background_color (
            uint32 c )
```

Set the font background color.

This function selects the background color to draw when a pixel is drawn in the font. This color is only used for pixels not covered by the font when you have selected to have the font be opaque.

Parameters

c	The color to use.

Returns

The old background color.

bfont_set_encoding()

```
void bfont_set_encoding (
            uint8 enc )
```

Set the font encoding.

This function selects the font encoding that is used for the font. This allows you to select between the various character sets available.

Parameters

enc	The character encoding in use

See also

BFONT_CODE_ISO8859_1

BFONT_CODE_EUC

BFONT_CODE_SJIS

BFONT_CODE_RAW

bfont_set_foreground_color()

```
uint32 bfont_set_foreground_color (
             uint32 c )
```

Set the font foreground color.

This function selects the foreground color to draw when a pixel is opaque in the font. The value passed in for the color should be in whatever pixel format that you intend to use for the image produced.

Parameters

c	The color to use.

Returns

The old foreground color.

9.165 biosfont.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/biosfont.h
00004    (c)2000-2001 Megan Potter
00005    Japanese Functions (c)2002 Kazuaki Matsumoto
00006    (c)2017 Donald Haase
00007
00008 */
00009
00010 /** \file   dc/biosfont.h
00011     \brief  BIOS font drawing functions.
00012
00013     This file provides support for utilizing the font built into the Dreamcast's
00014     BIOS. These functions allow access to both the western character set and
00015     Japanese characters.
00016
00017     \author Megan Potter
00018     \author Kazuaki Matsumoto
00019     \author Donald Haase
00020 */
00021
```

```
00022 #ifndef __DC_BIOSFONT_H
00023 #define __DC_BIOSFONT_H
00024
00025 #include <kos/cdefs.h>
00026 __BEGIN_DECLS
00027
00028 #include <arch/types.h>
00029
00030 /** \defgroup bfont_size  Dimensions of the Bios Font
00031     @{
00032 */
00033 #define BFONT_THIN_WIDTH                        12   /**< \brief Width of Thin Font (ISO8859_1, half-JP)
      */
00034 #define BFONT_WIDE_WIDTH      BFONT_THIN_WIDTH * 2   /**< \brief Width of Wide Font (full-JP) */
00035 #define BFONT_HEIGHT                            24   /**< \brief Height of All Fonts */
00036 /** @} */
00037
00038 #define JISX_0208_ROW_SIZE  94
00039 /** \defgroup bfont_indicies Structure of the Bios Font
00040     @{
00041 */
00042 #define BFONT_NARROW_START          0   /**< \brief Start of Narrow Characters in Font Block */
00043 #define BFONT_OVERBAR               BFONT_NARROW_START
00044 #define BFONT_ISO_8859_1_33_126     BFONT_NARROW_START+( 1*BFONT_THIN_WIDTH*BFONT_HEIGHT/8)
00045 #define BFONT_YEN                   BFONT_NARROW_START+(95*BFONT_THIN_WIDTH*BFONT_HEIGHT/8)
00046 #define BFONT_ISO_8859_1_160_255    BFONT_NARROW_START+(96*BFONT_THIN_WIDTH*BFONT_HEIGHT/8)
00047
00048 /* JISX-0208 Rows 1-7 and 16-84 are encoded between BFONT_WIDE_START and BFONT_DREAMCAST_SPECIFIC.
00049     Only the box-drawing characters (row 8) are missing. */
00050 #define BFONT_WIDE_START            (288*BFONT_THIN_WIDTH*BFONT_HEIGHT/8)   /**< \brief Start of Wide
      Characters in Font Block */
00051 #define BFONT_JISX_0208_ROW1        BFONT_WIDE_START    /**< \brief Start of JISX-0208 Rows 1-7 in Font
      Block */
00052 #define BFONT_JISX_0208_ROW16       BFONT_WIDE_START+(658*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)  /**< \brief
      Start of JISX-0208 Row 16-47 (Start of Level 1) in Font Block */
00053 #define BFONT_JISX_0208_ROW48
      BFONT_JISX_0208_ROW16+((32*JISX_0208_ROW_SIZE)*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8) /**< \brief JISX-0208 Row
      48-84 (Start of Level 2) in Font Block */
00054
00055
00056 #define BFONT_DREAMCAST_SPECIFIC    BFONT_WIDE_START+(7056*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8) /**< \brief
      Start of DC Specific Characters in Font Block */
00057 #define BFONT_CIRCLECOPYRIGHT       BFONT_DREAMCAST_SPECIFIC+(0*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00058 #define BFONT_CIRCLER               BFONT_DREAMCAST_SPECIFIC+(1*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00059 #define BFONT_TRADEMARK             BFONT_DREAMCAST_SPECIFIC+(2*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00060 #define BFONT_UPARROW               BFONT_DREAMCAST_SPECIFIC+(3*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00061 #define BFONT_DOWNARROW             BFONT_DREAMCAST_SPECIFIC+(4*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00062 #define BFONT_LEFTARROW             BFONT_DREAMCAST_SPECIFIC+(5*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00063 #define BFONT_RIGHTARROW            BFONT_DREAMCAST_SPECIFIC+(6*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00064 #define BFONT_UPRIGHTARROW          BFONT_DREAMCAST_SPECIFIC+(7*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00065 #define BFONT_DOWNRIGHTARROW        BFONT_DREAMCAST_SPECIFIC+(8*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00066 #define BFONT_DOWNLEFTARROW         BFONT_DREAMCAST_SPECIFIC+(9*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00067 #define BFONT_UPLEFTARROW           BFONT_DREAMCAST_SPECIFIC+(10*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00068 #define BFONT_ABUTTON               BFONT_DREAMCAST_SPECIFIC+(11*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00069 #define BFONT_BBUTTON               BFONT_DREAMCAST_SPECIFIC+(12*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00070 #define BFONT_CBUTTON               BFONT_DREAMCAST_SPECIFIC+(13*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00071 #define BFONT_DBUTTON               BFONT_DREAMCAST_SPECIFIC+(14*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00072 #define BFONT_XBUTTON               BFONT_DREAMCAST_SPECIFIC+(15*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00073 #define BFONT_YBUTTON               BFONT_DREAMCAST_SPECIFIC+(16*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00074 #define BFONT_ZBUTTON               BFONT_DREAMCAST_SPECIFIC+(17*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00075 #define BFONT_LTRIGGER              BFONT_DREAMCAST_SPECIFIC+(18*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00076 #define BFONT_RTRIGGER              BFONT_DREAMCAST_SPECIFIC+(19*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00077 #define BFONT_STARTBUTTON           BFONT_DREAMCAST_SPECIFIC+(20*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00078 #define BFONT_VMUICON               BFONT_DREAMCAST_SPECIFIC+(21*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00079
00080 #define BFONT_ICON_DIMEN                32    /**< \brief Dimension of vmu icons */
00081 #define BFONT_VMU_DREAMCAST_SPECIFIC    BFONT_DREAMCAST_SPECIFIC+(22*BFONT_WIDE_WIDTH*BFONT_HEIGHT/8)
00082 /** @} */
00083
00084 /** \defgroup vmu_icons Builtin VMU Icons
00085     \ingroup  bfont_indicies
00086
00087     Builtin VMU volume user icons. The Dreamcast's
00088     BIOS allows the user to set these when formatting the VMU.
00089
00090     @{
00091 */
00092 #define BFONT_ICON_INVALID_VMU          0x00
00093 #define BFONT_ICON_HOURGLASS_ONE        0x01
00094 #define BFONT_ICON_HOURGLASS_TWO        0x02
00095 #define BFONT_ICON_HOURGLASS_THREE      0x03
```

```
00096 #define BFONT_ICON_HOURGLASS_FOUR       0x04
00097 #define BFONT_ICON_VMUICON              0x05
00098 #define BFONT_ICON_EARTH                0x06
00099 #define BFONT_ICON_SATURN               0x07
00100 #define BFONT_ICON_QUARTER_MOON         0x08
00101 #define BFONT_ICON_LAUGHING_FACE        0x09
00102 #define BFONT_ICON_SMILING_FACE         0x0A
00103 #define BFONT_ICON_CASUAL_FACE          0x0B
00104 #define BFONT_ICON_ANGRY_FACE           0x0C
00105 #define BFONT_ICON_COW                  0x0D
00106 #define BFONT_ICON_HORSE                0x0E
00107 #define BFONT_ICON_RABBIT               0x0F
00108 #define BFONT_ICON_CAT                  0x10
00109 #define BFONT_ICON_CHICK                0x11
00110 #define BFONT_ICON_LION                 0x12
00111 #define BFONT_ICON_MONKEY               0x13
00112 #define BFONT_ICON_PANDA                0x14
00113 #define BFONT_ICON_BEAR                 0x15
00114 #define BFONT_ICON_PIG                  0x16
00115 #define BFONT_ICON_DOG                  0x17
00116 #define BFONT_ICON_FISH                 0x18
00117 #define BFONT_ICON_OCTOPUS              0x19
00118 #define BFONT_ICON_SQUID                0x1A
00119 #define BFONT_ICON_WHALE                0x1B
00120 #define BFONT_ICON_CRAB                 0x1C
00121 #define BFONT_ICON_BUTTERFLY            0x1D
00122 #define BFONT_ICON_LADYBUG              0x1E
00123 #define BFONT_ICON_ANGLER_FISH          0x1F
00124 #define BFONT_ICON_PENGUIN              0x20
00125 #define BFONT_ICON_CHERRIES             0x21
00126 #define BFONT_ICON_TULIP                0x22
00127 #define BFONT_ICON_LEAF                 0x23
00128 #define BFONT_ICON_SAKURA               0x24
00129 #define BFONT_ICON_APPLE                0x25
00130 #define BFONT_ICON_ICECREAM             0x26
00131 #define BFONT_ICON_CACTUS               0x27
00132 #define BFONT_ICON_PIANO                0x28
00133 #define BFONT_ICON_GUITAR               0x29
00134 #define BFONT_ICON_EIGHTH_NOTE          0x2A
00135 #define BFONT_ICON_TREBLE_CLEF          0x2B
00136 #define BFONT_ICON_BOAT                 0x2C
00137 #define BFONT_ICON_CAR                  0x2D
00138 #define BFONT_ICON_HELMET               0x2E
00139 #define BFONT_ICON_MOTORCYCLE           0x2F
00140 #define BFONT_ICON_VAN                  0x30
00141 #define BFONT_ICON_TRUCK                0x31
00142 #define BFONT_ICON_CLOCK                0x32
00143 #define BFONT_ICON_TELEPHONE            0x33
00144 #define BFONT_ICON_PENCIL               0x34
00145 #define BFONT_ICON_CUP                  0x35
00146 #define BFONT_ICON_SILVERWARE           0x36
00147 #define BFONT_ICON_HOUSE                0x37
00148 #define BFONT_ICON_BELL                 0x38
00149 #define BFONT_ICON_CROWN                0x39
00150 #define BFONT_ICON_SOCK                 0x3A
00151 #define BFONT_ICON_CAKE                 0x3B
00152 #define BFONT_ICON_KEY                  0x3C
00153 #define BFONT_ICON_BOOK                 0x3D
00154 #define BFONT_ICON_BASEBALL             0x3E
00155 #define BFONT_ICON_SOCCER               0x3F
00156 #define BFONT_ICON_BULB                 0x40
00157 #define BFONT_ICON_TEDDY_BEAR           0x41
00158 #define BFONT_ICON_BOW_TIE              0x42
00159 #define BFONT_ICON_BOW_ARROW            0x43
00160 #define BFONT_ICON_SNOWMAN              0x44
00161 #define BFONT_ICON_LIGHTNING            0x45
00162 #define BFONT_ICON_SUN                  0x46
00163 #define BFONT_ICON_CLOUD                0x47
00164 #define BFONT_ICON_UMBRELLA             0x48
00165 #define BFONT_ICON_ONE_STAR             0x49
00166 #define BFONT_ICON_TWO_STARS            0x4A
00167 #define BFONT_ICON_THREE_STARS          0x4B
00168 #define BFONT_ICON_FOUR_STARS           0x4C
00169 #define BFONT_ICON_HEART                0x4D
00170 #define BFONT_ICON_DIAMOND              0x4E
00171 #define BFONT_ICON_SPADE                0x4F
00172 #define BFONT_ICON_CLUB                 0x50
00173 #define BFONT_ICON_JACK                 0x51
00174 #define BFONT_ICON_QUEEN                0x52
00175 #define BFONT_ICON_KING                 0x53
00176 #define BFONT_ICON_JOKER                0x54
```

```
00177 #define BFONT_ICON_ISLAND               0x55
00178 #define BFONT_ICON_0                    0x56
00179 #define BFONT_ICON_1                    0x57
00180 #define BFONT_ICON_2                    0x58
00181 #define BFONT_ICON_3                    0x59
00182 #define BFONT_ICON_4                    0x5A
00183 #define BFONT_ICON_5                    0x5B
00184 #define BFONT_ICON_6                    0x5C
00185 #define BFONT_ICON_7                    0x5D
00186 #define BFONT_ICON_8                    0x5E
00187 #define BFONT_ICON_9                    0x5F
00188 #define BFONT_ICON_A                    0x60
00189 #define BFONT_ICON_B                    0x61
00190 #define BFONT_ICON_C                    0x62
00191 #define BFONT_ICON_D                    0x63
00192 #define BFONT_ICON_E                    0x64
00193 #define BFONT_ICON_F                    0x65
00194 #define BFONT_ICON_G                    0x66
00195 #define BFONT_ICON_H                    0x67
00196 #define BFONT_ICON_I                    0x68
00197 #define BFONT_ICON_J                    0x69
00198 #define BFONT_ICON_K                    0x6A
00199 #define BFONT_ICON_L                    0x6B
00200 #define BFONT_ICON_M                    0x6C
00201 #define BFONT_ICON_N                    0x6D
00202 #define BFONT_ICON_O                    0x6E
00203 #define BFONT_ICON_P                    0x6F
00204 #define BFONT_ICON_Q                    0x70
00205 #define BFONT_ICON_R                    0x71
00206 #define BFONT_ICON_S                    0x72
00207 #define BFONT_ICON_T                    0x73
00208 #define BFONT_ICON_U                    0x74
00209 #define BFONT_ICON_V                    0x75
00210 #define BFONT_ICON_W                    0x76
00211 #define BFONT_ICON_X                    0x77
00212 #define BFONT_ICON_Y                    0x78
00213 #define BFONT_ICON_Z                    0x79
00214 #define BFONT_ICON_CHECKER_BOARD        0x7A
00215 #define BFONT_ICON_GRID                 0x7B
00216 #define BFONT_ICON_LIGHT_GRAY           0x7C
00217 #define BFONT_ICON_DIAG_GRID            0x7D
00218 #define BFONT_ICON_PACMAN_GRID          0x7E
00219 #define BFONT_ICON_DARK_GRAY            0x7F
00220 #define BFONT_ICON_EMBROIDERY           0x80
00221 /** @} */
00222
00223 /** \brief  Set the font foreground color.
00224
00225     This function selects the foreground color to draw when a pixel is opaque in
00226     the font. The value passed in for the color should be in whatever pixel
00227     format that you intend to use for the image produced.
00228
00229     \param  c               The color to use.
00230     \return                 The old foreground color.
00231 */
00232 uint32 bfont_set_foreground_color(uint32 c);
00233
00234 /** \brief  Set the font background color.
00235
00236     This function selects the background color to draw when a pixel is drawn in
00237     the font. This color is only used for pixels not covered by the font when
00238     you have selected to have the font be opaque.
00239
00240     \param  c               The color to use.
00241     \return                 The old background color.
00242 */
00243 uint32 bfont_set_background_color(uint32 c);
00244
00245 /** \brief  Set the font to draw 32-bit color.
00246
00247     This function changes whether the font draws colors as 32-bit or 16-bit. The
00248     default is to use 16-bit.
00249
00250     \param  on              Set to 0 to use 16-bit color, 32-bit otherwise.
00251     \return                 The old state (1 = 32-bit, 0 = 16-bit).
00252 */
00253 int bfont_set_32bit_mode(int on)
00254     __depr("Please use the bpp function of the the bfont_draw_ex functions");
00255
00256 /* Constants for the function below */
00257 #define BFONT_CODE_ISO8859_1    0   /**< \brief ISO-8859-1 (western) charset */
```

```
00258 #define BFONT_CODE_EUC          1   /**< \brief EUC-JP charset */
00259 #define BFONT_CODE_SJIS         2   /**< \brief Shift-JIS charset */
00260 #define BFONT_CODE_RAW          3   /**< \brief Raw indexing to the BFONT */
00261
00262 /** \brief  Set the font encoding.
00263
00264     This function selects the font encoding that is used for the font. This
00265     allows you to select between the various character sets available.
00266
00267     \param  enc             The character encoding in use
00268     \see    BFONT_CODE_ISO8859_1
00269     \see    BFONT_CODE_EUC
00270     \see    BFONT_CODE_SJIS
00271     \see    BFONT_CODE_RAW
00272 */
00273 void bfont_set_encoding(uint8 enc);
00274
00275 /** \brief  Find an ISO-8859-1 character in the font.
00276
00277     This function retrieves a pointer to the font data for the specified
00278     character in the font, if its available. Generally, you will not have to
00279     use this function, use one of the bfont_draw_* functions instead.
00280
00281     \param  ch              The character to look up
00282     \return                 A pointer to the raw character data
00283 */
00284 uint8 *bfont_find_char(uint32 ch);
00285
00286 /** \brief  Find an full-width Japanese character in the font.
00287
00288     This function retrieves a pointer to the font data for the specified
00289     character in the font, if its available. Generally, you will not have to
00290     use this function, use one of the bfont_draw_* functions instead.
00291
00292     This function deals with full-width kana and kanji.
00293
00294     \param  ch              The character to look up
00295     \return                 A pointer to the raw character data
00296 */
00297 uint8 *bfont_find_char_jp(uint32 ch);
00298
00299 /** \brief  Find an half-width Japanese character in the font.
00300
00301     This function retrieves a pointer to the font data for the specified
00302     character in the font, if its available. Generally, you will not have to
00303     use this function, use one of the bfont_draw_* functions instead.
00304
00305     This function deals with half-width kana only.
00306
00307     \param  ch              The character to look up
00308     \return                 A pointer to the raw character data
00309 */
00310 uint8 *bfont_find_char_jp_half(uint32 ch);
00311
00312 /** \brief Draw a single character of any sort to the buffer.
00313
00314     This function draws a single character in the set encoding to the given
00315     buffer. This function sits under draw, draw_thin, and draw_wide, while
00316     exposing the colors and bitdepths desired. This will allow the writing
00317     of bfont characters to palletted textures.
00318
00319     \param buffer       The buffer to draw to.
00320     \param bufwidth     The width of the buffer in pixels.
00321     \param fg           The foreground color to use.
00322     \param bg           The background color to use.
00323     \param bpp          The number of bits per pixel in the buffer.
00324     \param opaque       If non-zero, overwrite background areas with black,
00325                             otherwise do not change them from what they are.
00326     \param c            The character to draw.
00327     \param wide         Draw a wide character.
00328     \param iskana       Draw a half-width kana character.
00329     \return             Amount of width covered in bytes.
00330 */
00331 unsigned char bfont_draw_ex(uint8 *buffer, uint32 bufwidth, uint32 fg,
00332                             uint32 bg, uint8 bpp, uint8 opaque, uint32 c,
00333                             uint8 wide, uint8 iskana);
00334
00335 /** \brief  Draw a single character to a buffer.
00336
00337     This function draws a single character in the set encoding to the given
00338     buffer. Calling this is equivalent to calling bfont_draw_thin() with 0 for
```

```
00339     the final parameter.
00340
00341     \param  buffer          The buffer to draw to (at least 12 x 24 pixels)
00342     \param  bufwidth        The width of the buffer in pixels
00343     \param  opaque          If non-zero, overwrite blank areas with black,
00344                             otherwise do not change them from what they are
00345     \param  c               The character to draw
00346     \return                 Amount of width covered in bytes.
00347 */
00348 unsigned char bfont_draw(void *buffer, uint32 bufwidth, uint8 opaque, uint32 c);
00349
00350 /** \brief  Draw a single thin character to a buffer.
00351
00352     This function draws a single character in the set encoding to the given
00353     buffer. This only works with ISO-8859-1 characters and half-width kana.
00354
00355     \param  buffer          The buffer to draw to (at least 12 x 24 pixels)
00356     \param  bufwidth        The width of the buffer in pixels
00357     \param  opaque          If non-zero, overwrite blank areas with black,
00358                             otherwise do not change them from what they are
00359     \param  c               The character to draw
00360     \param  iskana          Set to 1 if the character is a kana, 0 if ISO-8859-1
00361     \return                 Amount of width covered in bytes.
00362 */
00363 unsigned char bfont_draw_thin(void *buffer, uint32 bufwidth, uint8 opaque,
00364                               uint32 c, uint8 iskana);
00365
00366 /** \brief  Draw a single wide character to a buffer.
00367
00368     This function draws a single character in the set encoding to the given
00369     buffer. This only works with full-width kana and kanji.
00370
00371     \param  buffer          The buffer to draw to (at least 24 x 24 pixels)
00372     \param  bufwidth        The width of the buffer in pixels
00373     \param  opaque          If non-zero, overwrite blank areas with black,
00374                             otherwise do not change them from what they are
00375     \param  c               The character to draw
00376     \return                 Amount of width covered in bytes.
00377 */
00378 unsigned char bfont_draw_wide(void *buffer, uint32 bufwidth, uint8 opaque,
00379                               uint32 c);
00380
00381 /** \brief  Draw a full string to any sort of buffer.
00382
00383     This function draws a NUL-terminated string in the set encoding to the given
00384     buffer. This will automatically handle mixed half and full-width characters
00385     if the encoding is set to one of the Japanese encodings. Colors and bitdepth
00386     can be set.
00387
00388     \param b                The buffer to draw to.
00389     \param width            The width of the buffer in pixels.
00390     \param fg               The foreground color to use.
00391     \param bg               The background color to use.
00392     \param bpp              The number of bits per pixel in the buffer.
00393     \param opaque           If non-zero, overwrite background areas with black,
00394                             otherwise do not change them from what they are.
00395     \param str              The string to draw.
00396
00397 */
00398 void bfont_draw_str_ex(void *b, uint32 width, uint32 fg, uint32 bg, uint8 bpp,
00399                        uint8 opaque, const char *str);
00400
00401 /** \brief  Draw a full string to a buffer.
00402
00403     This function draws a NUL-terminated string in the set encoding to the given
00404     buffer. This will automatically handle mixed half and full-width characters
00405     if the encoding is set to one of the Japanese encodings. Draws pre-set
00406     16-bit colors.
00407
00408     \param  b               The buffer to draw to.
00409     \param  width           The width of the buffer in pixels.
00410     \param  opaque          If one, overwrite blank areas with bfont_bgcolor,
00411                             otherwise do not change them from what they are.
00412     \param  str             The string to draw.
00413 */
00414 void bfont_draw_str(void *b, uint32 width, uint8 opaque, const char *str);
00415
00416 /** \brief  Find a VMU icon.
00417
00418     This function retrieves a pointer to the icon data for the specified VMU
00419     icon in the bios, if its available. The icon data is flipped both vertically
```

```
00420     and horizontally. Each vmu icon has dimensions 32x32 pixels and is 128 bytes
00421     long.
00422
00423     \param  icon            The icon to look up. Use BFONT_ICON_* values
00424                             starting with BFONT_ICON_INVALID_VMU.
00425     \return                 A pointer to the raw icon data or NULL if icon value
00426                             is incorrect.
00427 */
00428 uint8 *bfont_find_icon(uint8 icon);
00429
00430 __END_DECLS
00431
00432 #endif  /* __DC_BIOSFONT_H */
```

9.166 kernel/arch/dreamcast/include/dc/cdrom.h File Reference

CD access to the GD-ROM drive.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Data Structures

- struct CDROM_TOC

  TOC structure returned by the BIOS.

Macros

- #define CMD_CHECK_LICENSE 2

  Check license.
- #define CMD_REQ_SPI_CMD 4

  Request to Sega Packet Interface.
- #define CMD_PIOREAD 16

  Read via PIO.
- #define CMD_DMAREAD 17

  Read via DMA.
- #define CMD_GETTOC 18

  Read TOC.
- #define CMD_GETTOC2 19

  Read TOC.
- #define CMD_PLAY 20

  Play track.
- #define CMD_PLAY2 21

  Play sectors.
- #define CMD_PAUSE 22

  Pause playback.
- #define CMD_RELEASE 23

  Resume from pause.
- #define CMD_INIT 24

Initialize the drive.

- #define CMD_DMA_ABORT 25

  Abort DMA transfer.
- #define CMD_OPEN_TRAY 26

  Open CD tray (on DevBox?)
- #define CMD_SEEK 27

  Seek to a new position.
- #define CMD_DMAREAD_STREAM 28

  Stream DMA until end/abort.
- #define CMD_NOP 29

  No operation.
- #define CMD_REQ_MODE 30

  Request mode.
- #define CMD_SET_MODE 31

  Setup mode.
- #define CMD_SCAN_CD 32

  Scan CD.
- #define CMD_STOP 33

  Stop the disc from spinning.
- #define CMD_GETSCD 34

  Get subcode data.
- #define CMD_GETSES 35

  Get session.
- #define CMD_REQ_STAT 36

  Request stat.
- #define CMD_PIOREAD_STREAM 37

  Stream PIO until end/abort.
- #define CMD_DMAREAD_STREAM_EX 38

  Stream DMA transfer.
- #define CMD_PIOREAD_STREAM_EX 39

  Stream PIO transfer.
- #define CMD_GET_VERS 40

  Get syscall driver version.
- #define CMD_MAX 47

  Max of GD syscall commands.
- #define ERR_OK 0

  No error.
- #define ERR_NO_DISC 1

  No disc in drive.
- #define ERR_DISC_CHG 2

  Disc changed, but not reinitted yet.
- #define ERR_SYS 3

  System error.
- #define ERR_ABORTED 4

  Command aborted.
- #define ERR_NO_ACTIVE 5

  System inactive?

- #define ERR_TIMEOUT 6

  Aborted due to timeout.
- #define FAILED -1

  Command failed.
- #define NO_ACTIVE 0

  System inactive?
- #define PROCESSING 1

  Processing command.
- #define COMPLETED 2

  Command completed successfully.
- #define STREAMING 3

  Stream type command is in progress.
- #define BUSY 4

  GD syscalls is busy.
- #define ATA_STAT_INTERNAL 0x00
- #define ATA_STAT_IRQ 0x01
- #define ATA_STAT_DRQ_0 0x02
- #define ATA_STAT_DRQ_1 0x03
- #define ATA_STAT_BUSY 0x04
- #define CDDA_TRACKS 1

  Play by track number.
- #define CDDA_SECTORS 2

  Play by sector number.
- #define CDROM_READ_WHOLE_SECTOR 0x1000

  Read the whole sector.
- #define CDROM_READ_DATA_AREA 0x2000

  Read the data area.
- #define CD_SUB_Q_ALL 0

  Read all Subcode Data.
- #define CD_SUB_Q_CHANNEL 1

  Read Q Channel Subcode Data.
- #define CD_SUB_MEDIA_CATALOG 2

  Read the Media Catalog Subcode Data.
- #define CD_SUB_TRACK_ISRC 3

  Read the ISRC Subcode Data.
- #define CD_SUB_RESERVED 4

  Reserved.
- #define CD_SUB_AUDIO_STATUS_INVALID 0x00
- #define CD_SUB_AUDIO_STATUS_PLAYING 0x11
- #define CD_SUB_AUDIO_STATUS_PAUSED 0x12
- #define CD_SUB_AUDIO_STATUS_ENDED 0x13
- #define CD_SUB_AUDIO_STATUS_ERROR 0x14
- #define CD_SUB_AUDIO_STATUS_NO_INFO 0x15
- #define CDROM_READ_PIO 0

  Read sector(s) in PIO mode.
- #define CDROM_READ_DMA 1

  Read sector(s) in DMA mode.
- #define CD_STATUS_READ_FAIL -1

Can't read status.

- #define CD_STATUS_BUSY 0

  Drive is busy.
- #define CD_STATUS_PAUSED 1

  Disc is paused.
- #define CD_STATUS_STANDBY 2

  Drive is in standby.
- #define CD_STATUS_PLAYING 3

  Drive is currently playing.
- #define CD_STATUS_SEEKING 4

  Drive is currently seeking.
- #define CD_STATUS_SCANNING 5

  Drive is scanning.
- #define CD_STATUS_OPEN 6

  Disc tray is open.
- #define CD_STATUS_NO_DISC 7

  No disc inserted.
- #define CD_STATUS_RETRY 8

  Retry is needed.
- #define CD_STATUS_ERROR 9

  System error.
- #define CD_STATUS_FATAL 12

  Need reset syscalls.
- #define CD_CDDA 0x00

  Audio CD (Red book) or no disc.
- #define CD_CDROM 0x10

  CD-ROM or CD-R (Yellow book)
- #define CD_CDROM_XA 0x20

  CD-ROM XA (Yellow book extension)
- #define CD_CDI 0x30

  CD-i (Green book)
- #define CD_GDROM 0x80

  GD-ROM.
- #define CD_FAIL 0xf0

  Need reset syscalls.
- #define TOC_LBA(n) ((n) & 0x00ffffff)

  Get the FAD address of a TOC entry.
- #define TOC_ADR(n) (((n) & 0x0f000000) >> 24)

  Get the address of a TOC entry.
- #define TOC_CTRL(n) (((n) & 0xf0000000) >> 28)

  Get the control data of a TOC entry.
- #define TOC_TRACK(n) (((n) & 0x00ff0000) >> 16)

  Get the track number of a TOC entry.

Functions

- int cdrom_set_sector_size (int size)

  Set the sector size for read sectors.
- int cdrom_exec_cmd (int cmd, void *param)

  Execute a CD-ROM command.
- int cdrom_exec_cmd_timed (int cmd, void *param, int timeout)

  Execute a CD-ROM command with timeout.
- int cdrom_get_status (int *status, int *disc_type)

  Get the status of the GD-ROM drive.
- int cdrom_change_dataype (int sector_part, int cdxa, int sector_size) __depr("Use cdrom_change_datatype instead.")

  Change the datatype of disc.
- int cdrom_change_datatype (int sector_part, int cdxa, int sector_size)

  Change the datatype of disc.
- int cdrom_reinit (void)

  Re-initialize the GD-ROM drive.
- int cdrom_reinit_ex (int sector_part, int cdxa, int sector_size)

  Re-initialize the GD-ROM drive with custom parameters.
- int cdrom_read_toc (CDROM_TOC *toc_buffer, int session)

  Read the table of contents from the disc.
- int cdrom_read_sectors_ex (void *buffer, int sector, int cnt, int mode)

  Read one or more sector from a CD-ROM.
- int cdrom_read_sectors (void *buffer, int sector, int cnt)

  Read one or more sector from a CD-ROM in PIO mode.
- int cdrom_get_subcode (void *buffer, int buflen, int which)

  Read subcode data from the most recently read sectors.
- uint32 cdrom_locate_data_track (CDROM_TOC *toc)

  Locate the sector of the data track.
- int cdrom_cdda_play (uint32 start, uint32 end, uint32 loops, int mode)

  Play CDDA audio tracks or sectors.
- int cdrom_cdda_pause (void)

  Pause CDDA audio playback.
- int cdrom_cdda_resume (void)

  Resume CDDA audio playback after a pause.
- int cdrom_spin_down (void)

  Spin down the CD.
- int cdrom_init (void)

  Initialize the GD-ROM for reading CDs.
- void cdrom_shutdown (void)

  Shutdown the CD reading system.

9.166.1 Detailed Description

CD access to the GD-ROM drive.

This file contains the interface to the Dreamcast's GD-ROM drive. It is simply called cdrom.h and cdrom.c because, by design, you cannot directly use this code to read the high-density area of GD-ROMs. This is the way it always has been, and always will be.

The way things are set up, as long as you're using fs_iso9660 to access the CD, it will automatically detect and react to disc changes for you.

This file only facilitates reading raw sectors and doing other fairly low- level things with CDs. If you're looking for higher-level stuff, like normal file reading, consult with the stuff for the fs and for fs_iso9660.

Author

Megan Potter

Ruslan Rostovtsev

See also

kos/fs.h

dc/fs_iso9660.h

9.166.2 Function Documentation

cdrom_cdda_pause()

```
int cdrom_cdda_pause (
            void  )
```

Pause CDDA audio playback.

Returns

CD-ROM command responses

cdrom_cdda_play()

```
int cdrom_cdda_play (
            uint32 start,
            uint32 end,
            uint32 loops,
            int mode )
```

Play CDDA audio tracks or sectors.

This function starts playback of CDDA audio.

Parameters

start	The track or sector to start playback from.
end	The track or sector to end playback at.
loops	The number of times to repeat (max of 15).
mode	The mode to play (see CDDA read modes).

Returns

CD-ROM command responses

cdrom_cdda_resume()

```
int cdrom_cdda_resume (
            void  )
```

Resume CDDA audio playback after a pause.

Returns

CD-ROM command responses

cdrom_change_datatype()

```
int cdrom_change_datatype (
            int sector_part,
            int cdxa,
            int sector_size )
```

Change the datatype of disc.

This function will take in all parameters to pass to the change_datatype syscall. This allows these parameters to be modified without a reinit. Each parameter allows -1 as a default, which is tied to the former static values provided by cdrom_reinit and cdrom_set_sector_size.

Parameters

sector_part	How much of each sector to return.
cdxa	What CDXA mode to read as (if applicable).
sector_size	What sector size to read (eg. - 2048, 2532).

Returns

CD-ROM command responses

See also

CD-ROM Read Sector Part

cdrom_change_datype()

```
int cdrom_change_datype (
            int sector_part,
            int cdxa,
            int sector_size )
```

Change the datatype of disc.

Note

This function is formally deprecated. It should not be used in any future code, and may be removed in the future. You should instead use cdrom_change_datatype.

cdrom_exec_cmd()

```
int cdrom_exec_cmd (
            int cmd,
            void * param )
```

Execute a CD-ROM command.

This function executes the specified command using the BIOS syscall for executing GD-ROM commands.

Parameters

cmd	The command number to execute.
param	Data to pass to the syscall.

Returns

CD-ROM command responses

cdrom_exec_cmd_timed()

```
int cdrom_exec_cmd_timed (
            int cmd,
            void * param,
            int timeout )
```

Execute a CD-ROM command with timeout.

This function executes the specified command using the BIOS syscall for executing GD-ROM commands with timeout.

Parameters

cmd	The command number to execute.
param	Data to pass to the syscall.
timeout	Timeout in milliseconds.

Returns

CD-ROM command responses

cdrom_get_status()

```
int cdrom_get_status (
            int * status,
            int * disc_type )
```

Get the status of the GD-ROM drive.

Parameters

status	Space to return the drive's status.
disc_type	Space to return the type of disc in the drive.

Returns

CD-ROM command responses

See also

CD-ROM status values

CD-ROM drive disc types

cdrom_get_subcode()

```
int cdrom_get_subcode (
            void * buffer,
            int buflen,
            int which )
```

Read subcode data from the most recently read sectors.

After reading sectors, this can pull subcode data regarding the sectors read. If reading all subcode data with CD_↩ SUB_CURRENT_POSITION, this needs to be performed one sector at a time.

Parameters

buffer	Space to store the read subcode data.
buflen	Amount of data to be read.
which	Which subcode type do you wish to get.

Returns

CD-ROM command responses

See also

CD-ROM Read Subcode Type

cdrom_init()

```
int cdrom_init (
             void  )
```

Initialize the GD-ROM for reading CDs.

This initializes the CD-ROM reading system, reactivating the drive and handling initial setup of the disc.

Return values

0	On success.
-1	Already initted, shutdown before initting again.

cdrom_locate_data_track()

```
uint32 cdrom_locate_data_track (
             CDROM_TOC * toc )
```

Locate the sector of the data track.

This function will search the toc for the last entry that has a CTRL value of 4, and return its FAD address.

Parameters

toc	The TOC to search through.

Returns

The FAD of the track, or 0 if none is found.

cdrom_read_sectors()

```
int cdrom_read_sectors (
            void * buffer,
            int sector,
            int cnt )
```

Read one or more sector from a CD-ROM in PIO mode.

Default version of cdrom_read_sectors_ex, which forces PIO mode.

Parameters

buffer	Space to store the read sectors.
sector	The sector to start reading from.
cnt	The number of sectors to read.

Returns

CD-ROM command responses

See also

cdrom_read_sectors_ex

cdrom_read_sectors_ex()

```
int cdrom_read_sectors_ex (
            void * buffer,
            int sector,
            int cnt,
            int mode )
```

Read one or more sector from a CD-ROM.

This function reads the specified number of sectors from the disc, starting where requested. This will respect the size of the sectors set with cdrom_change_datatype(). The buffer must have enough space to store the specified number of sectors.

Parameters

buffer	Space to store the read sectors.
sector	The sector to start reading from.
cnt	The number of sectors to read.
mode	DMA or PIO

Returns

CD-ROM command responses

See also

CD-ROM Read Sector Mode

cdrom_read_toc()

```
int cdrom_read_toc (
            CDROM_TOC * toc_buffer,
            int session )
```

Read the table of contents from the disc.

This function reads the TOC from the specified session of the disc.

Parameters

toc_buffer	Space to store the returned TOC in.
session	The session of the disc to read.

Returns

CD-ROM command responses

cdrom_reinit()

```
int cdrom_reinit (
            void  )
```

Re-initialize the GD-ROM drive.

This function is for reinitializing the GD-ROM drive after a disc change to its default settings. Calls cdrom_reinit(-1,-1,-1)

Returns

CD-ROM command responses

See also

cdrom_reinit_ex

cdrom_reinit_ex()

```
int cdrom_reinit_ex (
            int sector_part,
            int cdxa,
            int sector_size )
```

Re-initialize the GD-ROM drive with custom parameters.

At the end of each cdrom_reinit(), cdrom_change_datatype is called. This passes in the requested values to that function after reinitialization, as opposed to defaults.

Parameters

sector_part	How much of each sector to return.
cdxa	What CDXA mode to read as (if applicable).
sector_size	What sector size to read (eg. - 2048, 2532).

Returns

CD-ROM command responses

See also

CD-ROM Read Sector Part

cdrom_change_datatype

cdrom_set_sector_size()

```
int cdrom_set_sector_size (
            int size )
```

Set the sector size for read sectors.

This function sets the sector size that the cdrom_read_sectors() function will return. Be sure to set this to the correct value for the type of sectors you're trying to read. Common values are 2048 (for reading CD-ROM sectors) or 2352 (for reading raw sectors).

Parameters

size	The size of the sector data.

Returns

CD-ROM command responses

cdrom_shutdown()

```
void cdrom_shutdown (
            void  )
```

Shutdown the CD reading system.

cdrom_spin_down()

```
int cdrom_spin_down (
            void  )
```

Spin down the CD.

This stops the disc in the drive from spinning until it is accessed again.

Returns

CD-ROM command responses

9.167 cdrom.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/cdrom.h
00004    Copyright (C) 2000-2001 Megan Potter
00005    Copyright (C) 2014 Donald Haase
00006    Copyright (C) 2023 Ruslan Rostovtsev
00007 */
00008
00009 #ifndef __DC_CDROM_H
00010 #define __DC_CDROM_H
00011
00012 #include <sys/cdefs.h>
00013 __BEGIN_DECLS
00014
00015 #include <arch/types.h>
00016
00017 /** \file   dc/cdrom.h
00018     \brief  CD access to the GD-ROM drive.
00019
00020     This file contains the interface to the Dreamcast's GD-ROM drive. It is
00021     simply called cdrom.h and cdrom.c because, by design, you cannot directly
00022     use this code to read the high-density area of GD-ROMs. This is the way it
00023     always has been, and always will be.
00024
00025     The way things are set up, as long as you're using fs_iso9660 to access the
00026     CD, it will automatically detect and react to disc changes for you.
00027
00028     This file only facilitates reading raw sectors and doing other fairly low-
00029     level things with CDs. If you're looking for higher-level stuff, like
00030     normal file reading, consult with the stuff for the fs and for fs_iso9660.
00031
00032     \author Megan Potter
00033     \author Ruslan Rostovtsev
00034     \see    kos/fs.h
00035     \see    dc/fs_iso9660.h
00036 */
00037
00038 /** \defgroup cd_cmd_codes          CD-ROM syscall command codes
00039
00040     These are the syscall command codes used to actually do stuff with the
00041     GD-ROM drive. These were originally provided by maiwe.
```

```
00042
00043     @{
00044 */
00045 #define CMD_CHECK_LICENSE       2  /**< \brief Check license */
00046 #define CMD_REQ_SPI_CMD         4  /**< \brief Request to Sega Packet Interface */
00047 #define CMD_PIOREAD            16  /**< \brief Read via PIO */
00048 #define CMD_DMAREAD            17  /**< \brief Read via DMA */
00049 #define CMD_GETTOC             18  /**< \brief Read TOC */
00050 #define CMD_GETTOC2            19  /**< \brief Read TOC */
00051 #define CMD_PLAY               20  /**< \brief Play track */
00052 #define CMD_PLAY2              21  /**< \brief Play sectors */
00053 #define CMD_PAUSE              22  /**< \brief Pause playback */
00054 #define CMD_RELEASE            23  /**< \brief Resume from pause */
00055 #define CMD_INIT               24  /**< \brief Initialize the drive */
00056 #define CMD_DMA_ABORT          25  /**< \brief Abort DMA transfer */
00057 #define CMD_OPEN_TRAY          26  /**< \brief Open CD tray (on DevBox?) */
00058 #define CMD_SEEK               27  /**< \brief Seek to a new position */
00059 #define CMD_DMAREAD_STREAM     28  /**< \brief Stream DMA until end/abort */
00060 #define CMD_NOP                29  /**< \brief No operation */
00061 #define CMD_REQ_MODE           30  /**< \brief Request mode */
00062 #define CMD_SET_MODE           31  /**< \brief Setup mode */
00063 #define CMD_SCAN_CD            32  /**< \brief Scan CD */
00064 #define CMD_STOP               33  /**< \brief Stop the disc from spinning */
00065 #define CMD_GETSCD             34  /**< \brief Get subcode data */
00066 #define CMD_GETSES             35  /**< \brief Get session */
00067 #define CMD_REQ_STAT           36  /**< \brief Request stat */
00068 #define CMD_PIOREAD_STREAM     37  /**< \brief Stream PIO until end/abort */
00069 #define CMD_DMAREAD_STREAM_EX  38  /**< \brief Stream DMA transfer */
00070 #define CMD_PIOREAD_STREAM_EX  39  /**< \brief Stream PIO transfer */
00071 #define CMD_GET_VERS           40  /**< \brief Get syscall driver version */
00072 #define CMD_MAX                47  /**< \brief Max of GD syscall commands */
00073 /** @} */
00074
00075 /** \defgroup cd_cmd_response       CD-ROM command responses
00076
00077     These are the values that the various functions can return as error codes.
00078     @{
00079 */
00080 #define ERR_OK          0   /**< \brief No error */
00081 #define ERR_NO_DISC     1   /**< \brief No disc in drive */
00082 #define ERR_DISC_CHG    2   /**< \brief Disc changed, but not reinitted yet */
00083 #define ERR_SYS         3   /**< \brief System error */
00084 #define ERR_ABORTED     4   /**< \brief Command aborted */
00085 #define ERR_NO_ACTIVE   5   /**< \brief System inactive? */
00086 #define ERR_TIMEOUT     6   /**< \brief Aborted due to timeout */
00087 /** @} */
00088
00089 /** \defgroup cd_cmd_status         CD-ROM Command Status responses
00090
00091     These are the raw values the status syscall returns.
00092     @{
00093 */
00094 #define FAILED      -1  /**< \brief Command failed */
00095 #define NO_ACTIVE   0   /**< \brief System inactive? */
00096 #define PROCESSING  1   /**< \brief Processing command */
00097 #define COMPLETED   2   /**< \brief Command completed successfully */
00098 #define STREAMING   3   /**< \brief Stream type command is in progress */
00099 #define BUSY        4   /**< \brief GD syscalls is busy */
00100 /** @} */
00101
00102 /** \defgroup cd_cmd_ata_status       CD-ROM ATA status
00103     @{
00104 */
00105 #define ATA_STAT_INTERNAL   0x00
00106 #define ATA_STAT_IRQ        0x01
00107 #define ATA_STAT_DRQ_0      0x02
00108 #define ATA_STAT_DRQ_1      0x03
00109 #define ATA_STAT_BUSY       0x04
00110 /** @} */
00111
00112 /** \defgroup cdda_read_modes       CDDA read modes
00113
00114     Valid values to pass to the cdrom_cdda_play() function for the mode
00115     parameter.
00116     @{
00117 */
00118 #define CDDA_TRACKS     1   /**< \brief Play by track number */
00119 #define CDDA_SECTORS    2   /**< \brief Play by sector number */
00120 /** @} */
00121
00122 /** \defgroup cd_read_sector_part    CD-ROM Read Sector Part
```

```
00123
00124     Parts of the a CD-ROM sector to read. These are possible values for the
00125     third parameter word sent with the change data type syscall.
00126     @{
00127 */
00128 #define CDROM_READ_WHOLE_SECTOR 0x1000    /**< \brief Read the whole sector */
00129 #define CDROM_READ_DATA_AREA    0x2000    /**< \brief Read the data area */
00130 /** @} */
00131
00132 /** \defgroup cd_read_subcode_type    CD-ROM Read Subcode Type
00133
00134     Types of data available to read from the sector subcode. These are
00135     possible values for the first parameter sent to the GETSCD syscall.
00136     @{
00137 */
00138 #define CD_SUB_Q_ALL            0    /**< \brief Read all Subcode Data */
00139 #define CD_SUB_Q_CHANNEL        1    /**< \brief Read Q Channel Subcode Data */
00140 #define CD_SUB_MEDIA_CATALOG    2    /**< \brief Read the Media Catalog
00141                                           Subcode Data */
00142 #define CD_SUB_TRACK_ISRC       3    /**< \brief Read the ISRC Subcode Data */
00143 #define CD_SUB_RESERVED         4    /**< \brief Reserved */
00144 /** @} */
00145
00146 /** \defgroup cd_subcode_audio    CD-ROM Subcode audio status
00147
00148     Information about CDDA playback from GETSCD syscall.
00149     @{
00150 */
00151 #define CD_SUB_AUDIO_STATUS_INVALID    0x00
00152 #define CD_SUB_AUDIO_STATUS_PLAYING    0x11
00153 #define CD_SUB_AUDIO_STATUS_PAUSED     0x12
00154 #define CD_SUB_AUDIO_STATUS_ENDED      0x13
00155 #define CD_SUB_AUDIO_STATUS_ERROR      0x14
00156 #define CD_SUB_AUDIO_STATUS_NO_INFO    0x15
00157 /** @} */
00158
00159 /** \defgroup cd_read_sector_mode    CD-ROM Read Sector Mode
00160
00161     How to read the sectors of a CD, via PIO or DMA. 4th parameter of
00162     cdrom_read_sectors_ex.
00163     @{
00164 */
00165 #define CDROM_READ_PIO 0    /**< \brief Read sector(s) in PIO mode */
00166 #define CDROM_READ_DMA 1    /**< \brief Read sector(s) in DMA mode */
00167 /** @} */
00168
00169 /** \defgroup cd_status_values      CD-ROM status values
00170
00171     These are the values that can be returned as the status parameter from the
00172     cdrom_get_status() function.
00173     @{
00174 */
00175 #define CD_STATUS_READ_FAIL -1  /**< \brief Can't read status */
00176 #define CD_STATUS_BUSY      0   /**< \brief Drive is busy */
00177 #define CD_STATUS_PAUSED    1   /**< \brief Disc is paused */
00178 #define CD_STATUS_STANDBY   2   /**< \brief Drive is in standby */
00179 #define CD_STATUS_PLAYING   3   /**< \brief Drive is currently playing */
00180 #define CD_STATUS_SEEKING   4   /**< \brief Drive is currently seeking */
00181 #define CD_STATUS_SCANNING  5   /**< \brief Drive is scanning */
00182 #define CD_STATUS_OPEN      6   /**< \brief Disc tray is open */
00183 #define CD_STATUS_NO_DISC   7   /**< \brief No disc inserted */
00184 #define CD_STATUS_RETRY     8   /**< \brief Retry is needed */
00185 #define CD_STATUS_ERROR     9   /**< \brief System error */
00186 #define CD_STATUS_FATAL     12  /**< \brief Need reset syscalls */
00187 /** @} */
00188
00189 /** \defgroup cd_disc_types         CD-ROM drive disc types
00190
00191     These are the values that can be returned as the disc_type parameter from
00192     the cdrom_get_status() function.
00193     @{
00194 */
00195 #define CD_CDDA     0x00    /**< \brief Audio CD (Red book) or no disc */
00196 #define CD_CDROM    0x10    /**< \brief CD-ROM or CD-R (Yellow book) */
00197 #define CD_CDROM_XA 0x20    /**< \brief CD-ROM XA (Yellow book extension) */
00198 #define CD_CDI      0x30    /**< \brief CD-i (Green book) */
00199 #define CD_GDROM    0x80    /**< \brief GD-ROM */
00200 #define CD_FAIL     0xf0    /**< \brief Need reset syscalls */
00201 /** @} */
00202
00203 /** \brief  TOC structure returned by the BIOS.
```

```
00204
00205     This is the structure that the CMD_GETTOC2 syscall command will return for
00206     the TOC. Note the data is in FAD, not LBA/LSN.
00207
00208     \headerfile dc/cdrom.h
00209 */
00210 typedef struct {
00211     uint32  entry[99];          /**< \brief TOC space for 99 tracks */
00212     uint32  first;              /**< \brief Point A0 information (1st track) */
00213     uint32  last;               /**< \brief Point A1 information (last track) */
00214     uint32  leadout_sector;     /**< \brief Point A2 information (leadout) */
00215 } CDROM_TOC;
00216
00217 /** \defgroup cd_toc_access         CD-ROM TOC access macros
00218     @{
00219 */
00220 /** \brief  Get the FAD address of a TOC entry.
00221     \param  n               The actual entry from the TOC to look at.
00222     \return                 The FAD of the entry.
00223 */
00224 #define TOC_LBA(n) ((n) & 0x00ffffff)
00225
00226 /** \brief  Get the address of a TOC entry.
00227     \param  n               The entry from the TOC to look at.
00228     \return                 The entry's address.
00229 */
00230 #define TOC_ADR(n) ( ((n) & 0x0f000000) >> 24 )
00231
00232 /** \brief  Get the control data of a TOC entry.
00233     \param  n               The entry from the TOC to look at.
00234     \return                 The entry's control value.
00235 */
00236 #define TOC_CTRL(n) ( ((n) & 0xf0000000) >> 28 )
00237
00238 /** \brief  Get the track number of a TOC entry.
00239     \param  n               The entry from the TOC to look at.
00240     \return                 The entry's track.
00241 */
00242 #define TOC_TRACK(n) ( ((n) & 0x00ff0000) >> 16 )
00243 /** @} */
00244
00245 /** \brief  Set the sector size for read sectors.
00246
00247     This function sets the sector size that the cdrom_read_sectors() function
00248     will return. Be sure to set this to the correct value for the type of
00249     sectors you're trying to read. Common values are 2048 (for reading CD-ROM
00250     sectors) or 2352 (for reading raw sectors).
00251
00252     \param  size            The size of the sector data.
00253
00254     \return                 \ref cd_cmd_response
00255 */
00256 int cdrom_set_sector_size(int size);
00257
00258 /** \brief  Execute a CD-ROM command.
00259
00260     This function executes the specified command using the BIOS syscall for
00261     executing GD-ROM commands.
00262
00263     \param  cmd             The command number to execute.
00264     \param  param           Data to pass to the syscall.
00265
00266     \return                 \ref cd_cmd_response
00267 */
00268 int cdrom_exec_cmd(int cmd, void *param);
00269
00270 /** \brief  Execute a CD-ROM command with timeout.
00271
00272     This function executes the specified command using the BIOS syscall for
00273     executing GD-ROM commands with timeout.
00274
00275     \param  cmd             The command number to execute.
00276     \param  param           Data to pass to the syscall.
00277     \param  timeout         Timeout in milliseconds.
00278
00279     \return                 \ref cd_cmd_response
00280 */
00281 int cdrom_exec_cmd_timed(int cmd, void *param, int timeout);
00282
00283 /** \brief  Get the status of the GD-ROM drive.
00284
```

```
00285     \param  status          Space to return the drive's status.
00286     \param  disc_type       Space to return the type of disc in the drive.
00287
00288     \return                 \ref cd_cmd_response
00289     \see    cd_status_values
00290     \see    cd_disc_types
00291 */
00292 int cdrom_get_status(int *status, int *disc_type);
00293
00294 /** \brief    Change the datatype of disc.
00295
00296     \note                   This function is formally deprecated. It should not
00297                             be used in any future code, and may be removed in
00298                             the future. You should instead use
00299                             cdrom_change_datatype.
00300 */
00301 int cdrom_change_datatype(int sector_part, int cdxa, int sector_size)
00302                         __depr("Use cdrom_change_datatype instead.");
00303
00304 /** \brief    Change the datatype of disc.
00305
00306     This function will take in all parameters to pass to the change_datatype
00307     syscall. This allows these parameters to be modified without a reinit.
00308     Each parameter allows -1 as a default, which is tied to the former static
00309     values provided by cdrom_reinit and cdrom_set_sector_size.
00310
00311     \param sector_part      How much of each sector to return.
00312     \param cdxa             What CDXA mode to read as (if applicable).
00313     \param sector_size      What sector size to read (eg. - 2048, 2532).
00314
00315     \return                 \ref cd_cmd_response
00316     \see    cd_read_sector_part
00317 */
00318 int cdrom_change_datatype(int sector_part, int cdxa, int sector_size);
00319
00320 /** \brief  Re-initialize the GD-ROM drive.
00321
00322     This function is for reinitializing the GD-ROM drive after a disc change to
00323     its default settings. Calls cdrom_reinit(-1,-1,-1)
00324
00325     \return                 \ref cd_cmd_response
00326     \see    cdrom_reinit_ex
00327 */
00328 int cdrom_reinit(void);
00329
00330 /** \brief    Re-initialize the GD-ROM drive with custom parameters.
00331
00332     At the end of each cdrom_reinit(), cdrom_change_datatype is called.
00333     This passes in the requested values to that function after
00334     reinitialization, as opposed to defaults.
00335
00336     \param sector_part      How much of each sector to return.
00337     \param cdxa             What CDXA mode to read as (if applicable).
00338     \param sector_size      What sector size to read (eg. - 2048, 2532).
00339
00340     \return                 \ref cd_cmd_response
00341     \see    cd_read_sector_part
00342     \see    cdrom_change_datatype
00343 */
00344 int cdrom_reinit_ex(int sector_part, int cdxa, int sector_size);
00345
00346 /** \brief  Read the table of contents from the disc.
00347
00348     This function reads the TOC from the specified session of the disc.
00349
00350     \param  toc_buffer      Space to store the returned TOC in.
00351     \param  session         The session of the disc to read.
00352     \return                 \ref cd_cmd_response
00353 */
00354 int cdrom_read_toc(CDROM_TOC *toc_buffer, int session);
00355
00356 /** \brief  Read one or more sector from a CD-ROM.
00357
00358     This function reads the specified number of sectors from the disc, starting
00359     where requested. This will respect the size of the sectors set with
00360     cdrom_change_datatype(). The buffer must have enough space to store the
00361     specified number of sectors.
00362
00363     \param  buffer          Space to store the read sectors.
00364     \param  sector          The sector to start reading from.
00365     \param  cnt             The number of sectors to read.
```

```
00366     \param  mode            DMA or PIO
00367     \return                 \ref cd_cmd_response
00368     \see    cd_read_sector_mode
00369 */
00370 int cdrom_read_sectors_ex(void *buffer, int sector, int cnt, int mode);
00371
00372 /** \brief  Read one or more sector from a CD-ROM in PIO mode.
00373
00374     Default version of cdrom_read_sectors_ex, which forces PIO mode.
00375
00376     \param  buffer          Space to store the read sectors.
00377     \param  sector          The sector to start reading from.
00378     \param  cnt             The number of sectors to read.
00379     \return                 \ref cd_cmd_response
00380     \see    cdrom_read_sectors_ex
00381 */
00382 int cdrom_read_sectors(void *buffer, int sector, int cnt);
00383
00384 /** \brief    Read subcode data from the most recently read sectors.
00385
00386     After reading sectors, this can pull subcode data regarding the sectors
00387     read. If reading all subcode data with CD_SUB_CURRENT_POSITION, this needs
00388     to be performed one sector at a time.
00389
00390     \param  buffer          Space to store the read subcode data.
00391     \param  buflen          Amount of data to be read.
00392     \param  which           Which subcode type do you wish to get.
00393
00394     \return                 \ref cd_cmd_response
00395     \see    cd_read_subcode_type
00396 */
00397 int cdrom_get_subcode(void *buffer, int buflen, int which);
00398
00399 /** \brief  Locate the sector of the data track.
00400
00401     This function will search the toc for the last entry that has a CTRL value
00402     of 4, and return its FAD address.
00403
00404     \param  toc             The TOC to search through.
00405     \return                 The FAD of the track, or 0 if none is found.
00406 */
00407 uint32 cdrom_locate_data_track(CDROM_TOC *toc);
00408
00409 /** \brief  Play CDDA audio tracks or sectors.
00410
00411     This function starts playback of CDDA audio.
00412
00413     \param  start           The track or sector to start playback from.
00414     \param  end             The track or sector to end playback at.
00415     \param  loops           The number of times to repeat (max of 15).
00416     \param  mode            The mode to play (see \ref cdda_read_modes).
00417     \return                 \ref cd_cmd_response
00418 */
00419 int cdrom_cdda_play(uint32 start, uint32 end, uint32 loops, int mode);
00420
00421 /** \brief  Pause CDDA audio playback.
00422
00423     \return                 \ref cd_cmd_response
00424 */
00425 int cdrom_cdda_pause(void);
00426
00427 /** \brief  Resume CDDA audio playback after a pause.
00428
00429     \return                 \ref cd_cmd_response
00430 */
00431 int cdrom_cdda_resume(void);
00432
00433 /** \brief  Spin down the CD.
00434
00435     This stops the disc in the drive from spinning until it is accessed again.
00436
00437     \return                 \ref cd_cmd_response
00438 */
00439 int cdrom_spin_down(void);
00440
00441 /** \brief  Initialize the GD-ROM for reading CDs.
00442
00443     This initializes the CD-ROM reading system, reactivating the drive and
00444     handling initial setup of the disc.
00445
00446     \retval 0               On success.
```

```
00447     \retval -1              Already initted, shutdown before initting again.
00448 */
00449 int cdrom_init(void);
00450
00451 /** \brief  Shutdown the CD reading system. */
00452 void cdrom_shutdown(void);
00453
00454 __END_DECLS
00455
00456 #endif  /* __DC_CDROM_H */
```

9.168 kernel/arch/dreamcast/include/dc/dmac.h File Reference

Macros to access the DMA controller registers.

```
#include <sys/cdefs.h>
```

Macros

- #define DMAC_BASE 0xffa00000
- #define DMAC_DMAOR (*((vuint32 *)(DMAC_BASE + 0x40)))

  A register that dictates the overall operation of the DMAC.

DMA Source Address Registers (SAR0-SAR3)

These registers designate the source address for DMA transfers. Currently we only support 32-byte boundary addresses.

- #define DMAC_SAR0 (*((vuint32 *)(DMAC_BASE + 0x00)))
- #define DMAC_SAR1 (*((vuint32 *)(DMAC_BASE + 0x10)))
- #define DMAC_SAR2 (*((vuint32 *)(DMAC_BASE + 0x20)))
- #define DMAC_SAR3 (*((vuint32 *)(DMAC_BASE + 0x30)))

DMA Destination Address Registers (DAR0-DAR3)

These registers designate the destination address for DMA transfers. Currently we only support 32-byte boundary addresses.

- #define DMAC_DAR0 (*((vuint32 *)(DMAC_BASE + 0x04)))
- #define DMAC_DAR1 (*((vuint32 *)(DMAC_BASE + 0x14)))
- #define DMAC_DAR2 (*((vuint32 *)(DMAC_BASE + 0x24)))
- #define DMAC_DAR3 (*((vuint32 *)(DMAC_BASE + 0x34)))

DMA Transfer Count Registers (DMATCR0-DMATCR3)

These registers define the transfer count for each DMA channel. The count is defined as: num_bytes_to_transfer/32

- #define DMAC_DMATCR0 (*((vuint32 *)(DMAC_BASE + 0x08)))
- #define DMAC_DMATCR1 (*((vuint32 *)(DMAC_BASE + 0x18)))
- #define DMAC_DMATCR2 (*((vuint32 *)(DMAC_BASE + 0x28)))
- #define DMAC_DMATCR3 (*((vuint32 *)(DMAC_BASE + 0x38)))

DMA Channel Control Registers (CHCR0-CHCR3)

These registers configure the operating mode and transfer methodology for each channel.

For DMAC_CHCR2, it should always be set to 0x12c1 (source address incremented, burst mode, interrupt disable, DMA enable).

For DMAC_CHCR1 and DMAC_CHCR3, it would probably be set to 0x1241 (source address incremented, cycle steal mode, interrupt disable, DMA enable).

- #define DMAC_CHCR0 (∗((vuint32 ∗)(DMAC_BASE + 0x0c)))
- #define DMAC_CHCR1 (∗((vuint32 ∗)(DMAC_BASE + 0x1c)))
- #define DMAC_CHCR2 (∗((vuint32 ∗)(DMAC_BASE + 0x2c)))
- #define DMAC_CHCR3 (∗((vuint32 ∗)(DMAC_BASE + 0x3c)))

List of helpful masks to check operations

The DMAOR_STATUS_MASK captures the On-Demand Data Transfer Mode (Bit 15), Address Error Flag (Bit 2), NMI Flag (Bit 1), and DMAC Master Enable (Bit 0).

The DMAOR_NORMAL_OPERATION is a state where DMAC Master Enable is active, and the On-Demand Data Transfer Mode is not set, with no address errors or NMI inputs.

- #define DMAOR_STATUS_MASK 0x8007
- #define DMAOR_NORMAL_OPERATION 0x8001

9.168.1 Detailed Description

Macros to access the DMA controller registers.

This header provides a set of macros to facilitate checking the values of various DMA channels on the system.

DMA channel 0 and its registers (DMAC_SAR0, DMAC_DAR0, DMAC_DMATCR0, DMAC_CHCR0) are used by the hardware and not accessible to us but are documented here anyway.

DMA channel 2 is strictly used to transfer data to the PVR/TA.

DMA channel 1 & 3 are free to use.

Author

Andy Barajas

9.168.2 Macro Definition Documentation

DMAC_BASE

```
#define DMAC_BASE 0xffa00000
```

DMAC_CHCR0

```
#define DMAC_CHCR0 (*((vuint32 *)(DMAC_BASE + 0x0c)))
```

DMAC_CHCR1

```
#define DMAC_CHCR1 (*((vuint32 *)(DMAC_BASE + 0x1c)))
```

DMAC_CHCR2

```
#define DMAC_CHCR2 (*((vuint32 *)(DMAC_BASE + 0x2c)))
```

DMAC_CHCR3

```
#define DMAC_CHCR3 (*((vuint32 *)(DMAC_BASE + 0x3c)))
```

DMAC_DAR0

```
#define DMAC_DAR0 (*((vuint32 *)(DMAC_BASE + 0x04)))
```

DMAC_DAR1

```
#define DMAC_DAR1 (*((vuint32 *)(DMAC_BASE + 0x14)))
```

DMAC_DAR2

```
#define DMAC_DAR2 (*((vuint32 *)(DMAC_BASE + 0x24)))
```

DMAC_DAR3

```
#define DMAC_DAR3 (*((vuint32 *)(DMAC_BASE + 0x34)))
```

DMAC_DMAOR

```
#define DMAC_DMAOR (*((vuint32 *)(DMAC_BASE + 0x40)))
```

A register that dictates the overall operation of the DMAC.

So far we only use it check the status of DMA operations.

DMAC_DMATCR0

```
#define DMAC_DMATCR0 (*((vuint32 *)(DMAC_BASE + 0x08)))
```

DMAC_DMATCR1

```
#define DMAC_DMATCR1 (*((vuint32 *)(DMAC_BASE + 0x18)))
```

DMAC_DMATCR2

```
#define DMAC_DMATCR2 (*((vuint32 *)(DMAC_BASE + 0x28)))
```

DMAC_DMATCR3

```
#define DMAC_DMATCR3 (*((vuint32 *)(DMAC_BASE + 0x38)))
```

DMAC_SAR0

```
#define DMAC_SAR0 (*((vuint32 *)(DMAC_BASE + 0x00)))
```

DMAC_SAR1

```
#define DMAC_SAR1 (*((vuint32 *)(DMAC_BASE + 0x10)))
```

DMAC_SAR2

```
#define DMAC_SAR2 (*((vuint32 *)(DMAC_BASE + 0x20)))
```

DMAC_SAR3

```
#define DMAC_SAR3 (*((vuint32 *)(DMAC_BASE + 0x30)))
```

DMAOR_NORMAL_OPERATION

```
#define DMAOR_NORMAL_OPERATION 0x8001
```

DMAOR_STATUS_MASK

```
#define DMAOR_STATUS_MASK 0x8007
```

9.169 dmac.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dmac.h
00004    Copyright (C) 2023 Andy Barajas
00005
00006 */
00007
00008 /** \file   dc/dmac.h
00009     \brief  Macros to access the DMA controller
00010     registers.
00011
00012     This header provides a set of macros to facilitate checking
00013     the values of various DMA channels on the system.
00014
00015     DMA channel 0 and its registers (DMAC_SAR0, DMAC_DAR0, DMAC_DMATCR0,
00016     DMAC_CHCR0) are used by the hardware and not accessible to us but are
00017     documented here anyway.
00018
00019     DMA channel 2 is strictly used to transfer data to the PVR/TA.
00020
00021     DMA channel 1 & 3 are free to use.
00022
00023     \author Andy Barajas
00024 */
00025
00026 #ifndef __DC_DMAC_H
00027 #define __DC_DMAC_H
00028
00029 #include <sys/cdefs.h>
00030 __BEGIN_DECLS
00031
00032 #define DMAC_BASE     0xffa00000
00033
00034 /** \name DMA Source Address Registers (SAR0-SAR3)
00035
00036     These registers designate the source address for DMA transfers.
00037     Currently we only support 32-byte boundary addresses.
00038
00039     @{
00040  */
00041
00042 #define DMAC_SAR0     (*((vuint32 *)(DMAC_BASE + 0x00)))
00043 #define DMAC_SAR1     (*((vuint32 *)(DMAC_BASE + 0x10)))
00044 #define DMAC_SAR2     (*((vuint32 *)(DMAC_BASE + 0x20)))
00045 #define DMAC_SAR3     (*((vuint32 *)(DMAC_BASE + 0x30)))
00046
00047 /** @} */
00048
00049 /** \name  DMA Destination Address Registers (DAR0-DAR3)
00050
00051     These registers designate the destination address for DMA transfers.
00052     Currently we only support 32-byte boundary addresses.
00053
00054     @{
00055  */
00056
00057 #define DMAC_DAR0     (*((vuint32 *)(DMAC_BASE + 0x04)))
00058 #define DMAC_DAR1     (*((vuint32 *)(DMAC_BASE + 0x14)))
00059 #define DMAC_DAR2     (*((vuint32 *)(DMAC_BASE + 0x24)))
00060 #define DMAC_DAR3     (*((vuint32 *)(DMAC_BASE + 0x34)))
00061
00062 /** @} */
00063
00064 /** \name DMA Transfer Count Registers (DMATCR0-DMATCR3)
00065
00066     These registers define the transfer count for each DMA channel. The count
00067     is defined as: num_bytes_to_transfer/32
00068
00069     @{
00070  */
00071
00072 #define DMAC_DMATCR0  (*((vuint32 *)(DMAC_BASE + 0x08)))
00073 #define DMAC_DMATCR1  (*((vuint32 *)(DMAC_BASE + 0x18)))
00074 #define DMAC_DMATCR2  (*((vuint32 *)(DMAC_BASE + 0x28)))
00075 #define DMAC_DMATCR3  (*((vuint32 *)(DMAC_BASE + 0x38)))
00076
```

```
00077 /** @} */
00078
00079 /** \name DMA Channel Control Registers (CHCR0-CHCR3)
00080
00081     These registers configure the operating mode and transfer methodology for
00082     each channel.
00083
00084     For DMAC_CHCR2, it should always be set to 0x12c1 (source address
00085     incremented, burst mode, interrupt disable, DMA enable).
00086
00087     For DMAC_CHCR1 and DMAC_CHCR3, it would probably be set to 0x1241
00088     (source address incremented, cycle steal mode, interrupt disable,
00089     DMA enable).
00090
00091     @{
00092  */
00093
00094 #define DMAC_CHCR0     (*((vuint32 *)(DMAC_BASE + 0x0c)))
00095 #define DMAC_CHCR1     (*((vuint32 *)(DMAC_BASE + 0x1c)))
00096 #define DMAC_CHCR2     (*((vuint32 *)(DMAC_BASE + 0x2c)))
00097 #define DMAC_CHCR3     (*((vuint32 *)(DMAC_BASE + 0x3c)))
00098
00099 /** @} */
00100
00101
00102 /**
00103     \brief A register that dictates the overall operation of the DMAC.
00104
00105     So far we only use it check the status of DMA operations.
00106
00107  */
00108 #define DMAC_DMAOR     (*((vuint32 *)(DMAC_BASE + 0x40)))
00109
00110 /** \name   List of helpful masks to check operations
00111
00112     The DMAOR_STATUS_MASK captures the On-Demand Data Transfer Mode (Bit 15),
00113     Address Error Flag (Bit 2), NMI Flag (Bit 1), and DMAC Master Enable (Bit 0).
00114
00115     The DMAOR_NORMAL_OPERATION is a state where DMAC Master Enable is active,
00116     and the On-Demand Data Transfer Mode is not set, with no address errors
00117     or NMI inputs.
00118
00119     @{
00120 */
00121
00122 #define DMAOR_STATUS_MASK       0x8007
00123 #define DMAOR_NORMAL_OPERATION  0x8001
00124
00125 /** @} */
00126
00127 __END_DECLS
00128
00129 #endif  /* __DC_DMAC_H */
00130
```

9.170 kernel/arch/dreamcast/include/dc/fb_console.h File Reference

A simple dbgio interface to draw to the framebuffer.

```
#include <sys/cdefs.h>
#include <kos/dbgio.h>
```

Functions

- void dbgio_fb_set_target (uint16 *t, int w, int h, int borderx, int bordery)

  Set the target for the framebuffer dbgio device.

9.170.1 Detailed Description

A simple dbgio interface to draw to the framebuffer.

This file contains definitions to interact with a simple framebuffer dbgio interface. This interface can be moved around in memory and can have its width and height set so that you can truly customize it to the environment as needed. This utilizes the bios font functionality to actually draw any characters.

To actually use the framebuffer device, pass "fb" as the parameter to dbgio_dev_select().

Author

Lawrence Sebald

9.170.2 Function Documentation

dbgio_fb_set_target()

```
void dbgio_fb_set_target (
             uint16 * t,
             int w,
             int h,
             int borderx,
             int bordery )
```

Set the target for the framebuffer dbgio device.

This function allows you to set a target for the dbgio device on the framebuffer. This allows you to do things like setting it to render to a texture rather than to the whole framebuffer, for instance.

The default setup for the framebuffer dbgio device is to print to the full 640x480 framebuffer (minus a 32-pixel border around the outside). If you change this, you can restore the original functionality by passing NULL for t, 640 for w, 480 for h, and 32 for borderx and bordery.

Parameters

t	The target in memory to render to.
w	The width of the target.
h	The height of the target.
borderx	How much border to leave around the target in the X direction.
bordery	How much border to leave around the target in the Y direction.

9.171 fb_console.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
```

```
00003    dc/fb_console.h
00004    Copyright (C) 2009 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   dc/fb_console.h
00009     \brief  A simple dbgio interface to draw to the framebuffer.
00010
00011     This file contains definitions to interact with a simple framebuffer dbgio
00012     interface. This interface can be moved around in memory and can have its
00013     width and height set so that you can truly customize it to the environment
00014     as needed. This utilizes the bios font functionality to actually draw any
00015     characters.
00016
00017     To actually use the framebuffer device, pass "fb" as the parameter to
00018     dbgio_dev_select().
00019
00020     \author Lawrence Sebald
00021 */
00022
00023 #ifndef __DC_FB_CONSOLE_H
00024 #define __DC_FB_CONSOLE_H
00025
00026 #include <sys/cdefs.h>
00027 __BEGIN_DECLS
00028
00029 #include <kos/dbgio.h>
00030
00031 /* \cond */
00032 extern dbgio_handler_t dbgio_fb;
00033 /* \endcond */
00034
00035 /** \brief  Set the target for the framebuffer dbgio device.
00036
00037     This function allows you to set a target for the dbgio device on the
00038     framebuffer. This allows you to do things like setting it to render to a
00039     texture rather than to the whole framebuffer, for instance.
00040
00041     The default setup for the framebuffer dbgio device is to print to the full
00042     640x480 framebuffer (minus a 32-pixel border around the outside). If you
00043     change this, you can restore the original functionality by passing NULL for
00044     t, 640 for w, 480 for h, and 32 for borderx and bordery.
00045
00046     \param  t               The target in memory to render to.
00047     \param  w               The width of the target.
00048     \param  h               The height of the target.
00049     \param  borderx         How much border to leave around the target in the
00050                             X direction.
00051     \param  bordery         How much border to leave around the target in the
00052                             Y direction.
00053 */
00054 void dbgio_fb_set_target(uint16 *t, int w, int h, int borderx, int bordery);
00055
00056 __END_DECLS
00057
00058 #endif /* __DC_FB_CONSOLE_H */
```

9.172 kernel/arch/dreamcast/include/dc/fifo.h File Reference

Macros to assess FIFO status.

`#include <sys/cdefs.h>`

Macros

- #define FIFO_STATUS (*(vuint32 const *)0xa05f688c)

  Address of the FIFO status register. Accessing this value provides the current status of all FIFOs.

FIFO Status Indicators

Note

To determine the empty status of a specific FIFO, AND the desired FIFO status mask with the value returned by FIFO_STATUS.

If the resulting value is non-zero, the FIFO is not empty. Otherwise, it is empty.

- #define FIFO_AICA (1 << 0) /** \brief AICA FIFO status mask. */
- #define FIFO_BBA (1 << 1) /** \brief BBA FIFO status mask. */
- #define FIFO_EXT2 (1 << 2) /** \brief EXT2 FIFO status mask. */
- #define FIFO_EXTDEV (1 << 3) /** \brief EXTDEV FIFO status mask. */
- #define FIFO_G2 (1 << 4) /** \brief G2 FIFO status mask. */
- #define FIFO_SH4 (1 << 5) /** \brief SH4 FIFO status mask. */

9.172.1 Detailed Description

Macros to assess FIFO status.

This header provides a set of macros to facilitate checking the status of various FIFOs on the system.

Author

Andy Barajas

9.172.2 Macro Definition Documentation

FIFO_AICA

```
#define FIFO_AICA (1 << 0) /** \brief AICA FIFO status mask.  */
```

FIFO_BBA

```
#define FIFO_BBA (1 << 1) /** \brief BBA FIFO status mask.  */
```

FIFO_EXT2

```
#define FIFO_EXT2 (1 << 2) /** \brief EXT2 FIFO status mask.  */
```

FIFO_EXTDEV

```
#define FIFO_EXTDEV (1 << 3) /** \brief EXTDEV FIFO status mask.  */
```

FIFO_G2

```
#define FIFO_G2 (1 << 4) /** \brief G2 FIFO status mask.  */
```

FIFO_SH4

```
#define FIFO_SH4 (1 << 5) /** \brief SH4 FIFO status mask.  */
```

FIFO_STATUS

```
#define FIFO_STATUS (*(vuint32 const *)0xa05f688c)
```

Address of the FIFO status register. Accessing this value provides the current status of all FIFOs.

9.173 fifo.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    fifo.h
00004    Copyright (C) 2023 Andy Barajas
00005
00006 */
00007
00008 /** \file   dc/fifo.h
00009     \brief  Macros to assess FIFO status.
00010
00011     This header provides a set of macros to facilitate checking
00012     the status of various FIFOs on the system.
00013     \author Andy Barajas
00014 */
00015
00016 #ifndef __DC_FIFO_H
00017 #define __DC_FIFO_H
00018
00019 #include <sys/cdefs.h>
00020 __BEGIN_DECLS
00021
00022 /** \brief Address of the FIFO status register.
00023     Accessing this value provides the current status of all FIFOs.
00024
00025 */
00026 #define FIFO_STATUS     (*(vuint32 const *)0xa05f688c)
00027
00028 /** \name        FIFO Status Indicators
00029
00030     \note
00031     To determine the empty status of a specific FIFO, AND the desired FIFO
00032     status mask with the value returned by FIFO_STATUS.
00033
00034     If the resulting value is non-zero, the FIFO is not empty. Otherwise,
00035     it is empty.
00036
00037     @{
00038 */
00039
00040 #define FIFO_AICA   (1 << 0)   /** \brief AICA FIFO status mask. */
00041 #define FIFO_BBA    (1 << 1)   /** \brief BBA FIFO status mask. */
00042 #define FIFO_EXT2   (1 << 2)   /** \brief EXT2 FIFO status mask. */
00043 #define FIFO_EXTDEV (1 << 3)   /** \brief EXTDEV FIFO status mask. */
00044 #define FIFO_G2     (1 << 4)   /** \brief G2 FIFO status mask. */
00045 #define FIFO_SH4    (1 << 5)   /** \brief SH4 FIFO status mask. */
00046
00047 /** @} */
00048
00049 __END_DECLS
00050
00051 #endif  /* __DC_FIFO_H */
00052
```

9.174 kernel/arch/dreamcast/include/dc/flashrom.h File Reference

Dreamcast flashrom read/write support.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Data Structures

- struct flashrom_syscfg_t

  System configuration structure.
- struct flashrom_ispcfg_t

  ISP configuration structure.

Macros

- #define FLASHROM_PT_SYSTEM 0

  Factory settings (read-only, 8K)
- #define FLASHROM_PT_RESERVED 1

  reserved (all 0s, 8K)
- #define FLASHROM_PT_BLOCK_1 2

  Block allocated (16K)
- #define FLASHROM_PT_SETTINGS 3

  Game settings (block allocated, 32K)
- #define FLASHROM_PT_BLOCK_2 4

  Block allocated (64K)
- #define FLASHROM_B1_SYSCFG 0x05

  System config (BLOCK_1)
- #define FLASHROM_B1_PW_SETTINGS_1 0x80

  PlanetWeb settings (BLOCK_1)
- #define FLASHROM_B1_PW_SETTINGS_2 0x81

  PlanetWeb settings (BLOCK_1)
- #define FLASHROM_B1_PW_SETTINGS_3 0x82

  PlanetWeb settings (BLOCK_1)
- #define FLASHROM_B1_PW_SETTINGS_4 0x83

  PlanetWeb settings (BLOCK_1)
- #define FLASHROM_B1_PW_SETTINGS_5 0x84

  PlanetWeb settings (BLOCK_1)
- #define FLASHROM_B1_PW_PPP1 0xC0

  PlanetWeb PPP settings (BLOCK_1)
- #define FLASHROM_B1_PW_PPP2 0xC1

  PlanetWeb PPP settings (BLOCK_1)
- #define FLASHROM_B1_PW_DNS 0xC2

  PlanetWeb DNS settings (BLOCK_1)
- #define FLASHROM_B1_PW_EMAIL1 0xC3

PlanetWeb Email settings (BLOCK_1)

- #define FLASHROM_B1_PW_EMAIL2 0xC4

  PlanetWeb Email settings (BLOCK_1)
- #define FLASHROM_B1_PW_EMAIL_PROXY 0xC5

  PlanetWeb Email/Proxy settings (BLOCK_1)
- #define FLASHROM_B1_DK_PPP1 0xC6

  DreamKey PPP settings (also seen in PW)
- #define FLASHROM_B1_DK_PPP2 0xC7

  DreamKey PPP settings (also seen in PW)
- #define FLASHROM_B1_DK_DNS 0xC8

  DreamKey PPP settings (also seen in PW)
- #define FLASHROM_B1_IP_SETTINGS 0xE0

  IP settings for BBA (BLOCK_1)
- #define FLASHROM_B1_EMAIL 0xE2

  Email address (BLOCK_1)
- #define FLASHROM_B1_SMTP 0xE4

  SMTP server setting (BLOCK_1)
- #define FLASHROM_B1_POP3 0xE5

  POP3 server setting (BLOCK_1)
- #define FLASHROM_B1_POP3LOGIN 0xE6

  POP3 login setting (BLOCK_1)
- #define FLASHROM_B1_POP3PASSWD 0xE7

  POP3 password setting + proxy (BLOCK_1)
- #define FLASHROM_B1_PPPLOGIN 0xE8

  PPP username + proxy (BLOCK_1)
- #define FLASHROM_B1_PPPPASSWD 0xE9

  PPP passwd (BLOCK_1)
- #define FLASHROM_B1_PPPMODEM 0xEB

  PPP modem settings.
- #define FLASHROM_OFFSET_CRC 62

  Location of CRC in each block.
- #define FLASHROM_ERR_NONE 0

  Success.
- #define FLASHROM_ERR_NOT_FOUND -1

  Block not found.
- #define FLASHROM_ERR_NO_PARTITION -2

  Partition not found.
- #define FLASHROM_ERR_READ_PART -3

  Error reading partition.
- #define FLASHROM_ERR_BAD_MAGIC -4

  Invalid block magic.
- #define FLASHROM_ERR_BOGUS_PART -5

  Bogus partition size.
- #define FLASHROM_ERR_NOMEM -6

  Memory allocation failure.
- #define FLASHROM_ERR_READ_BITMAP -7

  Error reading bitmap.

- #define FLASHROM_ERR_EMPTY_PART -8

  Empty partition.
- #define FLASHROM_ERR_READ_BLOCK -9

  Error reading block.
- #define FLASHROM_LANG_JAPANESE 0

  Japanese language code.
- #define FLASHROM_LANG_ENGLISH 1

  English language code.
- #define FLASHROM_LANG_GERMAN 2

  German language code.
- #define FLASHROM_LANG_FRENCH 3

  French language code.
- #define FLASHROM_LANG_SPANISH 4

  Spanish language code.
- #define FLASHROM_LANG_ITALIAN 5

  Italian language code.
- #define FLASHROM_REGION_UNKNOWN 0

  Unknown region.
- #define FLASHROM_REGION_JAPAN 1

  Japanese region.
- #define FLASHROM_REGION_US 2

  US/Canada region.
- #define FLASHROM_REGION_EUROPE 3

  European region.
- #define FLASHROM_ISP_DIALUP 0

  Dialup ISP.
- #define FLASHROM_ISP_DHCP 1

  DHCP-based ethernet.
- #define FLASHROM_ISP_PPPOE 2

  PPPoE-based ethernet.
- #define FLASHROM_ISP_STATIC 3

  Static IP-based ethernet.
- #define FLASHROM_ISP_IP (1 << 0)

  Static IP address.
- #define FLASHROM_ISP_NETMASK (1 << 1)

  Netmask.
- #define FLASHROM_ISP_BROADCAST (1 << 2)

  Broadcast address.
- #define FLASHROM_ISP_GATEWAY (1 << 3)

  Gateway address.
- #define FLASHROM_ISP_DNS (1 << 4)

  DNS servers.
- #define FLASHROM_ISP_HOSTNAME (1 << 5)

  Hostname.
- #define FLASHROM_ISP_EMAIL (1 << 6)

  Email address.
- #define FLASHROM_ISP_SMTP (1 << 7)

SMTP server.

- #define FLASHROM_ISP_POP3 (1 << 8)

  POP3 server.
- #define FLASHROM_ISP_POP3_USER (1 << 9)

  POP3 username.
- #define FLASHROM_ISP_POP3_PASS (1 << 10)

  POP3 password.
- #define FLASHROM_ISP_PROXY_HOST (1 << 11)

  Proxy hostname.
- #define FLASHROM_ISP_PROXY_PORT (1 << 12)

  Proxy port.
- #define FLASHROM_ISP_PPP_USER (1 << 13)

  PPP username.
- #define FLASHROM_ISP_PPP_PASS (1 << 14)

  PPP password.
- #define FLASHROM_ISP_OUT_PREFIX (1 << 15)

  Outside dial prefix.
- #define FLASHROM_ISP_CW_PREFIX (1 << 16)

  Call waiting prefix.
- #define FLASHROM_ISP_REAL_NAME (1 << 17)

  Real name.
- #define FLASHROM_ISP_MODEM_INIT (1 << 18)

  Modem init string.
- #define FLASHROM_ISP_AREA_CODE (1 << 19)

  Area code.
- #define FLASHROM_ISP_LD_PREFIX (1 << 20)

  Long distance prefix.
- #define FLASHROM_ISP_PHONE1 (1 << 21)

  Phone number 1.
- #define FLASHROM_ISP_PHONE2 (1 << 22)

  Phone number 2.
- #define FLASHROM_ISP_DIAL_AREACODE (1 << 0)

  Dial area code before number.
- #define FLASHROM_ISP_USE_PROXY (1 << 1)

  Proxy enabled.
- #define FLASHROM_ISP_PULSE_DIAL (1 << 2)

  Pulse dialing (instead of tone)
- #define FLASHROM_ISP_BLIND_DIAL (1 << 3)

  Blind dial (don't wait for tone)

Functions

- int flashrom_info (int part, int *start_out, int *size_out)

  Retrieve information about the given partition.
- int flashrom_read (int offset, void *buffer_out, int bytes)

  Read data from the flashrom.

- int flashrom_write (int offset, void *buffer, int bytes)
  Write data to the flashrom.
- int flashrom_delete (int offset)
  Delete data from the flashrom.
- int flashrom_get_block (int partid, int blockid, uint8 *buffer_out)
  Get a logical block from the specified partition.
- int flashrom_get_syscfg (flashrom_syscfg_t *out)
  Retrieve the current system configuration settings.
- int flashrom_get_region (void)
  Retrieve the console's region code.
- int flashrom_get_ispcfg (flashrom_ispcfg_t *out)
  Retrieve DreamPassport's ISP configuration.
- int flashrom_get_pw_ispcfg (flashrom_ispcfg_t *out)
  Retrieve PlanetWeb's ISP configuration.

9.174.1 Detailed Description

Dreamcast flashrom read/write support.

This file implements wrappers for the BIOS flashrom syscalls, and some utilities to make it easier to use the flashrom info. Note that because the flash writing can be such a dangerous thing potentially (I haven't deleted my flash to see what happens, but given the info stored here it sounds like a Bad Idea(tm)), extreme care should be taken if you choose to use these functions!

Author

Megan Potter

Lawrence Sebald

9.174.2 Macro Definition Documentation

FLASHROM_OFFSET_CRC

```
#define FLASHROM_OFFSET_CRC 62
```

Location of CRC in each block.

9.174.3 Function Documentation

flashrom_delete()

```
int flashrom_delete (
            int offset )
```

Delete data from the flashrom.

This function implements the FLASHROM_DELETE syscall; given a partition offset, that entire partition of the flashrom will be deleted and all data will be reset to 0xFF bytes.

Note

This does not rewrite the magic block to the start of the partition. It is your responsibility to do this after running this function.

Parameters

offset	The offset of the start of the partition to erase.

Return values

0	On success.
-1	On error.

flashrom_get_block()

```
int flashrom_get_block (
            int partid,
            int blockid,
            uint8 * buffer_out )
```

Get a logical block from the specified partition.

This function retrieves the specified block ID from the given partition. The newest version of the data is returned.

Parameters

partid	The partition ID to look in.
blockid	The logical block ID to look for.
buffer_out	Space to store the data. Must be at least 60 bytes.

Returns

0 on success, <0 on error.

See also

Error values for the flashrom_get_block() function

flashrom_get_ispcfg()

```
int flashrom_get_ispcfg (
            flashrom_ispcfg_t * out )
```

Retrieve DreamPassport's ISP configuration.

This function retrieves the console's ISP settings as set by DreamPassport, if they exist. You should check the valid_↩ fields bitfield for the part of the struct you want before relying on the data.

Parameters

out	Storage for the structure.

Return values

0	On success.
-1	On error (no settings found, or other errors).

flashrom_get_pw_ispcfg()

```
int flashrom_get_pw_ispcfg (
            flashrom_ispcfg_t * out )
```

Retrieve PlanetWeb's ISP configuration.

This function retrieves the console's ISP settings as set by PlanetWeb (1.0 and 2.1 have been verified to work), if they exist. You should check the valid_fields bitfield for the part of the struct you want before relying on the data.

Parameters

out	Storage for the structure.

Return values

0	On success.
-1	On error (i.e, no PlanetWeb settings found).

flashrom_get_region()

```
int flashrom_get_region (
            void  )
```

Retrieve the console's region code.

This function attempts to find the region of the Dreamcast. It may or may not work on 100% of Dreamcasts, apparently.

Returns

A region code (>=0), or error (<0).

See also

Region settings possible in the system

Error values for the flashrom_get_block() function

flashrom_get_syscfg()

```
int flashrom_get_syscfg (
            flashrom_syscfg_t * out )
```

Retrieve the current system configuration settings.

Parameters

out	Storage for the configuration.

Returns

0 on success, <0 on error.

See also

Error values for the flashrom_get_block() function

flashrom_info()

```
int flashrom_info (
            int part,
            int * start_out,
            int * size_out )
```

Retrieve information about the given partition.

This function implements the FLASHROM_INFO syscall; given a partition ID, return two ints specifying the beginning and the size of the partition (respectively) inside the flashrom.

Parameters

part	The partition ID in question.
start_out	Buffer for storing the start address.
size_out	Buffer for storing the size of the partition.

Return values

0	On success.
-1	On error.

flashrom_read()

```
int flashrom_read (
```

```
            int offset,
            void * buffer_out,
            int bytes )
```

Read data from the flashrom.

This function implements the FLASHROM_READ syscall; given a flashrom offset, an output buffer, and a count, this reads data from the flashrom.

Parameters

offset	The offset to read from.
buffer_out	Space to read into.
bytes	The number of bytes to read.

Returns

The number of bytes read if successful, or -1 otherwise.

flashrom_write()

```
int flashrom_write (
            int offset,
            void * buffer,
            int bytes )
```

Write data to the flashrom.

This function implements the FLASHROM_WRITE syscall; given a flashrom offset, an input buffer, and a count, this writes data to the flashrom.

Note

It is not possible to write ones to the flashrom over zeros. If you want to do this, you must save the old data in the flashrom, delete it out, and save the new data back.

Parameters

offset	The offset to write at.
buffer	The data to write.
bytes	The number of bytes to write.

Returns

The number of bytes written if successful, -1 otherwise.

9.175 flashrom.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/flashrom.h
00004    Copyright (C) 2003 Megan Potter
00005    Copyright (C) 2008 Lawrence Sebald
00006
00007 */
00008
00009 /** \file   dc/flashrom.h
00010     \brief  Dreamcast flashrom read/write support.
00011
00012     This file implements wrappers for the BIOS flashrom syscalls, and some
00013     utilities to make it easier to use the flashrom info. Note that because the
00014     flash writing can be such a dangerous thing potentially (I haven't deleted
00015     my flash to see what happens, but given the info stored here it sounds like
00016     a Bad Idea(tm)), extreme care should be taken if you choose to use these
00017     functions!
00018
00019     \author Megan Potter
00020     \author Lawrence Sebald
00021 */
00022
00023 #ifndef __DC_FLASHROM_H
00024 #define __DC_FLASHROM_H
00025
00026 #include <sys/cdefs.h>
00027 __BEGIN_DECLS
00028
00029 #include <arch/types.h>
00030
00031 /** \defgroup fr_parts  Partitions available in the flashrom
00032     @{
00033 */
00034 #define FLASHROM_PT_SYSTEM      0   /**< \brief Factory settings (read-only, 8K) */
00035 #define FLASHROM_PT_RESERVED    1   /**< \brief reserved (all 0s, 8K) */
00036 #define FLASHROM_PT_BLOCK_1     2   /**< \brief Block allocated (16K) */
00037 #define FLASHROM_PT_SETTINGS    3   /**< \brief Game settings (block allocated, 32K) */
00038 #define FLASHROM_PT_BLOCK_2     4   /**< \brief Block allocated (64K) */
00039 /** @} */
00040
00041
00042 /** \defgroup fr_blocks Logical blocks available in the flashrom
00043     @{
00044 */
00045 #define FLASHROM_B1_SYSCFG          0x05    /**< \brief System config (BLOCK_1) */
00046 #define FLASHROM_B1_PW_SETTINGS_1   0x80    /**< \brief PlanetWeb settings (BLOCK_1) */
00047 #define FLASHROM_B1_PW_SETTINGS_2   0x81    /**< \brief PlanetWeb settings (BLOCK_1) */
00048 #define FLASHROM_B1_PW_SETTINGS_3   0x82    /**< \brief PlanetWeb settings (BLOCK_1) */
00049 #define FLASHROM_B1_PW_SETTINGS_4   0x83    /**< \brief PlanetWeb settings (BLOCK_1) */
00050 #define FLASHROM_B1_PW_SETTINGS_5   0x84    /**< \brief PlanetWeb settings (BLOCK_1) */
00051 #define FLASHROM_B1_PW_PPP1         0xC0    /**< \brief PlanetWeb PPP settings (BLOCK_1) */
00052 #define FLASHROM_B1_PW_PPP2         0xC1    /**< \brief PlanetWeb PPP settings (BLOCK_1) */
00053 #define FLASHROM_B1_PW_DNS          0xC2    /**< \brief PlanetWeb DNS settings (BLOCK_1) */
00054 #define FLASHROM_B1_PW_EMAIL1       0xC3    /**< \brief PlanetWeb Email settings (BLOCK_1) */
00055 #define FLASHROM_B1_PW_EMAIL2       0xC4    /**< \brief PlanetWeb Email settings (BLOCK_1) */
00056 #define FLASHROM_B1_PW_EMAIL_PROXY  0xC5    /**< \brief PlanetWeb Email/Proxy settings (BLOCK_1) */
00057 #define FLASHROM_B1_DK_PPP1         0xC6    /**< \brief DreamKey PPP settings (also seen in PW) */
00058 #define FLASHROM_B1_DK_PPP2         0xC7    /**< \brief DreamKey PPP settings (also seen in PW) */
00059 #define FLASHROM_B1_DK_DNS          0xC8    /**< \brief DreamKey PPP settings (also seen in PW) */
00060 #define FLASHROM_B1_IP_SETTINGS     0xE0    /**< \brief IP settings for BBA (BLOCK_1) */
00061 #define FLASHROM_B1_EMAIL           0xE2    /**< \brief Email address (BLOCK_1) */
00062 #define FLASHROM_B1_SMTP            0xE4    /**< \brief SMTP server setting (BLOCK_1) */
00063 #define FLASHROM_B1_POP3            0xE5    /**< \brief POP3 server setting (BLOCK_1) */
00064 #define FLASHROM_B1_POP3LOGIN       0xE6    /**< \brief POP3 login setting (BLOCK_1) */
00065 #define FLASHROM_B1_POP3PASSWD      0xE7    /**< \brief POP3 password setting + proxy (BLOCK_1) */
00066 #define FLASHROM_B1_PPPLOGIN        0xE8    /**< \brief PPP username + proxy (BLOCK_1) */
00067 #define FLASHROM_B1_PPPPASSWD       0xE9    /**< \brief PPP passwd (BLOCK_1) */
00068 #define FLASHROM_B1_PPPMODEM        0xEB    /**< \brief PPP modem settings */
00069 /** @} */
```

```
00070
00071 #define FLASHROM_OFFSET_CRC         62      /**< \brief Location of CRC in each block */
00072
00073 /** \defgroup fr_errs   Error values for the flashrom_get_block() function
00074     @{
00075 */
00076 #define FLASHROM_ERR_NONE           0       /**< \brief Success */
00077 #define FLASHROM_ERR_NOT_FOUND      -1      /**< \brief Block not found */
00078 #define FLASHROM_ERR_NO_PARTITION   -2      /**< \brief Partition not found */
00079 #define FLASHROM_ERR_READ_PART      -3      /**< \brief Error reading partition */
00080 #define FLASHROM_ERR_BAD_MAGIC      -4      /**< \brief Invalid block magic */
00081 #define FLASHROM_ERR_BOGUS_PART     -5      /**< \brief Bogus partition size */
00082 #define FLASHROM_ERR_NOMEM          -6      /**< \brief Memory allocation failure */
00083 #define FLASHROM_ERR_READ_BITMAP    -7      /**< \brief Error reading bitmap */
00084 #define FLASHROM_ERR_EMPTY_PART     -8      /**< \brief Empty partition */
00085 #define FLASHROM_ERR_READ_BLOCK     -9      /**< \brief Error reading block */
00086 /** @} */
00087
00088 /** \brief  Retrieve information about the given partition.
00089
00090     This function implements the FLASHROM_INFO syscall; given a partition ID,
00091     return two ints specifying the beginning and the size of the partition
00092     (respectively) inside the flashrom.
00093
00094     \param  part            The partition ID in question.
00095     \param  start_out       Buffer for storing the start address.
00096     \param  size_out        Buffer for storing the size of the partition.
00097     \retval 0               On success.
00098     \retval -1              On error.
00099 */
00100 int flashrom_info(int part, int * start_out, int * size_out);
00101
00102 /** \brief  Read data from the flashrom.
00103
00104     This function implements the FLASHROM_READ syscall; given a flashrom offset,
00105     an output buffer, and a count, this reads data from the flashrom.
00106
00107     \param  offset          The offset to read from.
00108     \param  buffer_out      Space to read into.
00109     \param  bytes           The number of bytes to read.
00110     \return                 The number of bytes read if successful, or -1
00111                             otherwise.
00112 */
00113 int flashrom_read(int offset, void * buffer_out, int bytes);
00114
00115 /** \brief  Write data to the flashrom.
00116
00117     This function implements the FLASHROM_WRITE syscall; given a flashrom
00118     offset, an input buffer, and a count, this writes data to the flashrom.
00119
00120     \note It is not possible to write ones to the flashrom over zeros. If you
00121     want to do this, you must save the old data in the flashrom, delete it out,
00122     and save the new data back.
00123
00124     \param  offset          The offset to write at.
00125     \param  buffer          The data to write.
00126     \param  bytes           The number of bytes to write.
00127     \return                 The number of bytes written if successful, -1
00128                             otherwise.
00129 */
00130 int flashrom_write(int offset, void * buffer, int bytes);
00131
00132 /** \brief  Delete data from the flashrom.
00133
00134     This function implements the FLASHROM_DELETE syscall; given a partition
00135     offset, that entire partition of the flashrom will be deleted and all data
00136     will be reset to 0xFF bytes.
00137
00138     \note This does not rewrite the magic block to the start of the partition.
00139     It is your responsibility to do this after running this function.
00140
00141     \param  offset          The offset of the start of the partition to erase.
00142     \retval 0               On success.
00143     \retval -1              On error.
00144 */
00145 int flashrom_delete(int offset);
00146
00147
00148 /* Medium-level functions */
00149 /** \brief  Get a logical block from the specified partition.
00150
```

```
00151     This function retrieves the specified block ID from the given partition. The
00152     newest version of the data is returned.
00153
00154     \param  partid          The partition ID to look in.
00155     \param  blockid         The logical block ID to look for.
00156     \param  buffer_out      Space to store the data. Must be at least 60 bytes.
00157     \return                 0 on success, <0 on error.
00158     \see    fr_errs
00159 */
00160 int flashrom_get_block(int partid, int blockid, uint8 * buffer_out);
00161
00162
00163 /* Higher level functions */
00164
00165 /** \defgroup fr_langs  Language settings possible in the BIOS menu
00166
00167     This set of constants will be returned as the language value in the
00168     flashrom_syscfg_t structure.
00169
00170     @{
00171 */
00172 #define FLASHROM_LANG_JAPANESE  0   /**< \brief Japanese language code */
00173 #define FLASHROM_LANG_ENGLISH   1   /**< \brief English language code */
00174 #define FLASHROM_LANG_GERMAN    2   /**< \brief German language code */
00175 #define FLASHROM_LANG_FRENCH    3   /**< \brief French language code */
00176 #define FLASHROM_LANG_SPANISH   4   /**< \brief Spanish language code */
00177 #define FLASHROM_LANG_ITALIAN   5   /**< \brief Italian language code */
00178 /** @} */
00179
00180 /** \brief  System configuration structure.
00181
00182     This structure is filled in with the settings set in the BIOS from the
00183     flashrom_get_syscfg() function.
00184
00185     \headerfile dc/flashrom.h
00186 */
00187 typedef struct flashrom_syscfg {
00188     int language;   /**< \brief Language setting.
00189                          \see fr_langs */
00190     int audio;      /**< \brief Stereo/mono setting. 0 == mono, 1 == stereo */
00191     int autostart;  /**< \brief Autostart discs? 0 == off, 1 == on */
00192 } flashrom_syscfg_t;
00193
00194 /** \brief  Retrieve the current system configuration settings.
00195     \param  out             Storage for the configuration.
00196     \return                 0 on success, <0 on error.
00197     \see    fr_errs
00198 */
00199 int flashrom_get_syscfg(flashrom_syscfg_t * out);
00200
00201
00202 /** \defgroup fr_region Region settings possible in the system
00203
00204     One of these values should be returned by flashrom_get_region().
00205
00206     @{
00207 */
00208 #define FLASHROM_REGION_UNKNOWN 0   /**< \brief Unknown region */
00209 #define FLASHROM_REGION_JAPAN   1   /**< \brief Japanese region */
00210 #define FLASHROM_REGION_US      2   /**< \brief US/Canada region */
00211 #define FLASHROM_REGION_EUROPE  3   /**< \brief European region */
00212 /** @} */
00213
00214 /** \brief  Retrieve the console's region code.
00215
00216     This function attempts to find the region of the Dreamcast. It may or may
00217     not work on 100% of Dreamcasts, apparently.
00218
00219     \return                 A region code (>=0), or error (<0).
00220     \see    fr_region
00221     \see    fr_errs
00222 */
00223 int flashrom_get_region(void);
00224
00225 /** \defgroup fr_method Connection method types
00226
00227     These values are representative of what type of ISP is configured in the
00228     flashrom.
00229
00230     @{
00231 */
```

```
00232 #define FLASHROM_ISP_DIALUP 0   /**< \brief Dialup ISP */
00233 #define FLASHROM_ISP_DHCP   1   /**< \brief DHCP-based ethernet */
00234 #define FLASHROM_ISP_PPPOE  2   /**< \brief PPPoE-based ethernet */
00235 #define FLASHROM_ISP_STATIC 3   /**< \brief Static IP-based ethernet */
00236
00237
00238 /** @} */
00239
00240 /** \defgroup fr_fields Valid field constants for the flashrom_ispcfg_t struct
00241
00242     The valid_fields field of the flashrom_ispcfg_t will have some combination
00243     of these ORed together to represent what data is filled in and believed
00244     valid.
00245
00246     @{
00247 */
00248 #define FLASHROM_ISP_IP          (1 <<  0)   /**< \brief Static IP address */
00249 #define FLASHROM_ISP_NETMASK     (1 <<  1)   /**< \brief Netmask */
00250 #define FLASHROM_ISP_BROADCAST   (1 <<  2)   /**< \brief Broadcast address */
00251 #define FLASHROM_ISP_GATEWAY     (1 <<  3)   /**< \brief Gateway address */
00252 #define FLASHROM_ISP_DNS         (1 <<  4)   /**< \brief DNS servers */
00253 #define FLASHROM_ISP_HOSTNAME    (1 <<  5)   /**< \brief Hostname */
00254 #define FLASHROM_ISP_EMAIL       (1 <<  6)   /**< \brief Email address */
00255 #define FLASHROM_ISP_SMTP        (1 <<  7)   /**< \brief SMTP server */
00256 #define FLASHROM_ISP_POP3        (1 <<  8)   /**< \brief POP3 server */
00257 #define FLASHROM_ISP_POP3_USER   (1 <<  9)   /**< \brief POP3 username */
00258 #define FLASHROM_ISP_POP3_PASS   (1 << 10)   /**< \brief POP3 password */
00259 #define FLASHROM_ISP_PROXY_HOST  (1 << 11)   /**< \brief Proxy hostname */
00260 #define FLASHROM_ISP_PROXY_PORT  (1 << 12)   /**< \brief Proxy port */
00261 #define FLASHROM_ISP_PPP_USER    (1 << 13)   /**< \brief PPP username */
00262 #define FLASHROM_ISP_PPP_PASS    (1 << 14)   /**< \brief PPP password */
00263 #define FLASHROM_ISP_OUT_PREFIX  (1 << 15)   /**< \brief Outside dial prefix */
00264 #define FLASHROM_ISP_CW_PREFIX   (1 << 16)   /**< \brief Call waiting prefix */
00265 #define FLASHROM_ISP_REAL_NAME   (1 << 17)   /**< \brief Real name */
00266 #define FLASHROM_ISP_MODEM_INIT  (1 << 18)   /**< \brief Modem init string */
00267 #define FLASHROM_ISP_AREA_CODE   (1 << 19)   /**< \brief Area code */
00268 #define FLASHROM_ISP_LD_PREFIX   (1 << 20)   /**< \brief Long distance prefix */
00269 #define FLASHROM_ISP_PHONE1      (1 << 21)   /**< \brief Phone number 1 */
00270 #define FLASHROM_ISP_PHONE2      (1 << 22)   /**< \brief Phone number 2 */
00271 /** @} */
00272
00273 /** \defgroup fr_flags  Flags for the flashrom_ispcfg_t struct
00274
00275     The flags field of the flashrom_ispcfg_t will have some combination of these
00276     ORed together to represent what settings were set.
00277
00278     @{
00279 */
00280 #define FLASHROM_ISP_DIAL_AREACODE   (1 <<  0)   /**< \brief Dial area code before number */
00281 #define FLASHROM_ISP_USE_PROXY       (1 <<  1)   /**< \brief Proxy enabled */
00282 #define FLASHROM_ISP_PULSE_DIAL      (1 <<  2)   /**< \brief Pulse dialing (instead of tone) */
00283 #define FLASHROM_ISP_BLIND_DIAL      (1 <<  3)   /**< \brief Blind dial (don't wait for tone) */
00284 /** @} */
00285
00286 /** \brief  ISP configuration structure.
00287
00288     This structure will be filled in by flashrom_get_ispcfg() (DreamPassport) or
00289     flashrom_get_pw_ispcfg() (PlanetWeb). Thanks to Sam Steele for the
00290     information about DreamPassport's ISP settings.
00291
00292     \headerfile dc/flashrom.h
00293 */
00294 typedef struct flashrom_ispcfg {
00295     int     method;         /**< \brief DHCP, Static, dialup(?), PPPoE
00296                                  \see   fr_method */
00297     uint32  valid_fields;   /**< \brief Which fields are valid?
00298                                  \see   fr_fields */
00299     uint32  flags;          /**< \brief Various flags that can be set in options
00300                                  \see   fr_flags */
00301
00302     uint8   ip[4];          /**< \brief Host IP address */
00303     uint8   nm[4];          /**< \brief Netmask */
00304     uint8   bc[4];          /**< \brief Broadcast address */
00305     uint8   gw[4];          /**< \brief Gateway address */
00306     uint8   dns[2][4];      /**< \brief DNS servers (2) */
00307     int     proxy_port;     /**< \brief Proxy server port */
00308     char    hostname[24];   /**< \brief DHCP/Host name */
00309     char    email[64];      /**< \brief Email address */
00310     char    smtp[31];       /**< \brief SMTP server */
00311     char    pop3[31];       /**< \brief POP3 server */
00312     char    pop3_login[20]; /**< \brief POP3 login */
```

```
00313     char    pop3_passwd[32];/**< \brief POP3 passwd */
00314     char    proxy_host[31]; /**< \brief Proxy server hostname */
00315     char    ppp_login[29];  /**< \brief PPP login */
00316     char    ppp_passwd[20]; /**< \brief PPP password */
00317     char    out_prefix[9];  /**< \brief Outside dial prefix */
00318     char    cw_prefix[9];   /**< \brief Call waiting prefix */
00319     char    real_name[31];  /**< \brief The "Real Name" field of PlanetWeb */
00320     char    modem_init[33]; /**< \brief The modem init string to use */
00321     char    area_code[4];   /**< \brief The area code the user is in */
00322     char    ld_prefix[21];  /**< \brief The long-distance dial prefix */
00323     char    p1_areacode[4]; /**< \brief Phone number 1's area code */
00324     char    phone1[26];     /**< \brief Phone number 1 */
00325     char    p2_areacode[4]; /**< \brief Phone number 2's area code */
00326     char    phone2[26];     /**< \brief Phone number 2 */
00327 } flashrom_ispcfg_t;
00328
00329 /** \brief  Retrieve DreamPassport's ISP configuration.
00330
00331     This function retrieves the console's ISP settings as set by DreamPassport,
00332     if they exist. You should check the valid_fields bitfield for the part of
00333     the struct you want before relying on the data.
00334
00335     \param  out             Storage for the structure.
00336     \retval 0               On success.
00337     \retval -1              On error (no settings found, or other errors).
00338 */
00339 int flashrom_get_ispcfg(flashrom_ispcfg_t * out);
00340
00341 /** \brief  Retrieve PlanetWeb's ISP configuration.
00342
00343     This function retrieves the console's ISP settings as set by PlanetWeb (1.0
00344     and 2.1 have been verified to work), if they exist. You should check the
00345     valid_fields bitfield for the part of the struct you want before relying on
00346     the data.
00347
00348     \param  out             Storage for the structure.
00349     \retval 0               On success.
00350     \retval -1              On error (i.e, no PlanetWeb settings found).
00351 */
00352 int flashrom_get_pw_ispcfg(flashrom_ispcfg_t *out);
00353
00354 /* More to come later */
00355
00356 __END_DECLS
00357
00358 #endif  /* __DC_FLASHROM_H */
```

9.176 kernel/arch/dreamcast/include/dc/fmath.h File Reference

Inline functions for the DC's special math instructions.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <dc/fmath_base.h>
```

Macros

- #define __FMINLINE static

Functions

- __FMINLINE float fipr (float x, float y, float z, float w, float a, float b, float c, float d)

  Floating point inner product.
- __FMINLINE float fipr_magnitude_sqr (float x, float y, float z, float w)

Floating point inner product w/self (square of vector magnitude)

- __FMINLINE float fsin (float r)

  Floating point sine.
- __FMINLINE float fcos (float r)

  Floating point cosine.
- __FMINLINE float ftan (float r)

  Floating point tangent.
- __FMINLINE float fisin (int d)

  Integer sine.
- __FMINLINE float ficos (int d)

  Integer cosine.
- __FMINLINE float fitan (int d)

  Integer tangent.
- __FMINLINE float fsqrt (float f)

  Floating point square root.
- __FMINLINE float frsqrt (float f)
- __FMINLINE void fsincos (float f, float *s, float *c)

  Calculate the sine and cosine of a value in degrees.
- __FMINLINE void fsincosr (float f, float *s, float *c)

  Calculate the sine and cosine of a value in radians.
- __FMINLINE uint32 pvr_pack_bump (float h, float t, float q)

  Calculate the offset color value for a set of bumpmap parameters.

9.176.1 Detailed Description

Inline functions for the DC's special math instructions.

Author

Andrew Kieschnick

Lawrence Sebald

9.176.2 Macro Definition Documentation

__FMINLINE

```
#define __FMINLINE static
```

9.176.3 Function Documentation

fcos()

```
__FMINLINE float fcos (
            float r )
```

Floating point cosine.

Parameters

r	a floating point number between 0 and 2∗PI

Returns

cos(r), where r is [0..2∗PI]

ficos()

```
__FMINLINE float ficos (
            int d )
```

Integer cosine.

Parameters

d	an integer between 0 and 65535

Returns

cos(d), where d is [0..65535]

fipr()

```
__FMINLINE float fipr (
            float x,
            float y,
            float z,
            float w,
            float a,
            float b,
            float c,
            float d )
```

Floating point inner product.

Returns

v1 dot v2 (inner product)

fipr_magnitude_sqr()

```
__FMINLINE float fipr_magnitude_sqr (
             float x,
             float y,
             float z,
             float w )
```

Floating point inner product w/self (square of vector magnitude)

Returns

v1 dot v1 (square of magnitude)

fisin()

```
__FMINLINE float fisin (
             int d )
```

Integer sine.

Parameters

d	an integer between 0 and 65535

Returns

sin(d), where d is [0..65535]

fitan()

```
__FMINLINE float fitan (
             int d )
```

Integer tangent.

Parameters

d	an integer between 0 and 65535

Returns

tan(d), where d is [0..65535]

frsqrt()

```
__FMINLINE float frsqrt (
            float f )
```

Returns

1.0f / sqrt(f)

fsin()

```
__FMINLINE float fsin (
            float r )
```

Floating point sine.

Parameters

r	a floating point number between 0 and 2∗PI

Returns

sin(r), where r is [0..2∗PI]

fsincos()

```
__FMINLINE void fsincos (
            float f,
            float * s,
            float * c )
```

Calculate the sine and cosine of a value in degrees.

This function uses the fsca instruction to calculate an approximation of the sine and cosine of the input value.

Parameters

f	The value to calculate the sine and cosine of.
s	Storage for the returned sine value.
c	Storage for the returned cosine value.

fsincosr()

```
__FMINLINE void fsincosr (
```

```
            float f,
            float * s,
            float * c )
```

Calculate the sine and cosine of a value in radians.

This function uses the fsca instruction to calculate an approximation of the sine and cosine of the input value.

Parameters

f	The value to calculate the sine and cosine of.
s	Storage for the returned sine value.
c	Storage for the returned cosine value.

fsqrt()

```
__FMINLINE float fsqrt (
            float f )
```

Floating point square root.

Returns

sqrt(f)

ftan()

```
__FMINLINE float ftan (
            float r )
```

Floating point tangent.

Parameters

r	a floating point number between 0 and 2∗PI

Returns

tan(r), where r is [0..2∗PI]

pvr_pack_bump()

```
__FMINLINE uint32 pvr_pack_bump (
            float h,
```

```
            float t,
            float q )
```

Calculate the offset color value for a set of bumpmap parameters.

This function calculates the value to be placed into the oargb value for the use of bumpmapping on a polygon. The angles specified should be expressed in radians and within the limits specified for the individual parameter.

Parameters

h	Weighting value in the range [0, 1] for how defined the bumpiness of the surface should be.
t	Spherical elevation angle in the range [0, pi/2] between the surface and the lighting source. A value of pi/2 implies that the light is directly overhead.
q	Spherical rotation angle in the range [0, 2∗pi] between the surface and the lighting source.

Returns

32-bit packed value to be used as an offset color on the surface to be bump mapped.

Note

For more information about how bumpmapping on the PVR works, refer to US Patent 6,819,319, which describes the algorithm implemented in the hardware (specifically look at Figures 2 and 3, along with the description in the Detailed Description section).

Thanks to Fredrik Ehnbom for figuring this stuff out and posting it to the mailing list back in 2005!

References F_PI.

9.177 fmath.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/fmath.h
00004    Copyright (C) 2001 Andrew Kieschnick
00005    Copyright (C) 2013, 2014 Lawrence Sebald
00006
00007 */
00008
00009 #ifndef __DC_FMATH_H
00010 #define __DC_FMATH_H
00011
00012 #include <sys/cdefs.h>
00013 __BEGIN_DECLS
00014
00015 #include <arch/types.h>
00016 #include <dc/fmath_base.h>
00017
00018 /**
00019     \file   dc/fmath.h
00020     \brief  Inline functions for the DC's special math instructions
00021     \author Andrew Kieschnick
00022     \author Lawrence Sebald
00023 */
00024
00025 /* Sigh... C99 treats inline stuff a lot differently than traditional GCC did,
00026    so we need to take care of that... */
00027 #if __STDC_VERSION__ >= 199901L
```

```
00028 #define __FMINLINE static inline
00029 #elif __GNUC__
00030 #define __FMINLINE extern inline
00031 #else
00032 /* Uhm... I guess this is the best we can do? */
00033 #define __FMINLINE static
00034 #endif
00035
00036 /**
00037     \brief  Floating point inner product.
00038     \return v1 dot v2 (inner product)
00039 */
00040 __FMINLINE float fipr(float x, float y, float z, float w,
00041                       float a, float b, float c, float d) {
00042     return __fipr(x, y, z, w, a, b, c, d);
00043 }
00044
00045 /**
00046     \brief  Floating point inner product w/self (square of vector magnitude)
00047     \return v1 dot v1 (square of magnitude)
00048 */
00049 __FMINLINE float fipr_magnitude_sqr(float x, float y, float z, float w) {
00050     return __fipr_magnitude_sqr(x, y, z, w);
00051 }
00052
00053 /**
00054     \brief Floating point sine
00055     \param r a floating point number between 0 and 2*PI
00056     \return sin(r), where r is [0..2*PI]
00057 */
00058 __FMINLINE float fsin(float r) {
00059     return __fsin(r);
00060 }
00061
00062 /**
00063     \brief Floating point cosine
00064     \param r a floating point number between 0 and 2*PI
00065     \return cos(r), where r is [0..2*PI]
00066 */
00067 __FMINLINE float fcos(float r) {
00068     return __fcos(r);
00069 }
00070
00071 /**
00072     \brief Floating point tangent
00073     \param r a floating point number between 0 and 2*PI
00074     \return tan(r), where r is [0..2*PI]
00075 */
00076 __FMINLINE float ftan(float r) {
00077     return __ftan(r);
00078 }
00079
00080 /**
00081     \brief Integer sine
00082     \param d an integer between 0 and 65535
00083     \return sin(d), where d is [0..65535]
00084 */
00085 __FMINLINE float fisin(int d) {
00086     return __fisin(d);
00087 }
00088
00089 /**
00090     \brief Integer cosine
00091     \param d an integer between 0 and 65535
00092     \return cos(d), where d is [0..65535]
00093 */
00094 __FMINLINE float ficos(int d) {
00095     return __ficos(d);
00096 }
00097
00098 /**
00099     \brief Integer tangent
00100     \param d an integer between 0 and 65535
00101     \return tan(d), where d is [0..65535]
00102 */
00103 __FMINLINE float fitan(int d) {
00104     return __fitan(d);
00105 }
00106
00107 /**
00108     \brief Floating point square root
```

```
00109     \return sqrt(f)
00110 */
00111 __FMINLINE float fsqrt(float f) {
00112     return __fsqrt(f);
00113 }
00114
00115 /**
00116     \return 1.0f / sqrt(f)
00117 */
00118 __FMINLINE float frsqrt(float f) {
00119     return __frsqrt(f);
00120 }
00121
00122 /** \brief  Calculate the sine and cosine of a value in degrees.
00123
00124     This function uses the fsca instruction to calculate an approximation of the
00125     sine and cosine of the input value.
00126
00127     \param  f               The value to calculate the sine and cosine of.
00128     \param  s               Storage for the returned sine value.
00129     \param  c               Storage for the returned cosine value.
00130 */
00131 __FMINLINE void fsincos(float f, float *s, float *c) {
00132     __fsincos(f, *s, *c);
00133 }
00134
00135 /** \brief  Calculate the sine and cosine of a value in radians.
00136
00137     This function uses the fsca instruction to calculate an approximation of the
00138     sine and cosine of the input value.
00139
00140     \param  f               The value to calculate the sine and cosine of.
00141     \param  s               Storage for the returned sine value.
00142     \param  c               Storage for the returned cosine value.
00143 */
00144 __FMINLINE void fsincosr(float f, float *s, float *c) {
00145     __fsincosr(f, *s, *c);
00146 }
00147
00148 /** \brief  Calculate the offset color value for a set of bumpmap parameters.
00149
00150     This function calculates the value to be placed into the oargb value for the
00151     use of bumpmapping on a polygon. The angles specified should be expressed in
00152     radians and within the limits specified for the individual parameter.
00153
00154     \param  h               Weighting value in the range [0, 1] for how defined
00155                             the bumpiness of the surface should be.
00156     \param  t               Spherical elevation angle in the range [0, pi/2]
00157                             between the surface and the lighting source. A value
00158                             of pi/2 implies that the light is directly overhead.
00159     \param  q               Spherical rotation angle in the range [0, 2*pi]
00160                             between the surface and the lighting source.
00161     \return                 32-bit packed value to be used as an offset color on
00162                             the surface to be bump mapped.
00163
00164     \note   For more information about how bumpmapping on the PVR works, refer
00165             to <a href="https://google.com/patents/US6819319">US Patent
00166             6,819,319</a>, which describes the algorithm implemented in the
00167             hardware (specifically look at Figures 2 and 3, along with the
00168             description in the Detailed Description section).
00169     \note   Thanks to Fredrik Ehnbom for figuring this stuff out and posting it
00170             to the mailing list back in 2005!
00171 */
00172 __FMINLINE uint32 pvr_pack_bump(float h, float t, float q) {
00173     uint8 hp = (uint8)(h * 255.0f);
00174     uint8 k1 = ~hp;
00175     uint8 k2 = (uint8)(hp * __fsin(t));
00176     uint8 k3 = (uint8)(hp * __fcos(t));
00177     uint8 qp = (uint8)((q / (2 * F_PI)) * 255.0f);
00178
00179
00180     return (k1 « 24) | (k2 « 16) | (k3 « 8) | qp;
00181 }
00182
00183 /* Make sure we declare the non-inline versions for C99 and non-gcc. Why they'd
00184    ever be needed, since they're inlined above, who knows? I guess in case
00185    someone tries to take the address of one of them? */
00186 /** \cond */
00187 #if __STDC_VERSION__ >= 199901L || !defined(__GNUC__)
00188 extern float fipr(float x, float y, float z, float w, float a, float b, float c,
00189                   float d);
```

```
00190 extern float fipr_magnitude_sqr(float x, float y, float z, float w);
00191 extern float fsin(float r);
00192 extern float fcos(float r);
00193 extern float ftan(float r);
00194 extern float fisin(int d);
00195 extern float ficos(int d);
00196 extern float fitan(int d);
00197 extern float fsqrt(float f);
00198 extern float frsqrt(float f);
00199 extern void fsincos(float f, float *s, float *c);
00200 extern void fsincosr(float f, float *s, float *c);
00201 #endif /* __STDC_VERSION__ >= 199901L || !defined(__GNUC__) */
00202 /** \endcond */
00203
00204 __END_DECLS
00205
00206 #endif  /* __DC_FMATH_H */
```

9.178 kernel/arch/dreamcast/include/dc/fmath_base.h File Reference

Base definitions for the DC's special math instructions.

```
#include <sys/cdefs.h>
```

Macros

- #define F_PI 3.1415926f

9.178.1 Detailed Description

Base definitions for the DC's special math instructions.

Author

Andrew Kieschnick

Josh Pearson

9.178.2 Macro Definition Documentation

F_PI

```
#define F_PI 3.1415926f
```

PI constant (if you don't want full math.h)

9.179 fmath_base.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/fmath_base.h
00004    Copyright (C) 2001 Andrew Kieschnick
00005    Copyright (C) 2014 Josh Pearson
00006
00007 */
00008
00009 #ifndef __DC_FMATH_BASE_H
00010 #define __DC_FMATH_BASE_H
00011
00012 #include <sys/cdefs.h>
00013 __BEGIN_DECLS
00014
00015 /**
00016     \file dc/fmath_base.h
00017     \brief  Base definitions for the DC's special math instructions
00018     \author Andrew Kieschnick
00019     \author Josh Pearson
00020 */
00021
00022 /** PI constant (if you don't want full math.h) */
00023 #define F_PI 3.1415926f
00024
00025 /** \cond */
00026 #define __fsin(x) \
00027     ({ float __value, __arg = (x), __scale = 10430.37835; \
00028         __asm__("fmul   %2,%1\n\t" \
00029                 "ftrc   %1,fpul\n\t" \
00030                 "fsca   fpul,dr0\n\t" \
00031                 "fmov   fr0,%0" \
00032                 : "=f" (__value), "+&f" (__scale) \
00033                 : "f" (__arg) \
00034                 : "fpul", "fr0", "fr1"); \
00035         __value; })
00036
00037 #define __fcos(x) \
00038     ({ float __value, __arg = (x), __scale = 10430.37835; \
00039         __asm__("fmul   %2,%1\n\t" \
00040                 "ftrc   %1,fpul\n\t" \
00041                 "fsca   fpul,dr0\n\t" \
00042                 "fmov   fr1,%0" \
00043                 : "=f" (__value), "+&f" (__scale) \
00044                 : "f" (__arg) \
00045                 : "fpul", "fr0", "fr1"); \
00046         __value; })
00047
00048 #define __ftan(x) \
00049     ({ float __value, __arg = (x), __scale = 10430.37835; \
00050         __asm__("fmul   %2,%1\n\t" \
00051                 "ftrc   %1,fpul\n\t" \
00052                 "fsca   fpul,dr0\n\t" \
00053                 "fdiv   fr1, fr0\n\t" \
00054                 "fmov   fr0,%0" \
00055                 : "=f" (__value), "+&f" (__scale) \
00056                 : "f" (__arg) \
00057                 : "fpul", "fr0", "fr1"); \
00058         __value; })
00059
00060
00061 #define __fisin(x) \
00062     ({ float __value, __arg = (x); \
00063         __asm__("lds    %1,fpul\n\t" \
00064                 "fsca   fpul,dr0\n\t" \
00065                 "fmov   fr0,%0" \
00066                 : "=f" (__value) \
00067                 : "r" (__arg) \
00068                 : "fpul", "fr0", "fr1"); \
00069         __value; })
00070
00071 #define __ficos(x) \
00072     ({ float __value, __arg = (x); \
00073         __asm__("lds    %1,fpul\n\t" \
00074                 "fsca   fpul,dr0\n\t" \
00075                 "fmov   fr1,%0" \
00076                 : "=f" (__value) \
```

```
00077                 : "r" (__arg) \
00078                 : "fpul", "fr0", "fr1"); \
00079         __value; })
00080
00081 #define __fitan(x) \
00082     ({ float __value, __arg = (x); \
00083         __asm__("lds    %1,fpul\n\t" \
00084                 "fsca   fpul,dr0\n\t" \
00085                 "fdiv   fr1, fr0\n\t" \
00086                 "fmov   fr0,%0" \
00087                 : "=f" (__value) \
00088                 : "r" (__arg) \
00089                 : "fpul", "fr0", "fr1"); \
00090         __value; })
00091
00092 #define __fsincos(r, s, c) \
00093     ({  register float __r __asm__("fr10") = r; \
00094         register float __a __asm__("fr11") = 182.04444443; \
00095         __asm__("fmul fr11, fr10\n\t" \
00096                 "ftrc fr10, fpul\n\t" \
00097                 "fsca fpul, dr10\n\t" \
00098                 : "+f" (__r), "+f" (__a) \
00099                 : "0" (__r), "1" (__a) \
00100                 : "fpul"); \
00101         s = __r; c = __a; })
00102
00103 #define __fsincosr(r, s, c) \
00104     ({  register float __r __asm__("fr10") = r; \
00105         register float __a __asm__("fr11") = 10430.37835; \
00106         __asm__("fmul fr11, fr10\n\t" \
00107                 "ftrc fr10, fpul\n\t" \
00108                 "fsca fpul, dr10\n\t" \
00109                 : "+f" (__r), "+f" (__a) \
00110                 : "0" (__r), "1" (__a) \
00111                 : "fpul"); \
00112         s = __r; c = __a; })
00113
00114 #define __fsqrt(x) \
00115     ({ float __arg = (x); \
00116         __asm__("fsqrt %0\n\t" \
00117                 : "=f" (__arg) : "0" (__arg)); \
00118         __arg; })
00119
00120 #define __frsqrt(x) \
00121     ({ float __arg = (x); \
00122         __asm__("fsrra %0\n\t" \
00123                 : "=f" (__arg) : "0" (__arg)); \
00124         __arg; })
00125
00126 /* Floating point inner product (dot product) */
00127 #define __fipr(x, y, z, w, a, b, c, d) ({ \
00128         register float __x __asm__("fr5") = (x); \
00129         register float __y __asm__("fr4") = (y); \
00130         register float __z __asm__("fr7") = (z); \
00131         register float __w __asm__("fr6") = (w); \
00132         register float __a __asm__("fr9") = (a); \
00133         register float __b __asm__("fr8") = (b); \
00134         register float __c __asm__("fr11") = (c); \
00135         register float __d __asm__("fr10") = (d); \
00136         __asm__ __volatile__( \
00137                                 "fipr     fv8,fv4" \
00138                                 : "+f" (__z) \
00139                                 : "f" (__x), "f" (__y), "f" (__z), "f" (__w), \
00140                                 "f" (__a), "f" (__b), "f" (__c), "f" (__d) \
00141                               ); \
00142         __z; })
00143
00144 /* Floating point inner product w/self (square of vector magnitude) */
00145 #define __fipr_magnitude_sqr(x, y, z, w) ({ \
00146         register float __x __asm__("fr5") = (x); \
00147         register float __y __asm__("fr4") = (y); \
00148         register float __z __asm__("fr7") = (z); \
00149         register float __w __asm__("fr6") = (w); \
00150         __asm__ __volatile__( \
00151                                 "fipr     fv4,fv4" \
00152                                 : "+f" (__z) \
00153                                 : "f" (__x), "f" (__y), "f" (__z), "f" (__w) \
00154                               ); \
00155         __z; })
00156
00157 /** \endcond */
```

```
00158 __END_DECLS
00159
00160 #endif /* !__DC_FMATH_BASE_H */
```

9.180 kernel/arch/dreamcast/include/dc/fs_dcload.h File Reference

Implementation of dcload "filesystem".

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <kos/fs.h>
#include <kos/dbgio.h>
```

Macros

- #define DCLOADMAGICVALUE 0xdeadbeef

  The dcload magic value!
- #define DCLOADMAGICADDR (unsigned int *)0x8c004004

  The address of the dcload magic value.
- #define DCLOAD_TYPE_NONE -1

  No dcload connection.
- #define DCLOAD_TYPE_SER 0

  dcload-serial connection
- #define DCLOAD_TYPE_IP 1

  dcload-ip connection

Variables

- int dcload_type

  What type of dcload connection do we have?

9.180.1 Detailed Description

Implementation of dcload "filesystem".

This file contains declarations related to using dcload, both in its -ip and -serial forms. This is only used for dcload-ip support if the internal network stack is not initialized at start via KOS_INIT_FLAGS().

Author

Andrew Kieschnick

See also

dc/fs_dclsocket.h

9.180.2 Macro Definition Documentation

DCLOAD_TYPE_IP

```
#define DCLOAD_TYPE_IP 1
```

dcload-ip connection

DCLOAD_TYPE_NONE

```
#define DCLOAD_TYPE_NONE -1
```

No dcload connection.

DCLOAD_TYPE_SER

```
#define DCLOAD_TYPE_SER 0
```

dcload-serial connection

DCLOADMAGICADDR

```
#define DCLOADMAGICADDR (unsigned int *)0x8c004004
```

The address of the dcload magic value.

DCLOADMAGICVALUE

```
#define DCLOADMAGICVALUE 0xdeadbeef
```

The dcload magic value!

9.180.3 Variable Documentation

dcload_type

```
int dcload_type  [extern]
```

What type of dcload connection do we have?

9.181 fs_dcload.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kernel/arch/dreamcast/include/dc/fs_dcload.h
00004    (c)2002 Andrew Kieschnick
00005
00006 */
00007
00008 /** \file   dc/fs_dcload.h
00009     \brief  Implementation of dcload "filesystem".
00010
00011     This file contains declarations related to using dcload, both in its -ip and
00012     -serial forms. This is only used for dcload-ip support if the internal
00013     network stack is not initialized at start via KOS_INIT_FLAGS().
00014
00015     \author Andrew Kieschnick
00016     \see    dc/fs_dclsocket.h
00017 */
00018
00019 #ifndef __DC_FS_DCLOAD_H
00020 #define __DC_FS_DCLOAD_H
00021
00022 /* Definitions for the "dcload" file system */
00023
00024 #include <sys/cdefs.h>
00025 __BEGIN_DECLS
00026
00027 #include <arch/types.h>
00028 #include <kos/fs.h>
00029 #include <kos/dbgio.h>
00030
00031 /* \cond */
00032 extern dbgio_handler_t dbgio_dcload;
00033 /* \endcond */
00034
00035 /* dcload magic value */
00036 /** \brief  The dcload magic value! */
00037 #define DCLOADMAGICVALUE 0xdeadbeef
00038
00039 /** \brief  The address of the dcload magic value */
00040 #define DCLOADMAGICADDR (unsigned int *)0x8c004004
00041
00042 /* Are we using dc-load-serial or dc-load-ip? */
00043 #define DCLOAD_TYPE_NONE    -1      /**< \brief No dcload connection */
00044 #define DCLOAD_TYPE_SER     0       /**< \brief dcload-serial connection */
00045 #define DCLOAD_TYPE_IP      1       /**< \brief dcload-ip connection */
00046
00047 /** \brief  What type of dcload connection do we have? */
00048 extern int dcload_type;
00049
00050 /* \cond */
00051 /* Available dcload console commands */
00052
00053 #define DCLOAD_READ 0
00054 #define DCLOAD_WRITE 1
00055 #define DCLOAD_OPEN 2
00056 #define DCLOAD_CLOSE 3
00057 #define DCLOAD_CREAT 4
00058 #define DCLOAD_LINK 5
00059 #define DCLOAD_UNLINK 6
00060 #define DCLOAD_CHDIR 7
00061 #define DCLOAD_CHMOD 8
00062 #define DCLOAD_LSEEK 9
00063 #define DCLOAD_FSTAT 10
00064 #define DCLOAD_TIME 11
00065 #define DCLOAD_STAT 12
00066 #define DCLOAD_UTIME 13
00067 #define DCLOAD_ASSIGNWRKMEM 14
00068 #define DCLOAD_EXIT 15
00069 #define DCLOAD_OPENDIR 16
00070 #define DCLOAD_CLOSEDIR 17
00071 #define DCLOAD_READDIR 18
00072 #define DCLOAD_GETHOSTINFO 19
00073 #define DCLOAD_GDBPACKET 20
00074
00075 /* dcload syscall function */
00076
```

```
00077 int dcloadsyscall(unsigned int syscall, ...);
00078
00079 /* dcload dirent */
00080
00081 struct dcload_dirent {
00082     long            d_ino;  /* inode number */
00083     off_t           d_off;  /* offset to the next dirent */
00084     unsigned short  d_reclen;/* length of this record */
00085     unsigned char   d_type;         /* type of file */
00086     char            d_name[256];    /* filename */
00087 };
00088
00089 typedef struct dcload_dirent dcload_dirent_t;
00090
00091 /* dcload stat */
00092
00093 struct  dcload_stat {
00094     unsigned short st_dev;
00095     unsigned short st_ino;
00096     int st_mode;
00097     unsigned short st_nlink;
00098     unsigned short st_uid;
00099     unsigned short st_gid;
00100     unsigned short st_rdev;
00101     long st_size;
00102     long atime;
00103     long st_spare1;
00104     long mtime;
00105     long st_spare2;
00106     long ctime;
00107     long st_spare3;
00108     long st_blksize;
00109     long st_blocks;
00110     long st_spare4[2];
00111 };
00112
00113 typedef struct dcload_stat dcload_stat_t;
00114
00115 /* Printk replacement */
00116 void dcload_printk(const char *str);
00117
00118 /* GDB tunnel */
00119 size_t dcload_gdbpacket(const char* in_buf, size_t in_size, char* out_buf, size_t out_size);
00120
00121 /* File functions */
00122 void*   dcload_open(vfs_handler_t * vfs, const char *fn, int mode);
00123 int     dcload_close(void * hnd);
00124 ssize_t dcload_read(void * hnd, void *buf, size_t cnt);
00125 off_t   dcload_seek(void * hnd, off_t offset, int whence);
00126 off_t   dcload_tell(void * hnd);
00127 size_t  dcload_total(void * hnd);
00128 dirent_t* dcload_readdir(void * hnd);
00129 int     dcload_rename(vfs_handler_t * vfs, const char *fn1, const char *fn2);
00130 int     dcload_unlink(vfs_handler_t * vfs, const char *fn);
00131
00132 /* Init func */
00133 void fs_dcload_init_console(void);
00134 int fs_dcload_init(void);
00135 int fs_dcload_shutdown(void);
00136
00137 /* Init func for dcload-ip + lwIP */
00138 int fs_dcload_init_lwip(void *p);
00139
00140 /* \endcond */
00141
00142 __END_DECLS
00143
00144 #endif  /* __DC_FS_DCLOAD_H */
```

9.182 kernel/arch/dreamcast/include/dc/fs_dclsocket.h File Reference

Implementation of dcload-ip over KOS sockets.

```
#include <sys/cdefs.h>
#include <dc/fs_dcload.h>
```

9.182.1 Detailed Description

Implementation of dcload-ip over KOS sockets.

This file contains declarations related to using dcload-ip over the KOS sockets system. If dcload-ip support is enabled at the same time as network support, this is how the communications will happen. There isn't really anything that users will need to deal with in here.

Author

Lawrence Sebald

9.183 fs_dclsocket.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/fs_dclsocket.h
00004    Copyright (C) 2007, 2008 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   dc/fs_dclsocket.h
00009     \brief  Implementation of dcload-ip over KOS sockets.
00010
00011     This file contains declarations related to using dcload-ip over the KOS
00012     sockets system. If dcload-ip support is enabled at the same time as network
00013     support, this is how the communications will happen. There isn't really
00014     anything that users will need to deal with in here.
00015
00016     \author Lawrence Sebald
00017 */
00018
00019 #ifndef __DC_FSDCLSOCKET_H
00020 #define __DC_FSDCLSOCKET_H
00021
00022 #include <sys/cdefs.h>
00023 __BEGIN_DECLS
00024
00025 #include <dc/fs_dcload.h>
00026
00027 /* \cond */
00028 extern dbgio_handler_t dbgio_dcls;
00029
00030 /* Initialization */
00031 void fs_dclsocket_init_console(void);
00032 int fs_dclsocket_init(void);
00033
00034 int fs_dclsocket_shutdown(void);
00035 /* \endcond */
00036
00037 __END_DECLS
00038
00039 #endif /* __DC_FSDCLSOCKET_H */
```

9.184 kernel/arch/dreamcast/include/dc/fs_iso9660.h File Reference

ISO9660 (CD-ROM) filesystem driver.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <kos/limits.h>
#include <kos/fs.h>
```

Functions

- int iso_reset (void)

  Reset the internal ISO9660 cache.

9.184.1 Detailed Description

ISO9660 (CD-ROM) filesystem driver.

This driver implements support for reading files from a CD-ROM or CD-R in the Dreamcast's disc drive. This filesystem mounts itself on /cd.

This driver supports Rock Ridge, thanks to Andrew Kieschnick. The driver also supports the Joliet extensions thanks to Bero.

The implementation was originally based on a simple ISO9660 implementation by Marcus Comstedt.

Author

Megan Potter

Andrew Kieschnick

Bero

9.184.2 Function Documentation

iso_reset()

```
int iso_reset (
            void  )
```

Reset the internal ISO9660 cache.

This function resets the cache of the ISO9660 driver, breaking connections to all files. This generally assumes that a new disc has been or will be inserted.

Return values

0	On success.

9.185 fs_iso9660.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/fs/iso9660.h
00004    (c)2000-2001 Megan Potter
```

```
00005
00006 */
00007
00008 /** \file   dc/fs_iso9660.h
00009     \brief  ISO9660 (CD-ROM) filesystem driver.
00010
00011     This driver implements support for reading files from a CD-ROM or CD-R in
00012     the Dreamcast's disc drive. This filesystem mounts itself on /cd.
00013
00014     This driver supports Rock Ridge, thanks to Andrew Kieschnick. The driver
00015     also supports the Joliet extensions thanks to Bero.
00016
00017     The implementation was originally based on a simple ISO9660 implementation
00018     by Marcus Comstedt.
00019
00020     \author Megan Potter
00021     \author Andrew Kieschnick
00022     \author Bero
00023 */
00024
00025 #ifndef __DC_FS_ISO9660_H
00026 #define __DC_FS_ISO9660_H
00027
00028 #include <sys/cdefs.h>
00029 __BEGIN_DECLS
00030
00031 #include <arch/types.h>
00032 #include <kos/limits.h>
00033 #include <kos/fs.h>
00034
00035 /** \brief  Reset the internal ISO9660 cache.
00036
00037     This function resets the cache of the ISO9660 driver, breaking connections
00038     to all files. This generally assumes that a new disc has been or will be
00039     inserted.
00040
00041     \retval 0               On success.
00042 */
00043 int iso_reset(void);
00044
00045 /* \cond */
00046 int fs_iso9660_init(void);
00047 int fs_iso9660_shutdown(void);
00048 /* \endcond */
00049
00050 __END_DECLS
00051
00052 #endif  /* __DC_FS_ISO9660_H */
00053
```

9.186 kernel/arch/dreamcast/include/dc/fs_vmu.h File Reference

VMU filesystem driver.

```
#include <sys/cdefs.h>
#include <kos/fs.h>
```

9.186.1 Detailed Description

VMU filesystem driver.

The VMU filesystem driver mounts itself on /vmu of the VFS. Each memory card has its own subdirectory off of that directory (i.e, /vmu/a1 for slot 1 of the first controller). VMUs themselves have no subdirectories, so the driver itself is fairly simple.

Files on a VMU must be multiples of 512 bytes in size, and should have a header attached so that they show up in the BIOS menu.

This layer is built off of the vmufs layer, which does all the low-level operations. It is generally easier to work with things at this level though, so that you can use the normal libc file access functions.

Author

Megan Potter

See also

dc/vmu_pkg.h

dc/vmufs.h

9.187 fs_vmu.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/fs_vmu.h
00004    (c)2000-2001 Jordan DeLong
00005
00006 */
00007
00008 /** \file   dc/fs_vmu.h
00009     \brief  VMU filesystem driver.
00010
00011     The VMU filesystem driver mounts itself on /vmu of the VFS. Each memory card
00012     has its own subdirectory off of that directory (i.e, /vmu/a1 for slot 1 of
00013     the first controller). VMUs themselves have no subdirectories, so the driver
00014     itself is fairly simple.
00015
00016     Files on a VMU must be multiples of 512 bytes in size, and should have a
00017     header attached so that they show up in the BIOS menu.
00018
00019     This layer is built off of the vmufs layer, which does all the low-level
00020     operations. It is generally easier to work with things at this level though,
00021     so that you can use the normal libc file access functions.
00022
00023     \author Megan Potter
00024     \see    dc/vmu_pkg.h
00025     \see    dc/vmufs.h
00026 */
00027
00028 #ifndef __DC_FS_VMU_H
00029 #define __DC_FS_VMU_H
00030
00031 #include <sys/cdefs.h>
00032 __BEGIN_DECLS
00033
00034 #include <kos/fs.h>
00035
00036 /* \cond */
00037 /* Initialization */
00038 int fs_vmu_init(void);
00039 int fs_vmu_shutdown(void);
00040 /* \endcond */
00041
00042 __END_DECLS
00043
00044 #endif  /* __DC_FS_VMU_H */
00045
```

9.188 kernel/arch/dreamcast/include/dc/g1ata.h File Reference

G1 bus ATA interface.

```
#include <sys/cdefs.h>
#include <stdint.h>
#include <kos/blockdev.h>
```

Macros

- #define G1_ATA_MASTER 0x00

  ATA master device.
- #define G1_ATA_MASTER_ALT 0x90

  ATA master device (compatible with old drives).
- #define G1_ATA_SLAVE 0xB0

  ATA slave device.
- #define G1_ATA_LBA_MODE 0x40

  Select LBA addressing mode.

Functions

- int g1_dma_in_progress (void)

  Is there a G1 DMA in progress currently?
- int g1_ata_mutex_lock (void)

  Lock the G1 ATA mutex.
- int g1_ata_mutex_unlock (void)

  Unlock the G1 ATA mutex.
- uint8_t g1_ata_select_device (uint8_t dev)

  Set the active ATA device.
- int g1_ata_read_chs (uint16_t c, uint8_t h, uint8_t s, size_t count, void *buf)

  Read one or more disk sectors with Cylinder-Head-Sector addressing.
- int g1_ata_write_chs (uint16_t c, uint8_t h, uint8_t s, size_t count, const void *buf)

  Write one or more disk sectors with Cylinder-Head-Sector addressing.
- int g1_ata_read_lba (uint64_t sector, size_t count, void *buf)

  Read one or more disk sectors with Linear Block Addressing (LBA).
- int g1_ata_read_lba_dma (uint64_t sector, size_t count, void *buf, int block)

  DMA read disk sectors with Linear Block Addressing (LBA).
- int g1_ata_write_lba (uint64_t sector, size_t count, const void *buf)

  Write one or more disk sectors with Linear Block Addressing (LBA).
- int g1_ata_write_lba_dma (uint64_t sector, size_t count, const void *buf, int block)

  DMA Write disk sectors with Linear Block Addressing (LBA).
- int g1_ata_flush (void)

  Flush the write cache on the attached disk.
- int g1_ata_lba_mode (void)

  Get LBA mode of the attached disk.
- int g1_ata_blockdev_for_partition (int partition, int dma, kos_blockdev_t *rv, uint8_t *partition_type)

  Get a block device for a given partition on the slave ATA device.
- int g1_ata_blockdev_for_device (int dma, kos_blockdev_t *rv)

  Get a block device for the attached ATA device.
- int g1_ata_init (void)

  Initialize G1 ATA support.
- void g1_ata_shutdown (void)

  Shut down G1 ATA support.

9.188.1 Detailed Description

G1 bus ATA interface.

This file provides support for accessing an ATA device on the G1 bus in the Dreamcast. The G1 bus usually contains a few useful pieces of the system, including the flashrom and the GD-ROM drive. The interesting piece here is that the GD-ROM drive itself is actually an ATA device.

Luckily, Sega left everything in place to access both a master and slave device on this ATA port. The GD-ROM drive should always be the master device on the chain, but you can hook up a hard drive or some other device as a slave. The functions herein are for accessing just such a slave device.

Note

The functions herein do not provide for direct access to the GD-ROM drive. There is not really any sort of compelling reason to access the GD-ROM drive directly instead of using the system calls, so you should continue to use the normal cdrom_* functions for accessing the GD-ROM drive. Also, currently there is no locking done to prevent you from doing "bad things" with concurrent access on the bus, so be careful. ;)

Author

Lawrence Sebald

Ruslan Rostovtsev

9.188.2 Function Documentation

g1_ata_blockdev_for_device()

```
int g1_ata_blockdev_for_device (
            int dma,
            kos_blockdev_t * rv )
```

Get a block device for the attached ATA device.

This function creates a block device descriptor for the attached ATA device.

Parameters

dma	Set to 1 to use DMA for reads/writes on the device, if available.
rv	Used to return the block device. Must be non-NULL.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

ENXIO - ATA support not initialized or no device attached
EFAULT - rv was NULL
ENOMEM - out of memory

g1_ata_blockdev_for_partition()

```
int g1_ata_blockdev_for_partition (
            int partition,
            int dma,
            kos_blockdev_t * rv,
            uint8_t * partition_type )
```

Get a block device for a given partition on the slave ATA device.

This function creates a block device descriptor for the given partition on the attached ATA device. This block device is used to interface with various filesystems on the device.

Parameters

partition	The partition number (0-3) to use.
dma	Set to 1 to use DMA for reads/writes on the device, if available.
rv	Used to return the block device. Must be non-NULL.
partition_type	Used to return the partition type. Must be non-NULL.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

ENXIO - ATA support not initialized or no device attached
EIO - an I/O error occurred in reading data
EINVAL - invalid partition number was given
EFAULT - rv or partition_type was NULL
ENOENT - no MBR found
ENOENT - no partition at the specified position
ENOMEM - out of memory

Note

This interface currently only supports MBR-formatted disks. There is currently no support for GPT partition tables.

g1_ata_flush()

```
int g1_ata_flush (
            void  )
```

Flush the write cache on the attached disk.

This function flushes the write cache on the disk attached as the slave device on the G1 ATA bus. This ensures that all writes that have previously completed are fully persisted to the disk. You should do this before unmounting any disks or exiting your program if you have called any of the write functions in here.

Returns

0 on success. <0 on error, setting errno as appropriate.

Error Conditions:

ENXIO - ATA support not initialized or no device attached

g1_ata_init()

```
int g1_ata_init (
            void  )
```

Initialize G1 ATA support.

This function initializes the rest of this subsystem and completes a scan of the G1 ATA bus for devices. This function may take a while to complete with some devices. Currently only the slave device is scanned, as the master device should always be the GD-ROM drive.

Returns

0 on success, <0 on error or if no device is present

g1_ata_lba_mode()

```
int g1_ata_lba_mode (
            void  )
```

Get LBA mode of the attached disk.

Returns

-1 on error, 0 - CHS, 28 - LBA28, 48 - LBA48

Error Conditions:

ENXIO - ATA support not initialized or no device attached

g1_ata_mutex_lock()

```
int g1_ata_mutex_lock (
            void  )
```

Lock the G1 ATA mutex.

This function locks the mutex that arbitrates access to the ATA bus. You should never have to do this on your own unless you're accessing devices manually yourself.

Returns

0 on success, -1 on failure.

Note

Failure conditions are exactly the same as the mutex_lock() function.

g1_ata_mutex_unlock()

```
int g1_ata_mutex_unlock (
            void  )
```

Unlock the G1 ATA mutex.

This function unlocks the mutex that arbitrates access to the ATA bus. You should never have to do this on your own unless you're accessing devices manually yourself.

Returns

0 on success, -1 on failure.

Note

Failure conditions are exactly the same as the mutex_unlock() function.

g1_ata_read_chs()

```
int g1_ata_read_chs (
            uint16_t c,
            uint8_t h,
            uint8_t s,
            size_t count,
            void * buf )
```

Read one or more disk sectors with Cylinder-Head-Sector addressing.

This function reads one or more 512-byte disk blocks from the slave device on the G1 ATA bus using Cylinder-Head-↩ Sector addressing. This function uses PIO and blocks until the data is read in.

Parameters

c	The cylinder to start reading from.
h	The head to start reading from.
s	The sector to start reading from.
count	The number of disk sectors to read.
buf	Storage for the read-in disk sectors. This should be at least (count $*$ 512) bytes in length, and must be at least 16-bit aligned.

Returns

0 on success. $<$ 0 on failure, setting errno as appropriate.

Note

Unless you're accessing a really old hard drive, you probably do not want to use this function to access the disk. Use the g1_ata_read_lba() function instead of this one, unless you get an error from that function indicating that LBA addressing is not supported.

Error Conditions:

EIO - an I/O error occurred in reading data
ENXIO - ATA support not initialized or no device attached
EOVERFLOW - one or more of the requested sectors is out of the range of the disk

g1_ata_read_lba()

```
int g1_ata_read_lba (
            uint64_t sector,
            size_t count,
            void * buf )
```

Read one or more disk sectors with Linear Block Addressing (LBA).

This function reads one or more 512-byte disk blocks from the slave device on the G1 ATA bus using LBA mode (either 28 or 48 bits, as appropriate). This function uses PIO and blocks until the data is read.

Parameters

sector	The sector to start reading from.
count	The number of disk sectors to read.
buf	Storage for the read-in disk sectors. This should be at least (count $*$ 512) bytes in length, and must be at least 16-bit aligned.

Returns

0 on success. $<$ 0 on failure, setting errno as appropriate.

Note

If errno is set to ENOTSUP after calling this function, you must use the g1_ata_read_chs() function instead.

Error Conditions:

EIO - an I/O error occurred in reading data
ENXIO - ATA support not initialized or no device attached
EOVERFLOW - one or more of the requested sectors is out of the range of the disk
ENOTSUP - LBA mode not supported by the device

g1_ata_read_lba_dma()

```
int g1_ata_read_lba_dma (
            uint64_t sector,
            size_t count,
            void * buf,
            int block )
```

DMA read disk sectors with Linear Block Addressing (LBA).

This function reads one or more 512-byte disk blocks from the slave device on the G1 ATA bus using LBA mode (either 28 or 48 bits, as appropriate). This function uses DMA and optionally blocks until the data is read.

Parameters

sector	The sector to start reading from.
count	The number of disk sectors to read.
buf	Storage for the read-in disk sectors. This should be at least (count $*$ 512) bytes in length, and must be at least 32-byte aligned.
block	Non-zero to block until the transfer completes.

Returns

0 on success. $<$ 0 on failure, setting errno as appropriate.

Note

If errno is set to ENOTSUP after calling this function, you must use a CHS addressed transfer function instead, like g1_ata_read_chs().

If errno is set to EPERM after calling this function, DMA mode is not supported. You should use a PIO transfer function like g1_ata_read_lba() instead.

Error Conditions:

EIO - an I/O error occurred in reading data
ENXIO - ATA support not initialized or no device attached
EOVERFLOW - one or more of the requested sectors is out of the range of the disk
ENOTSUP - LBA mode not supported by the device
EPERM - device does not support DMA

g1_ata_select_device()

```
uint8_t g1_ata_select_device (
             uint8_t dev )
```

Set the active ATA device.

This function sets the device that any further ATA commands will go to. You shouldn't have to ever call this yourself, as it should be done for you by any of the access functions. This must be called with the ATA lock held.

Parameters

dev	The device to access (generally either G1_ATA_MASTER or G1_ATA_SLAVE).

Returns

The previous active device (or 0x0F if the function would block in an IRQ handler).

Note

This function may block if there is a transfer ongoing. If called in an IRQ handler and the call would otherwise block, 0x0F is returned.

g1_ata_shutdown()

```
void g1_ata_shutdown (
             void  )
```

Shut down G1 ATA support.

This function shuts down the rest of this subsystem, and attempts to flush the write cache of any attached slave devices. Accessing any ATA devices using this subsystem after this function is called may produce undefined results.

g1_ata_write_chs()

```
int g1_ata_write_chs (
            uint16_t c,
            uint8_t h,
            uint8_t s,
            size_t count,
            const void * buf )
```

Write one or more disk sectors with Cylinder-Head-Sector addressing.

This function writes one or more 512-byte disk blocks to the slave device on the G1 ATA bus using Cylinder-Head-Sector addressing. This function uses PIO and blocks until the data is written.

Parameters

c	The cylinder to start writing to.
h	The head to start writing to.
s	The sector to start writing to.
count	The number of disk sectors to write.
buf	The data to write to the disk. This should be (count $*$ 512) bytes in length and must be at least 16-bit aligned.

Returns

0 on success. $<$ 0 on failure, setting errno as appropriate.

Note

Unless you're accessing a really old hard drive, you probably do not want to use this function to access the disk. Use the g1_ata_write_lba() function instead of this one, unless you get an error from that function indicating that LBA addressing is not supported.

Error Conditions:

ENXIO - ATA support not initialized or no device attached
EOVERFLOW - one or more of the requested sectors is out of the range of the disk

g1_ata_write_lba()

```
int g1_ata_write_lba (
            uint64_t sector,
            size_t count,
            const void * buf )
```

Write one or more disk sectors with Linear Block Addressing (LBA).

This function writes one or more 512-byte disk blocks to the slave device on the G1 ATA bus using LBA mode (either 28 or 48 bits, as appropriate). This function uses PIO and blocks until the data is written.

Parameters

sector	The sector to start writing to.
count	The number of disk sectors to write.
buf	The data to write to the disk. This should be (count $*$ 512) bytes in length and must be at least 16-bit aligned.

Returns

0 on success. $<$ 0 on failure, setting errno as appropriate.

Note

If errno is set to ENOTSUP after calling this function, you must use the g1_ata_write_chs() function instead.

Error Conditions:

ENXIO - ATA support not initialized or no device attached
EOVERFLOW - one or more of the requested sectors is out of the range of the disk
ENOTSUP - LBA mode not supported by the device

g1_ata_write_lba_dma()

```
int g1_ata_write_lba_dma (
            uint64_t sector,
            size_t count,
            const void * buf,
            int block )
```

DMA Write disk sectors with Linear Block Addressing (LBA).

This function writes one or more 512-byte disk blocks to the slave device on the G1 ATA bus using LBA mode (either 28 or 48 bits, as appropriate). This function uses DMA and optionally blocks until the data is written.

Parameters

sector	The sector to start writing to.
count	The number of disk sectors to write.
buf	The data to write to the disk. This should be (count $*$ 512) bytes in length and must be at least 32-byte aligned.
block	Non-zero to block until the transfer completes.

Returns

0 on success. $<$ 0 on failure, setting errno as appropriate.

Note

If errno is set to ENOTSUP after calling this function, you must use the g1_ata_write_chs() function instead.

If errno is set to EPERM after calling this function, DMA mode is not supported. You should use a PIO transfer function like g1_ata_write_lba() instead.

Error Conditions:

ENXIO - ATA support not initialized or no device attached
EOVERFLOW - one or more of the requested sectors is out of the range of the disk
ENOTSUP - LBA mode not supported by the device
EPERM - device does not support DMA

g1_dma_in_progress()

```
int g1_dma_in_progress (
            void  )
```

Is there a G1 DMA in progress currently?

This function returns non-zero if a DMA is in progress. This can be used to check on the completion of DMA transfers when non-blocking mode was selected at transfer time.

Returns

0 if no DMA is in progress, nonzero otherwise.

9.189 g1ata.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/g1ata.h
00004    Copyright (C) 2013, 2014 Lawrence Sebald
00005    Copyright (C) 2023 Ruslan Rostovtsev
00006 */
00007
00008 /** \file   dc/g1ata.h
00009     \brief  G1 bus ATA interface.
00010
00011     This file provides support for accessing an ATA device on the G1 bus in the
00012     Dreamcast. The G1 bus usually contains a few useful pieces of the system,
00013     including the flashrom and the GD-ROM drive. The interesting piece here is
00014     that the GD-ROM drive itself is actually an ATA device.
00015
00016     Luckily, Sega left everything in place to access both a master and slave
00017     device on this ATA port. The GD-ROM drive should always be the master device
00018     on the chain, but you can hook up a hard drive or some other device as a
00019     slave. The functions herein are for accessing just such a slave device.
00020
00021     \note   The functions herein do not provide for direct access to the GD-ROM
00022             drive. There is not really any sort of compelling reason to access
00023             the GD-ROM drive directly instead of using the system calls, so you
00024             should continue to use the normal cdrom_* functions for accessing
00025             the GD-ROM drive. Also, currently there is no locking done to
00026             prevent you from doing "bad things" with concurrent access on the
00027             bus, so be careful. ;)
00028
00029     \author Lawrence Sebald
00030     \author Ruslan Rostovtsev
00031 */
00032
00033 #ifndef __DC_G1ATA_H
00034 #define __DC_G1ATA_H
00035
00036 #include <sys/cdefs.h>
00037 __BEGIN_DECLS
00038
00039 #include <stdint.h>
00040 #include <kos/blockdev.h>
00041
00042 /** \defgroup ata_devices           ATA device definitions
00043
00044     The constants here represent the valid values that can be set as the active
00045     device on the ATA bus. You should pass one of these values to the
00046     g1_ata_select_device() function to select the appropriate device.
00047
00048     \note   Many times, the value returned by the g1_ata_select_device()
00049             function will have other bits set than the constants below. You
00050             should AND the value returned from that function with 0x10 if you
00051             really need to know what device is actually selected. If the value
00052             returned is true when ANDed with 0x10, then the slave device was
```

```
00053             selected, otherwise the master device was. The other bits are either
00054             command-specific or are reserved for compatibility sake.
00055
00056     @{
00057 */
00058 /** \brief  ATA master device.
00059
00060     This constant selects the master device on the ATA bus. This is normally the
00061     GD-ROM drive.
00062
00063     \note   The GD-ROM really does not like the reserved bits being set in the
00064             device select register, hence why this constant doesn't select them.
00065             Some hard drives may require them, however. If you find one that
00066             does, then you should use the \ref G1_ATA_MASTER_ALT constant to
00067             access it if it is the master device on the bus.
00068 */
00069 #define G1_ATA_MASTER       0x00
00070
00071 /** \brief  ATA master device (compatible with old drives).
00072
00073     This constant selects the master device on the ATA bus, with the old
00074     reserved bits set to 1. If you have a drive that predates ATA-2, then this
00075     will probably be the constant you want to access it as the master device.
00076
00077     \note   Do not use this constant to access the GD-ROM. It will not work. Use
00078             \ref G1_ATA_MASTER instead.
00079 */
00080 #define G1_ATA_MASTER_ALT   0x90
00081
00082 /** \brief  ATA slave device.
00083
00084     This constant selects the slave device on the ATA bus. This is where you
00085     would find a hard drive, if the user has an adapter installed.
00086 */
00087 #define G1_ATA_SLAVE        0xB0
00088
00089 /** \brief  Select LBA addressing mode.
00090
00091     OR this constant with one of the device constants (\ref G1_ATA_MASTER or
00092     \ref G1_ATA_SLAVE) to select LBA addressing mode. The various g1_ata_*
00093     functions all do this as appropriate already, so you shouldn't have to worry
00094     about this one at all. This bit is irrelevant for packet devices.
00095 */
00096 #define G1_ATA_LBA_MODE     0x40
00097 /** @} */
00098
00099 /** \brief  Is there a G1 DMA in progress currently?
00100
00101     This function returns non-zero if a DMA is in progress. This can be used to
00102     check on the completion of DMA transfers when non-blocking mode was selected
00103     at transfer time.
00104
00105     \return                 0 if no DMA is in progress, nonzero otherwise.
00106 */
00107 int g1_dma_in_progress(void);
00108
00109 /** \brief  Lock the G1 ATA mutex.
00110
00111     This function locks the mutex that arbitrates access to the ATA bus. You
00112     should never have to do this on your own unless you're accessing devices
00113     manually yourself.
00114
00115     \return                 0 on success, -1 on failure.
00116     \note                   Failure conditions are exactly the same as the
00117                             \ref mutex_lock() function.
00118 */
00119 int g1_ata_mutex_lock(void);
00120
00121 /** \brief  Unlock the G1 ATA mutex.
00122
00123     This function unlocks the mutex that arbitrates access to the ATA bus. You
00124     should never have to do this on your own unless you're accessing devices
00125     manually yourself.
00126
00127     \return                 0 on success, -1 on failure.
00128     \note                   Failure conditions are exactly the same as the
00129                             \ref mutex_unlock() function.
00130 */
00131 int g1_ata_mutex_unlock(void);
00132
00133 /** \brief  Set the active ATA device.
```

```
00134
00135     This function sets the device that any further ATA commands will go to. You
00136     shouldn't have to ever call this yourself, as it should be done for you by
00137     any of the access functions. This must be called with the ATA lock held.
00138
00139     \param  dev             The device to access (generally either
00140                             \ref G1_ATA_MASTER or \ref G1_ATA_SLAVE).
00141     \return                 The previous active device (or 0x0F if the function
00142                             would block in an IRQ handler).
00143
00144     \note                   This function may block if there is a transfer
00145                             ongoing. If called in an IRQ handler and the call
00146                             would otherwise block, 0x0F is returned.
00147 */
00148 uint8_t g1_ata_select_device(uint8_t dev);
00149
00150 /** \brief  Read one or more disk sectors with Cylinder-Head-Sector addressing.
00151
00152     This function reads one or more 512-byte disk blocks from the slave device
00153     on the G1 ATA bus using Cylinder-Head-Sector addressing. This function uses
00154     PIO and blocks until the data is read in.
00155
00156     \param  c               The cylinder to start reading from.
00157     \param  h               The head to start reading from.
00158     \param  s               The sector to start reading from.
00159     \param  count           The number of disk sectors to read.
00160     \param  buf             Storage for the read-in disk sectors. This should be
00161                             at least (count * 512) bytes in length, and must be
00162                             at least 16-bit aligned.
00163     \return                 0 on success. < 0 on failure, setting errno as
00164                             appropriate.
00165
00166     \note                   Unless you're accessing a really old hard drive, you
00167                             probably do not want to use this function to access
00168                             the disk. Use the g1_ata_read_lba() function instead
00169                             of this one, unless you get an error from that
00170                             function indicating that LBA addressing is not
00171                             supported.
00172
00173     \par    Error Conditions:
00174     \em     EIO - an I/O error occurred in reading data \n
00175     \em     ENXIO - ATA support not initialized or no device attached \n
00176     \em     EOVERFLOW - one or more of the requested sectors is out of the
00177                         range of the disk
00178 */
00179 int g1_ata_read_chs(uint16_t c, uint8_t h, uint8_t s, size_t count,
00180                     void *buf);
00181
00182 /** \brief  Write one or more disk sectors with Cylinder-Head-Sector addressing.
00183
00184     This function writes one or more 512-byte disk blocks to the slave device
00185     on the G1 ATA bus using Cylinder-Head-Sector addressing. This function uses
00186     PIO and blocks until the data is written.
00187
00188     \param  c               The cylinder to start writing to.
00189     \param  h               The head to start writing to.
00190     \param  s               The sector to start writing to.
00191     \param  count           The number of disk sectors to write.
00192     \param  buf             The data to write to the disk. This should be
00193                             (count * 512) bytes in length and must be at least
00194                             16-bit aligned.
00195     \return                 0 on success. < 0 on failure, setting errno as
00196                             appropriate.
00197
00198     \note                   Unless you're accessing a really old hard drive, you
00199                             probably do not want to use this function to access
00200                             the disk. Use the g1_ata_write_lba() function
00201                             instead of this one, unless you get an error from
00202                             that function indicating that LBA addressing is not
00203                             supported.
00204
00205     \par    Error Conditions:
00206     \em     ENXIO - ATA support not initialized or no device attached \n
00207     \em     EOVERFLOW - one or more of the requested sectors is out of the
00208                         range of the disk
00209 */
00210 int g1_ata_write_chs(uint16_t c, uint8_t h, uint8_t s, size_t count,
00211                      const void *buf);
00212
00213 /** \brief  Read one or more disk sectors with Linear Block Addressing (LBA).
00214
```

```
00215     This function reads one or more 512-byte disk blocks from the slave device
00216     on the G1 ATA bus using LBA mode (either 28 or 48 bits, as appropriate).
00217     This function uses PIO and blocks until the data is read.
00218
00219     \param  sector          The sector to start reading from.
00220     \param  count           The number of disk sectors to read.
00221     \param  buf             Storage for the read-in disk sectors. This should be
00222                             at least (count * 512) bytes in length, and must be
00223                             at least 16-bit aligned.
00224     \return                 0 on success. < 0 on failure, setting errno as
00225                             appropriate.
00226
00227     \note                   If errno is set to ENOTSUP after calling this
00228                             function, you must use the g1_ata_read_chs()
00229                             function instead.
00230
00231     \par    Error Conditions:
00232     \em     EIO - an I/O error occurred in reading data \n
00233     \em     ENXIO - ATA support not initialized or no device attached \n
00234     \em     EOVERFLOW - one or more of the requested sectors is out of the
00235                         range of the disk \n
00236     \em     ENOTSUP - LBA mode not supported by the device
00237 */
00238 int g1_ata_read_lba(uint64_t sector, size_t count, void *buf);
00239
00240 /** \brief  DMA read disk sectors with Linear Block Addressing (LBA).
00241
00242     This function reads one or more 512-byte disk blocks from the slave device
00243     on the G1 ATA bus using LBA mode (either 28 or 48 bits, as appropriate).
00244     This function uses DMA and optionally blocks until the data is read.
00245
00246     \param  sector          The sector to start reading from.
00247     \param  count           The number of disk sectors to read.
00248     \param  buf             Storage for the read-in disk sectors. This should be
00249                             at least (count * 512) bytes in length, and must be
00250                             at least 32-byte aligned.
00251     \param  block           Non-zero to block until the transfer completes.
00252     \return                 0 on success. < 0 on failure, setting errno as
00253                             appropriate.
00254
00255     \note                   If errno is set to ENOTSUP after calling this
00256                             function, you must use a CHS addressed transfer
00257                             function instead, like g1_ata_read_chs().
00258
00259     \note                   If errno is set to EPERM after calling this
00260                             function, DMA mode is not supported. You should use
00261                             a PIO transfer function like g1_ata_read_lba()
00262                             instead.
00263
00264     \par    Error Conditions:
00265     \em     EIO - an I/O error occurred in reading data \n
00266     \em     ENXIO - ATA support not initialized or no device attached \n
00267     \em     EOVERFLOW - one or more of the requested sectors is out of the
00268                         range of the disk \n
00269     \em     ENOTSUP - LBA mode not supported by the device \n
00270     \em     EPERM - device does not support DMA
00271 */
00272 int g1_ata_read_lba_dma(uint64_t sector, size_t count, void *buf,
00273                         int block);
00274
00275 /** \brief  Write one or more disk sectors with Linear Block Addressing (LBA).
00276
00277     This function writes one or more 512-byte disk blocks to the slave device
00278     on the G1 ATA bus using LBA mode (either 28 or 48 bits, as appropriate).
00279     This function uses PIO and blocks until the data is written.
00280
00281     \param  sector          The sector to start writing to.
00282     \param  count           The number of disk sectors to write.
00283     \param  buf             The data to write to the disk. This should be
00284                             (count * 512) bytes in length and must be at least
00285                             16-bit aligned.
00286     \return                 0 on success. < 0 on failure, setting errno as
00287                             appropriate.
00288
00289     \note                   If errno is set to ENOTSUP after calling this
00290                             function, you must use the g1_ata_write_chs()
00291                             function instead.
00292
00293     \par    Error Conditions:
00294     \em     ENXIO - ATA support not initialized or no device attached \n
00295     \em     EOVERFLOW - one or more of the requested sectors is out of the
```

```
00296                         range of the disk \n
00297     \em     ENOTSUP - LBA mode not supported by the device
00298 */
00299 int g1_ata_write_lba(uint64_t sector, size_t count, const void *buf);
00300
00301 /** \brief  DMA Write disk sectors with Linear Block Addressing (LBA).
00302
00303     This function writes one or more 512-byte disk blocks to the slave device
00304     on the G1 ATA bus using LBA mode (either 28 or 48 bits, as appropriate).
00305     This function uses DMA and optionally blocks until the data is written.
00306
00307     \param  sector          The sector to start writing to.
00308     \param  count           The number of disk sectors to write.
00309     \param  buf             The data to write to the disk. This should be
00310                             (count * 512) bytes in length and must be at least
00311                             32-byte aligned.
00312     \param  block           Non-zero to block until the transfer completes.
00313     \return                 0 on success. < 0 on failure, setting errno as
00314                             appropriate.
00315
00316     \note                   If errno is set to ENOTSUP after calling this
00317                             function, you must use the g1_ata_write_chs()
00318                             function instead.
00319
00320     \note                   If errno is set to EPERM after calling this
00321                             function, DMA mode is not supported. You should use
00322                             a PIO transfer function like g1_ata_write_lba()
00323                             instead.
00324
00325     \par    Error Conditions:
00326     \em     ENXIO - ATA support not initialized or no device attached \n
00327     \em     EOVERFLOW - one or more of the requested sectors is out of the
00328                         range of the disk \n
00329     \em     ENOTSUP - LBA mode not supported by the device \n
00330     \em     EPERM - device does not support DMA
00331 */
00332 int g1_ata_write_lba_dma(uint64_t sector, size_t count, const void *buf,
00333                          int block);
00334
00335 /** \brief  Flush the write cache on the attached disk.
00336
00337     This function flushes the write cache on the disk attached as the slave
00338     device on the G1 ATA bus. This ensures that all writes that have previously
00339     completed are fully persisted to the disk. You should do this before
00340     unmounting any disks or exiting your program if you have called any of the
00341     write functions in here.
00342
00343     \return                 0 on success. <0 on error, setting errno as
00344                             appropriate.
00345
00346     \par    Error Conditions:
00347     \em     ENXIO - ATA support not initialized or no device attached
00348 */
00349 int g1_ata_flush(void);
00350
00351 /** \brief  Get LBA mode of the attached disk.
00352
00353     \return                 -1 on error, 0 - CHS, 28 - LBA28, 48 - LBA48
00354
00355     \par    Error Conditions:
00356     \em     ENXIO - ATA support not initialized or no device attached
00357 */
00358 int g1_ata_lba_mode(void);
00359
00360 /** \brief  Get a block device for a given partition on the slave ATA device.
00361
00362     This function creates a block device descriptor for the given partition on
00363     the attached ATA device. This block device is used to interface with various
00364     filesystems on the device.
00365
00366     \param  partition       The partition number (0-3) to use.
00367     \param  dma             Set to 1 to use DMA for reads/writes on the device,
00368                             if available.
00369     \param  rv              Used to return the block device. Must be non-NULL.
00370     \param  partition_type  Used to return the partition type. Must be non-NULL.
00371     \retval 0               On success.
00372     \retval -1              On error, errno will be set as appropriate.
00373
00374     \par    Error Conditions:
00375     \em     ENXIO - ATA support not initialized or no device attached \n
00376     \em     EIO - an I/O error occurred in reading data \n
```

```
00377     \em     EINVAL - invalid partition number was given \n
00378     \em     EFAULT - rv or partition_type was NULL \n
00379     \em     ENOENT - no MBR found \n
00380     \em     ENOENT - no partition at the specified position \n
00381     \em     ENOMEM - out of memory
00382
00383     \note   This interface currently only supports MBR-formatted disks. There
00384             is currently no support for GPT partition tables.
00385 */
00386 int g1_ata_blockdev_for_partition(int partition, int dma, kos_blockdev_t *rv,
00387                                   uint8_t *partition_type);
00388
00389 /** \brief  Get a block device for the attached ATA device.
00390
00391     This function creates a block device descriptor for the attached ATA device.
00392
00393     \param  dma             Set to 1 to use DMA for reads/writes on the device,
00394                             if available.
00395     \param  rv              Used to return the block device. Must be non-NULL.
00396     \retval 0               On success.
00397     \retval -1              On error, errno will be set as appropriate.
00398
00399     \par    Error Conditions:
00400     \em     ENXIO - ATA support not initialized or no device attached \n
00401     \em     EFAULT - rv was NULL \n
00402     \em     ENOMEM - out of memory
00403 */
00404 int g1_ata_blockdev_for_device(int dma, kos_blockdev_t *rv);
00405
00406 /** \brief  Initialize G1 ATA support.
00407
00408     This function initializes the rest of this subsystem and completes a scan of
00409     the G1 ATA bus for devices. This function may take a while to complete with
00410     some devices. Currently only the slave device is scanned, as the master
00411     device should always be the GD-ROM drive.
00412
00413     \return                 0 on success, <0 on error or if no device is present
00414 */
00415 int g1_ata_init(void);
00416
00417 /** \brief  Shut down G1 ATA support.
00418
00419     This function shuts down the rest of this subsystem, and attempts to flush
00420     the write cache of any attached slave devices. Accessing any ATA devices
00421     using this subsystem after this function is called may produce undefined
00422     results.
00423 */
00424 void g1_ata_shutdown(void);
00425
00426 __END_DECLS
00427
00428 #endif /* __DC_G1ATA_H */
```

9.190 kernel/arch/dreamcast/include/dc/g2bus.h File Reference

G2 bus memory interface.

```
#include <sys/cdefs.h>
#include <stdint.h>
#include <arch/irq.h>
#include <arch/types.h>
#include <dc/fifo.h>
```

Data Structures

- struct g2_ctx_t

  G2 context.

Macros

- #define G2_DMA_TO_G2 0

  G2Bus DMA direction.
- #define G2_DMA_TO_SH4 1

List of G2 Bus channels

AICA (SPU) is channel 0, BBA uses channel 1. CH2_DMA_G2CHN and CH3_DMA_G2CHN are not currently tied to any specific device.

Note

A change in the implementation has rendered ∗_DMA_MODE and ∗_DMA_SHCHN obsolete.

In the current implementation, ∗_DMA_MODE should always be set to zero (representing CPU_TRIGGER). There is also no involvement of SH4-DMA with G2-DMA; therefore, the ∗_DMA_SHCHN values have been deprecated.

- #define G2_DMA_CHAN_SPU 0

  AICA: G2 channel 0.
- #define G2_DMA_CHAN_BBA 1

  BBA: G2 channel 1.
- #define G2_DMA_CHAN_CH2 2

  CH2: G2 channel 2.
- #define G2_DMA_CHAN_CH3 3

  CH3: G2 channel 3.

Typedefs

- typedef void(∗ g2_dma_callback_t) (void ∗data)

  G2Bus DMA interrupt callback type.

Functions

- int g2_dma_transfer (void ∗sh4, void ∗g2bus, size_t length, uint32_t block, g2_dma_callback_t callback, void ∗cbdata, uint32_t dir, uint32_t mode, uint32_t g2chn, uint32_t sh4chn)

  Perform a DMA transfer between SH-4 RAM and G2 Bus.
- int g2_dma_init (void)

  Initialize DMA support.
- void g2_dma_shutdown (void)

  Shutdown DMA support.
- static g2_ctx_t g2_lock (void)

  Disable IRQs and G2 DMA.
- static void g2_unlock (g2_ctx_t ctx)

  Enable IRQs and G2 DMA.
- uint8_t g2_read_8 (uintptr_t address)

  Read one byte from G2.
- void g2_write_8 (uintptr_t address, uint8_t value)

  Write a single byte to G2.
- uint16 g2_read_16 (uintptr_t address)

Read one 16-bit word from G2.

- void g2_write_16 (uintptr_t address, uint16_t value)

  Write a 16-bit word to G2.
- uint32_t g2_read_32 (uintptr_t address)

  Read one 32-bit dword from G2.
- void g2_write_32 (uintptr_t address, uint32_t value)

  Write a 32-bit dword to G2.
- void g2_read_block_8 (uint8_t *output, uintptr_t address, size_t amt)

  Read a block of bytes from G2.
- void g2_write_block_8 (const uint8_t *input, uintptr_t address, size_t amt)

  Write a block of bytes to G2.
- void g2_read_block_16 (uint16_t *output, uintptr_t address, size_t amt)

  Read a block of words from G2.
- void g2_write_block_16 (const uint16_t *input, uintptr_t address, size_t amt)

  Write a block of words to G2.
- void g2_read_block_32 (uint32_t *output, uintptr_t address, size_t amt)

  Read a block of dwords from G2.
- void g2_write_block_32 (const uint32_t *input, uintptr_t address, size_t amt)

  Write a block of dwords to G2.
- void g2_memset_8 (uintptr_t address, uint8_t c, size_t amt)

  Set a block of bytes to G2.
- void g2_fifo_wait (void)

  Wait for the G2 write FIFO to empty.

9.190.1 Detailed Description

G2 bus memory interface.

This file provides low-level support for accessing devices on the G2 bus in the Dreamcast. The G2 bus contains the AICA, as well as the expansion port. Generally, you won't be dealing with things at this level, but rather on the level of the device you're actually interested in working with. Most of the expansion port devices (the modem, bba, and lan adapter) all have their own drivers that work off of this functionality.

The G2 bus is notoroiously picky about a lot of things. You have to be careful to use the right access size for whatever you're working with. Also you can't be doing PIO and DMA at the same time. Finally, there's a FIFO to contend with when you're doing PIO stuff as well. Generally, G2 is a pain in the rear, so you really do want to be using the higher-level stuff related to each device if at all possible!

Author

Megan Potter

Andy Barajas

9.190.2 Macro Definition Documentation

G2_DMA_CHAN_BBA

```
#define G2_DMA_CHAN_BBA 1
```

BBA: G2 channel 1.

G2_DMA_CHAN_CH2

```
#define G2_DMA_CHAN_CH2 2
```

CH2: G2 channel 2.

G2_DMA_CHAN_CH3

```
#define G2_DMA_CHAN_CH3 3
```

CH3: G2 channel 3.

G2_DMA_CHAN_SPU

```
#define G2_DMA_CHAN_SPU 0
```

AICA: G2 channel 0.

G2_DMA_TO_G2

```
#define G2_DMA_TO_G2 0
```

G2Bus DMA direction.

The direction you want the data to go. SH4 to AICA, BBA, etc use SH4TOG2BUS, otherwise G2BUSTOSH4.

G2_DMA_TO_SH4

```
#define G2_DMA_TO_SH4 1
```

9.190.3 Typedef Documentation

g2_dma_callback_t

```
typedef void(* g2_dma_callback_t) (void *data)
```

G2Bus DMA interrupt callback type.

Functions that act as callbacks when G2-DMA completes should be of this type. These functions will be called inside an interrupt context, so don't try to use anything that might stall.

Parameters

data	User data passed in to the g2_dma_transfer() function.

9.190.4 Function Documentation

g2_dma_init()

```
int g2_dma_init (
            void  )
```

Initialize DMA support.

This function sets up the DMA support for transfers to/from the G2 Bus.

Return values

0	On success (no error conditions defined).

g2_dma_shutdown()

```
void g2_dma_shutdown (
            void  )
```

Shutdown DMA support.

g2_dma_transfer()

```
int g2_dma_transfer (
            void * sh4,
            void * g2bus,
            size_t length,
            uint32_t block,
            g2_dma_callback_t callback,
            void * cbdata,
            uint32_t dir,
            uint32_t mode,
            uint32_t g2chn,
            uint32_t sh4chn )
```

Perform a DMA transfer between SH-4 RAM and G2 Bus.

This function copies a block of data between SH-4 RAM and G2 Bus via DMA. You specify the direction of the copy (SH4TOG2BUS/G2BUSTOSH4). There are all kinds of constraints that must be fulfilled to actually do this, so make sure to read all the fine print with the parameter list.

If a callback is specified, it will be called in an interrupt context, so keep that in mind in writing the callback.

Parameters

sh4	Where to copy from/to. Must be 32-byte aligned.
g2bus	Where to copy from/to. Must be 32-byte aligned.
length	The number of bytes to copy. Must be a multiple of 32.
block	Non-zero if you want the function to block until the DMA completes.
callback	A function to call upon completion of the DMA.
cbdata	Data to pass to the callback function.
dir	SH4TOG2BUS or G2BUSTOSH4.
mode	Ignored; for compatibility only.
g2chn	See g2b_channels.
sh4chn	Ignored; for compatibility only.

Return values

0	On success.
-1	On failure. Sets errno as appropriate.

Error Conditions:

EINPROGRESS - DMA already in progress
EFAULT - sh4 and/or g2bus is not 32-byte aligned
EINVAL - Invalid g2chn *EIO* - I/O error

g2_fifo_wait()

```
void g2_fifo_wait (
             void  )
```

Wait for the G2 write FIFO to empty.

This function will spinwait until the G2 FIFO indicates that it has been drained. The FIFO is 32 bytes in length, and thus when accessing AICA you must do this at least for every 8 32-bit writes that you execute.

g2_lock()

```
static g2_ctx_t g2_lock (
             void  )  [inline], [static]
```

Disable IRQs and G2 DMA.

This function makes the following g2_read_∗()/g2_write_∗() functions atomic by disabling IRQs and G2 DMA and storing their states. Pass the context created by this function to g2_unlock() to re-enable IRQs and G2 DMA.

Returns

The context containing the IRQ and G2 DMA states.

References FIFO_G2, FIFO_SH4, FIFO_STATUS, irq_disable(), and g2_ctx_t::irq_state.

g2_memset_8()

```
void g2_memset_8 (
            uintptr_t address,
            uint8_t c,
            size_t amt )
```

Set a block of bytes to G2.

This function acts as memset() for setting a block of bytes on G2. It will take the necessary precautions for accessing G2.

Parameters

address	The address in G2-space to write to.
c	The byte to write.
amt	The number of bytes to write.

g2_read_16()

```
uint16 g2_read_16 (
            uintptr_t address )
```

Read one 16-bit word from G2.

This function reads a single word from the specified address, taking all necessary precautions that are required for accessing G2.

Parameters

address	The address in memory to read.

Returns

The word read from the address specified.

g2_read_32()

```
uint32_t g2_read_32 (
            uintptr_t address )
```

Read one 32-bit dword from G2.

This function reads a single dword from the specified address, taking all necessary precautions that are required for accessing G2.

Parameters

address	The address in memory to read.

Returns

The dword read from the address specified.

g2_read_8()

```
uint8_t g2_read_8 (
            uintptr_t address )
```

Read one byte from G2.

This function reads a single byte from the specified address, taking all necessary precautions that are required for accessing G2.

Parameters

address	The address in memory to read.

Returns

The byte read from the address specified.

g2_read_block_16()

```
void g2_read_block_16 (
            uint16_t * output,
            uintptr_t address,
            size_t amt )
```

Read a block of words from G2.

This function acts as memcpy() for copying data from G2 to system memory, but it copies 16 bits at a time. It will take the necessary precautions before accessing G2 for you as well.

Parameters

output	Pointer in system memory to write to.
address	The address in G2-space to read from.
amt	The number of words to read.

g2_read_block_32()

```
void g2_read_block_32 (
            uint32_t * output,
            uintptr_t address,
            size_t amt )
```

Read a block of dwords from G2.

This function acts as memcpy() for copying data from G2 to system memory, but it copies 32 bits at a time. It will take the necessary precautions before accessing G2 for you as well.

Parameters

output	Pointer in system memory to write to.
address	The address in G2-space to read from.
amt	The number of dwords to read.

g2_read_block_8()

```
void g2_read_block_8 (
            uint8_t * output,
            uintptr_t address,
            size_t amt )
```

Read a block of bytes from G2.

This function acts as memcpy() for copying data from G2 to system memory. It will take the necessary precautions before accessing G2 for you as well.

Parameters

output	Pointer in system memory to write to.
address	The address in G2-space to read from.
amt	The number of bytes to read.

g2_unlock()

```
static void g2_unlock (
            g2_ctx_t ctx )  [inline], [static]
```

Enable IRQs and G2 DMA.

This function restores the IRQ and G2 DMA states using the context value generated by g2_lock().

Parameters

ctx	The context containing IRQ and G2 DMA states.

References irq_restore(), and g2_ctx_t::irq_state.

g2_write_16()

```
void g2_write_16 (
            uintptr_t address,
            uint16_t value )
```

Write a 16-bit word to G2.

This function writes one word to the specified address, taking all the necessary precautions to ensure your write actually succeeds.

Parameters

address	The address in memory to write to.
value	The value to write to that address.

g2_write_32()

```
void g2_write_32 (
            uintptr_t address,
            uint32_t value )
```

Write a 32-bit dword to G2.

This function writes one dword to the specified address, taking all the necessary precautions to ensure your write actually succeeds.

Parameters

address	The address in memory to write to.
value	The value to write to that address.

g2_write_8()

```
void g2_write_8 (
            uintptr_t address,
            uint8_t value )
```

Write a single byte to G2.

This function writes one byte to the specified address, taking all the necessary precautions to ensure your write actually succeeds.

Parameters

address	The address in memory to write to.
value	The value to write to that address.

g2_write_block_16()

```
void g2_write_block_16 (
            const uint16_t * input,
            uintptr_t address,
            size_t amt )
```

Write a block of words to G2.

This function acts as memcpy() for copying data to G2 from system memory, copying 16 bits at a time. It will take the necessary precautions for accessing G2.

Parameters

input	The pointer in system memory to read from.
address	The address in G2-space to write to.
amt	The number of words to write.

g2_write_block_32()

```
void g2_write_block_32 (
            const uint32_t * input,
            uintptr_t address,
            size_t amt )
```

Write a block of dwords to G2.

This function acts as memcpy() for copying data to G2 from system memory, copying 32 bits at a time. It will take the necessary precautions for accessing G2.

Parameters

input	The pointer in system memory to read from.
address	The address in G2-space to write to.
amt	The number of dwords to write.

g2_write_block_8()

```
void g2_write_block_8 (
             const uint8_t * input,
             uintptr_t address,
             size_t amt )
```

Write a block of bytes to G2.

This function acts as memcpy() for copying data to G2 from system memory. It will take the necessary precautions for accessing G2.

Parameters

input	The pointer in system memory to read from.
address	The address in G2-space to write to.
amt	The number of bytes to write.

9.191 g2bus.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    g2bus.h
00004    Copyright (C) 2002 Megan Potter
00005    Copyright (C) 2023 Andy Barajas
00006
00007 */
00008
00009 /** \file   dc/g2bus.h
00010     \brief  G2 bus memory interface.
00011
00012     This file provides low-level support for accessing devices on the G2 bus in
00013     the Dreamcast. The G2 bus contains the AICA, as well as the expansion port.
00014     Generally, you won't be dealing with things at this level, but rather on the
00015     level of the device you're actually interested in working with. Most of the
00016     expansion port devices (the modem, bba, and lan adapter) all have their own
00017     drivers that work off of this functionality.
00018
00019     The G2 bus is notoroiously picky about a lot of things. You have to be
00020     careful to use the right access size for whatever you're working with. Also
00021     you can't be doing PIO and DMA at the same time. Finally, there's a FIFO to
00022     contend with when you're doing PIO stuff as well. Generally, G2 is a pain in
00023     the rear, so you really do want to be using the higher-level stuff related
00024     to each device if at all possible!
00025
00026     \author Megan Potter
00027     \author Andy Barajas
00028 */
00029
00030 #ifndef __DC_G2BUS_H
00031 #define __DC_G2BUS_H
00032
00033 #include <sys/cdefs.h>
00034 __BEGIN_DECLS
00035
00036 #include <stdint.h>
00037 #include <arch/irq.h>
00038 #include <arch/types.h>
00039
00040 #include <dc/fifo.h>
00041
00042 /** \name       List of G2 Bus channels
00043
```

```
00044     AICA (SPU) is channel 0, BBA uses channel 1. CH2_DMA_G2CHN and
00045     CH3_DMA_G2CHN are not currently tied to any specific device.
00046
00047     \note
00048       A change in the implementation has rendered *_DMA_MODE and *_DMA_SHCHN
00049       obsolete.
00050
00051       In the current implementation, *_DMA_MODE should always be set to zero
00052       (representing CPU_TRIGGER). There is also no involvement of SH4-DMA with
00053       G2-DMA; therefore, the *_DMA_SHCHN values have been deprecated.
00054
00055     @{
00056 */
00057 #define G2_DMA_CHAN_SPU  0 /**< \brief AICA: G2 channel 0 */
00058 #define G2_DMA_CHAN_BBA  1 /**< \brief BBA:  G2 channel 1 */
00059 #define G2_DMA_CHAN_CH2  2 /**< \brief CH2: G2 channel 2 */
00060 #define G2_DMA_CHAN_CH3  3 /**< \brief CH3: G2 channel 3 */
00061 /** @} */
00062
00063 /** \brief  G2Bus DMA direction
00064
00065     The direction you want the data to go. SH4 to AICA, BBA, etc use
00066     SH4TOG2BUS, otherwise G2BUSTOSH4.
00067 */
00068 #define G2_DMA_TO_G2   0
00069 #define G2_DMA_TO_SH4  1
00070
00071 /** \brief  G2Bus DMA interrupt callback type.
00072
00073     Functions that act as callbacks when G2-DMA completes should be of this type.
00074     These functions will be called inside an interrupt context, so don't try to
00075     use anything that might stall.
00076
00077     \param  data            User data passed in to the g2_dma_transfer()
00078                             function.
00079 */
00080 typedef void (*g2_dma_callback_t)(void *data);
00081
00082 /** \brief  Perform a DMA transfer between SH-4 RAM and G2 Bus
00083
00084     This function copies a block of data between SH-4 RAM and G2 Bus via DMA.
00085     You specify the direction of the copy (SH4TOG2BUS/G2BUSTOSH4). There are all
00086     kinds of constraints that must be fulfilled to actually do this, so
00087     make sure to read all the fine print with the parameter list.
00088
00089     If a callback is specified, it will be called in an interrupt context, so
00090     keep that in mind in writing the callback.
00091
00092     \param  sh4             Where to copy from/to. Must be 32-byte aligned.
00093     \param  g2bus           Where to copy from/to. Must be 32-byte aligned.
00094     \param  length          The number of bytes to copy. Must be a multiple of
00095                             32.
00096     \param  block           Non-zero if you want the function to block until the
00097                             DMA completes.
00098     \param  callback        A function to call upon completion of the DMA.
00099     \param  cbdata          Data to pass to the callback function.
00100     \param  dir             SH4TOG2BUS or G2BUSTOSH4.
00101     \param  mode            Ignored; for compatibility only.
00102     \param  g2chn           See g2b_channels.
00103     \param  sh4chn          Ignored; for compatibility only.
00104     \retval 0               On success.
00105     \retval -1              On failure. Sets errno as appropriate.
00106
00107     \par    Error Conditions:
00108     \em     EINPROGRESS - DMA already in progress \n
00109     \em     EFAULT - sh4 and/or g2bus is not 32-byte aligned \n
00110     \em     EINVAL - Invalid g2chn
00111     \em     EIO - I/O error
00112
00113 */
00114 int g2_dma_transfer(void *sh4, void *g2bus, size_t length, uint32_t block,
00115                     g2_dma_callback_t callback, void *cbdata,
00116                     uint32_t dir, uint32_t mode, uint32_t g2chn, uint32_t sh4chn);
00117
00118 /** \brief  Initialize DMA support.
00119
00120     This function sets up the DMA support for transfers to/from the G2 Bus.
00121
00122     \retval 0               On success (no error conditions defined).
00123 */
00124 int g2_dma_init(void);
```

```
00125
00126 /** \brief  Shutdown DMA support. */
00127 void g2_dma_shutdown(void);
00128
00129 /** \brief  G2 context
00130
00131     A G2 context containing the states of IRQs and G2 DMA. This struct
00132     is used in with g2_lock() and g2_unlock().
00133 */
00134 typedef struct {
00135     uint32_t irq_state;    /** \brief IRQ state when entering a G2 critical block */
00136 } g2_ctx_t;
00137
00138 /* Internal constants to access suspend registers for G2 DMA. They are not meant for
00139    user-code use. */
00140 /** \cond */
00141 #define G2_DMA_SUSPEND_SPU     (*((vuint32 *)0xa05f781C))
00142 #define G2_DMA_SUSPEND_BBA     (*((vuint32 *)0xa05f783C))
00143 #define G2_DMA_SUSPEND_CH2     (*((vuint32 *)0xa05f785C))
00144 /** \endcond */
00145
00146 /** \brief  Disable IRQs and G2 DMA
00147
00148     This function makes the following g2_read_*()/g2_write_*() functions atomic
00149     by disabling IRQs and G2 DMA and storing their states. Pass the context
00150     created by this function to g2_unlock() to re-enable IRQs and G2 DMA.
00151
00152     \return                The context containing the IRQ and G2 DMA states.
00153 */
00154 static inline g2_ctx_t g2_lock(void) {
00155     g2_ctx_t ctx;
00156
00157     ctx.irq_state = irq_disable();
00158
00159     /* Suspend any G2 DMA */
00160     G2_DMA_SUSPEND_SPU = 1;
00161     G2_DMA_SUSPEND_BBA = 1;
00162     G2_DMA_SUSPEND_CH2 = 1;
00163
00164     while(FIFO_STATUS & (FIFO_SH4 | FIFO_G2));
00165
00166     return ctx;
00167 }
00168
00169 /** \brief  Enable IRQs and G2 DMA
00170
00171     This function restores the IRQ and G2 DMA states using the context value
00172     generated by g2_lock().
00173
00174     \param  ctx            The context containing IRQ and G2 DMA states.
00175 */
00176 static inline void g2_unlock(g2_ctx_t ctx) {
00177     /* Restore suspended G2 DMA */
00178     G2_DMA_SUSPEND_SPU = 0;
00179     G2_DMA_SUSPEND_BBA = 0;
00180     G2_DMA_SUSPEND_CH2 = 0;
00181
00182     irq_restore(ctx.irq_state);
00183 }
00184
00185
00186 #undef G2_DMA_SUSPEND_SPU
00187 #undef G2_DMA_SUSPEND_BBA
00188 #undef G2_DMA_SUSPEND_CH2
00189
00190 /** \brief  Read one byte from G2.
00191
00192     This function reads a single byte from the specified address, taking all
00193     necessary precautions that are required for accessing G2.
00194
00195     \param  address        The address in memory to read.
00196     \return                The byte read from the address specified.
00197 */
00198 uint8_t g2_read_8(uintptr_t address);
00199
00200 /** \brief  Write a single byte to G2.
00201
00202     This function writes one byte to the specified address, taking all the
00203     necessary precautions to ensure your write actually succeeds.
00204
00205     \param  address        The address in memory to write to.
```

```
00206     \param  value           The value to write to that address.
00207 */
00208 void g2_write_8(uintptr_t address, uint8_t value);
00209
00210 /** \brief  Read one 16-bit word from G2.
00211
00212     This function reads a single word from the specified address, taking all
00213     necessary precautions that are required for accessing G2.
00214
00215     \param  address         The address in memory to read.
00216     \return                 The word read from the address specified.
00217 */
00218 uint16 g2_read_16(uintptr_t address);
00219
00220 /** \brief  Write a 16-bit word to G2.
00221
00222     This function writes one word to the specified address, taking all the
00223     necessary precautions to ensure your write actually succeeds.
00224
00225     \param  address         The address in memory to write to.
00226     \param  value           The value to write to that address.
00227 */
00228 void g2_write_16(uintptr_t address, uint16_t value);
00229
00230 /** \brief  Read one 32-bit dword from G2.
00231
00232     This function reads a single dword from the specified address, taking all
00233     necessary precautions that are required for accessing G2.
00234
00235     \param  address         The address in memory to read.
00236     \return                 The dword read from the address specified.
00237 */
00238 uint32_t g2_read_32(uintptr_t address);
00239
00240 /** \brief  Write a 32-bit dword to G2.
00241
00242     This function writes one dword to the specified address, taking all the
00243     necessary precautions to ensure your write actually succeeds.
00244
00245     \param  address         The address in memory to write to.
00246     \param  value           The value to write to that address.
00247 */
00248 void g2_write_32(uintptr_t address, uint32_t value);
00249
00250 /** \brief  Read a block of bytes from G2.
00251
00252     This function acts as memcpy() for copying data from G2 to system memory. It
00253     will take the necessary precautions before accessing G2 for you as well.
00254
00255     \param  output          Pointer in system memory to write to.
00256     \param  address         The address in G2-space to read from.
00257     \param  amt             The number of bytes to read.
00258 */
00259 void g2_read_block_8(uint8_t * output, uintptr_t address, size_t amt);
00260
00261 /** \brief  Write a block of bytes to G2.
00262
00263     This function acts as memcpy() for copying data to G2 from system memory. It
00264     will take the necessary precautions for accessing G2.
00265
00266     \param  input           The pointer in system memory to read from.
00267     \param  address         The address in G2-space to write to.
00268     \param  amt             The number of bytes to write.
00269 */
00270 void g2_write_block_8(const uint8_t * input, uintptr_t address, size_t amt);
00271
00272 /** \brief  Read a block of words from G2.
00273
00274     This function acts as memcpy() for copying data from G2 to system memory,
00275     but it copies 16 bits at a time. It will take the necessary precautions
00276     before accessing G2 for you as well.
00277
00278     \param  output          Pointer in system memory to write to.
00279     \param  address         The address in G2-space to read from.
00280     \param  amt             The number of words to read.
00281 */
00282 void g2_read_block_16(uint16_t * output, uintptr_t address, size_t amt);
00283
00284 /** \brief  Write a block of words to G2.
00285
00286     This function acts as memcpy() for copying data to G2 from system memory,
```

```
00287     copying 16 bits at a time. It will take the necessary precautions for
00288     accessing G2.
00289
00290     \param  input           The pointer in system memory to read from.
00291     \param  address         The address in G2-space to write to.
00292     \param  amt             The number of words to write.
00293 */
00294 void g2_write_block_16(const uint16_t * input, uintptr_t address, size_t amt);
00295
00296 /** \brief  Read a block of dwords from G2.
00297
00298     This function acts as memcpy() for copying data from G2 to system memory,
00299     but it copies 32 bits at a time. It will take the necessary precautions
00300     before accessing G2 for you as well.
00301
00302     \param  output          Pointer in system memory to write to.
00303     \param  address         The address in G2-space to read from.
00304     \param  amt             The number of dwords to read.
00305 */
00306 void g2_read_block_32(uint32_t * output, uintptr_t address, size_t amt);
00307
00308 /** \brief  Write a block of dwords to G2.
00309
00310     This function acts as memcpy() for copying data to G2 from system memory,
00311     copying 32 bits at a time. It will take the necessary precautions for
00312     accessing G2.
00313
00314     \param  input           The pointer in system memory to read from.
00315     \param  address         The address in G2-space to write to.
00316     \param  amt             The number of dwords to write.
00317 */
00318 void g2_write_block_32(const uint32_t * input, uintptr_t address, size_t amt);
00319
00320 /** \brief  Set a block of bytes to G2.
00321
00322     This function acts as memset() for setting a block of bytes on G2. It will
00323     take the necessary precautions for accessing G2.
00324
00325     \param  address         The address in G2-space to write to.
00326     \param  c               The byte to write.
00327     \param  amt             The number of bytes to write.
00328 */
00329 void g2_memset_8(uintptr_t address, uint8_t c, size_t amt);
00330
00331 /** \brief  Wait for the G2 write FIFO to empty.
00332
00333     This function will spinwait until the G2 FIFO indicates that it has been
00334     drained. The FIFO is 32 bytes in length, and thus when accessing AICA you
00335     must do this at least for every 8 32-bit writes that you execute.
00336 */
00337 void g2_fifo_wait(void);
00338
00339 __END_DECLS
00340
00341 #endif  /* __DC_G2BUS_H */
00342
```

9.192 kernel/arch/dreamcast/include/dc/maple.h File Reference

Maple Bus driver interface.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <sys/queue.h>
```

Data Structures

- struct maple_frame_t

Maple frame to be queued for transport.

- struct maple_devinfo_t

  Maple device info structure.
- struct maple_response_t

  Maple response frame structure.
- struct maple_device_t

  One maple device.
- struct maple_port_t

  Internal representation of a Maple port.
- struct maple_driver_t

  A maple device driver.
- struct maple_state_t

  Maple state structure.

Macros

- #define MAPLE_DMA_DEBUG 0

  Enable Maple DMA debugging.
- #define MAPLE_IRQ_DEBUG 0

  Enable Maple IRQ debugging.
- #define MAPLE_BASE 0xa05f6c00

  Maple register base.
- #define MAPLE_DMAADDR (MAPLE_BASE+0x04)

  DMA address register.
- #define MAPLE_RESET2 (MAPLE_BASE+0x10)

  Reset register #2.
- #define MAPLE_ENABLE (MAPLE_BASE+0x14)

  Enable register.
- #define MAPLE_STATE (MAPLE_BASE+0x18)

  Status register.
- #define MAPLE_SPEED (MAPLE_BASE+0x80)

  Speed register.
- #define MAPLE_RESET1 (MAPLE_BASE+0x8c)

  Reset register #1.
- #define MAPLE_RESET2_MAGIC 0

  2nd reset value
- #define MAPLE_ENABLE_ENABLED 1

  Enable Maple.
- #define MAPLE_ENABLE_DISABLED 0

  Disable Maple.
- #define MAPLE_STATE_IDLE 0

  Idle state.
- #define MAPLE_STATE_DMA 1

  DMA in-progress.
- #define MAPLE_SPEED_2MBPS 0

  2Mbps bus speed

- #define MAPLE_SPEED_TIMEOUT(n) ((n) << 16)

  Bus timeout macro.
- #define MAPLE_RESET1_MAGIC 0x6155404f

  First reset value.
- #define MAPLE_RESPONSE_FILEERR -5

  File error.
- #define MAPLE_RESPONSE_AGAIN -4

  Try again later.
- #define MAPLE_RESPONSE_BADCMD -3

  Bad command sent.
- #define MAPLE_RESPONSE_BADFUNC -2

  Bad function code.
- #define MAPLE_RESPONSE_NONE -1

  No response.
- #define MAPLE_COMMAND_DEVINFO 1

  Device info request.
- #define MAPLE_COMMAND_ALLINFO 2

  All info request.
- #define MAPLE_COMMAND_RESET 3

  Reset device request.
- #define MAPLE_COMMAND_KILL 4

  Kill device request.
- #define MAPLE_RESPONSE_DEVINFO 5

  Device info response.
- #define MAPLE_RESPONSE_ALLINFO 6

  All info response.
- #define MAPLE_RESPONSE_OK 7

  Command completed ok.
- #define MAPLE_RESPONSE_DATATRF 8

  Data transfer.
- #define MAPLE_COMMAND_GETCOND 9

  Get condition request.
- #define MAPLE_COMMAND_GETMINFO 10

  Get memory information.
- #define MAPLE_COMMAND_BREAD 11

  Block read.
- #define MAPLE_COMMAND_BWRITE 12

  Block write.
- #define MAPLE_COMMAND_BSYNC 13

  Block sync.
- #define MAPLE_COMMAND_SETCOND 14

  Set condition request.
- #define MAPLE_COMMAND_MICCONTROL 15

  Microphone control.
- #define MAPLE_COMMAND_CAMCONTROL 17

  Camera control.
- #define MAPLE_FUNC_PURUPURU 0x00010000

Jump pack.

- #define MAPLE_FUNC_MOUSE 0x00020000

  Mouse.
- #define MAPLE_FUNC_CAMERA 0x00080000

  Camera (Dreameye)
- #define MAPLE_FUNC_CONTROLLER 0x01000000

  Controller.
- #define MAPLE_FUNC_MEMCARD 0x02000000

  Memory card.
- #define MAPLE_FUNC_LCD 0x04000000

  LCD screen.
- #define MAPLE_FUNC_CLOCK 0x08000000

  Clock.
- #define MAPLE_FUNC_MICROPHONE 0x10000000

  Microphone.
- #define MAPLE_FUNC_ARGUN 0x20000000

  AR gun?
- #define MAPLE_FUNC_KEYBOARD 0x40000000

  Keyboard.
- #define MAPLE_FUNC_LIGHTGUN 0x80000000

  Lightgun.
- #define MAPLE_FRAME_VACANT 0

  Ready to be used.
- #define MAPLE_FRAME_UNSENT 1

  Ready to be sent.
- #define MAPLE_FRAME_SENT 2

  Frame has been sent, but no response yet.
- #define MAPLE_FRAME_RESPONDED 3

  Frame has a response.
- #define MAPLE_PORT_COUNT 4

  Number of ports on the bus.
- #define MAPLE_UNIT_COUNT 6

  Max number of units per port.
- #define MAPLE_DMA_SIZE 16384

  Maple DMA buffer size.
- #define maple_read(A) (*((vuint32*)(A)))

  Maple memory read macro.
- #define maple_write(A, V) (*((vuint32*)(A)) = (V))

  Maple memory write macro.
- #define MAPLE_EOK 0

  No error.
- #define MAPLE_EFAIL -1

  Command failed.
- #define MAPLE_EAGAIN -2

  Try again later.
- #define MAPLE_EINVALID -3

  Invalid command.

- #define MAPLE_ENOTSUPP -4

  Command not suppoted by device.
- #define MAPLE_ETIMEOUT -5

  Command timed out.
- #define MAPLE_FOREACH_BEGIN(TYPE, VARTYPE, VAR)

  Begin a foreach loop over Maple devices.
- #define MAPLE_FOREACH_END()

  End a foreach loop over Maple devices.

Typedefs

- typedef void(∗ maple_attach_callback_t) (maple_device_t ∗dev)

  Maple attach callback type.
- typedef void(∗ maple_detach_callback_t) (maple_device_t ∗dev)

  Maple detach callback type.

Functions

- void maple_bus_enable (void)

  Enable the Maple bus.
- void maple_bus_disable (void)

  Disable the Maple bus.
- void maple_dma_start (void)

  Start a Maple DMA.
- void maple_dma_stop (void)

  Stop a Maple DMA.
- int maple_dma_in_progress (void)

  Is a Maple DMA in progress?
- void maple_dma_addr (void ∗ptr)

  Set the Maple DMA address.
- uint8 maple_addr (int port, int unit)

  Return a "maple address" for a port, unit pair.
- void maple_raddr (uint8 addr, int ∗port, int ∗unit)

  Decompose a "maple address" into a port, unit pair.
- const char ∗ maple_pcaps (uint32 functions)

  Return a string with the capabilities of a given function code.
- const char ∗ maple_perror (int response)

  Return a string representing the maple response code.
- int maple_dev_valid (int p, int u)

  Determine if a given device is valid.
- int maple_gun_enable (int port)

  Enable light gun mode for this frame.
- void maple_gun_disable (void)

  Disable light gun mode.
- void maple_gun_read_pos (int ∗x, int ∗y)

  Read the light gun position values.

- void maple_queue_flush (void)

  Send all queued frames.
- int maple_queue_frame (maple_frame_t *frame)

  Submit a frame for queueing.
- int maple_queue_remove (maple_frame_t *frame)

  Remove a used frame from the queue.
- void maple_frame_init (maple_frame_t *frame)

  Initialize a new frame to prepare it to be placed on the queue.
- int maple_frame_lock (maple_frame_t *frame)

  Lock a frame so that someone else can't use it in the mean time.
- void maple_frame_unlock (maple_frame_t *frame)

  Unlock a frame.
- int maple_driver_reg (maple_driver_t *driver)

  Register a maple device driver.
- int maple_driver_unreg (maple_driver_t *driver)

  Unregister a maple device driver.
- int maple_driver_attach (maple_frame_t *det)

  Attach a maple device to a driver, if possible.
- int maple_driver_detach (int p, int u)

  Detach an attached maple device.
- int maple_driver_foreach (maple_driver_t *drv, int(*callback)(maple_device_t *))

  For each device which the given driver controls, call the callback.
- void maple_attach_callback (uint32 functions, maple_attach_callback_t cb)

  Set an automatic maple attach callback.
- void maple_detach_callback (uint32 functions, maple_detach_callback_t cb)

  Set an automatic maple detach callback.
- void maple_vbl_irq_hnd (uint32 code)

  Called on every VBL (~60fps).
- void maple_dma_irq_hnd (uint32 code)

  Called after a Maple DMA send / receive pair completes.
- int maple_enum_count (void)

  Return the number of connected devices.
- maple_device_t * maple_enum_dev (int p, int u)

  Get a raw device info struct for the given device.
- maple_device_t * maple_enum_type (int n, uint32 func)

  Get the Nth device of the requested type (where N is zero-indexed).
- maple_device_t * maple_enum_type_ex (int n, uint32 func, uint32 cap)

  Return the Nth device that is of the requested type and supports the list of capabilities given.
- void * maple_dev_status (maple_device_t *dev)

  Get the status struct for the requested maple device.
- void maple_init (void)

  Initialize Maple.
- void maple_shutdown (void)

  Shutdown Maple.
- void maple_wait_scan (void)

  Wait for the initial bus scan to complete.

Variables

- maple_state_t maple_state

  Global state info.

9.192.1 Detailed Description

Maple Bus driver interface.

This file provides support for accessing the Maple bus on the Dreamcast. Maple is the bus that all of your controllers and memory cards and the like connect to, so this is one of those types of things that are quite important to know how to use.

Each peripheral device registers their driver within this system, and can be accessed through the functions here. Most of the drivers have their own functionality that is implemented in their header files, as well.

Author

Megan Potter

Lawrence Sebald

See also

dc/maple/controller.h

dc/maple/dreameye.h

dc/maple/keyboard.h

dc/maple/mouse.h

dc/maple/purupuru.h

dc/maple/sip.h

dc/maple/vmu.h

9.192.2 Macro Definition Documentation

MAPLE_DMA_DEBUG

```
#define MAPLE_DMA_DEBUG 0
```

Enable Maple DMA debugging.

Changing this to a 1 will add massive amounts of processing time to the maple system in general, but it can help in verifying DMA errors. In general, for most purposes this should stay disabled.

MAPLE_DMA_SIZE

```
#define MAPLE_DMA_SIZE 16384
```

Maple DMA buffer size.

Increase if you do a *LOT* of maple stuff on every periodic interrupt.

MAPLE_FOREACH_BEGIN

```
#define MAPLE_FOREACH_BEGIN(
                TYPE,
                VARTYPE,
                VAR )
```

Value:

```
    do { \
        maple_device_t  * __dev; \
        VARTYPE * VAR; \
        int __i; \
        \
        __i = 0; \
        while( (__dev = maple_enum_type(__i, TYPE)) ) { \
            VAR = (VARTYPE *)maple_dev_status(__dev); \
            do {
```

Begin a foreach loop over Maple devices.

This macro (along with the MAPLE_FOREACH_END() one) implements a simple foreach-style loop over the given type of devices. Essentially, it grabs the status of the device, and leaves it to you to figure out what to do with it.

The most common use of this would be to look for input on any controller.

Parameters

TYPE	The function code of devices to look at.
VARTYPE	The type to cast the return value of maple_dev_status() to.
VAR	The name of the result of maple_dev_status().

MAPLE_FOREACH_END

```
#define MAPLE_FOREACH_END( )
```

Value:

```
    } while(0); \
    __i++; \
    } \
    } while(0);
```

End a foreach loop over Maple devices.

Each MAPLE_FOREACH_BEGIN() must be paired with one of these after the loop body.

MAPLE_IRQ_DEBUG

```
#define MAPLE_IRQ_DEBUG 0
```

Enable Maple IRQ debugging.

Changing this to a 1 will turn on intra-interrupt debugging messages, which may cause issues if you're using dcload rather than a raw serial debug terminal. You probably will never have a good reason to enable this, so keep it disabled for normal use.

MAPLE_PORT_COUNT

```
#define MAPLE_PORT_COUNT 4
```

Number of ports on the bus.

maple_read

```
#define maple_read(
              A ) ( *((vuint32*)(A)) )
```

Maple memory read macro.

MAPLE_UNIT_COUNT

```
#define MAPLE_UNIT_COUNT 6
```

Max number of units per port.

maple_write

```
#define maple_write(
              A,
              V ) ( *((vuint32*)(A)) = (V) )
```

Maple memory write macro.

9.192.3 Typedef Documentation

maple_attach_callback_t

```
typedef void(* maple_attach_callback_t) (maple_device_t *dev)
```

Maple attach callback type.

Functions of this type can be set with maple_attach_callback() to respond automatically to the attachment of a maple device that supports specified functions.

maple_detach_callback_t

```
typedef void(* maple_detach_callback_t) (maple_device_t *dev)
```

Maple detach callback type.

Functions of this type can be set with maple_detach_callback() to respond automatically to the detachment of a maple device that supports specified functions.

9.192.4 Function Documentation

maple_addr()

```
uint8 maple_addr (
             int port,
             int unit )
```

Return a "maple address" for a port, unit pair.

Parameters

port	The port to build the address for.
unit	The unit to build the address for.

Returns

The Maple address of the pair.

maple_attach_callback()

```
void maple_attach_callback (
            uint32 functions,
            maple_attach_callback_t cb )
```

Set an automatic maple attach callback.

This function sets a callback function to be called when the specified maple device that supports functions has been attached.

Parameters

functions	The functions maple device must support. Set to 0 to support all maple devices.
cb	The callback to call when the maple is attached.

maple_bus_disable()

```
void maple_bus_disable (
            void  )
```

Disable the Maple bus.

There's really not many good reasons to be mucking with this at runtime.

maple_bus_enable()

```
void maple_bus_enable (
            void  )
```

Enable the Maple bus.

This will be done for you autmatically at init time, and there's probably not many reasons to be doing this during runtime.

maple_detach_callback()

```
void maple_detach_callback (
            uint32 functions,
            maple_detach_callback_t cb )
```

Set an automatic maple detach callback.

This function sets a callback function to be called when the specified maple device that supports functions has been detached.

Parameters

functions	The functions maple device must support. Set to 0 to support all maple devices.
cb	The callback to call when the maple is detached.

maple_dev_status()

```
void * maple_dev_status (
            maple_device_t * dev )
```

Get the status struct for the requested maple device.

This function will wait until the status is valid before returning. You should cast to the appropriate type you're expecting.

Parameters

dev	The device to look up.

Returns

The device's status.

maple_dev_valid()

```
int maple_dev_valid (
            int p,
            int u )
```

Determine if a given device is valid.

Parameters

p	The port to check.
u	The unit to check.

Returns

Non-zero if the device is valid.

maple_dma_addr()

```
void maple_dma_addr (
            void * ptr )
```

Set the Maple DMA address.

Once again, you should not muck around with this in your programs.

maple_dma_in_progress()

```
int maple_dma_in_progress (
            void  )
```

Is a Maple DMA in progress?

Returns

Non-zero if a DMA is in progress.

maple_dma_irq_hnd()

```
void maple_dma_irq_hnd (
            uint32 code )
```

Called after a Maple DMA send / receive pair completes.

Parameters

code	The ASIC event code.

maple_dma_start()

```
void maple_dma_start (
            void  )
```

Start a Maple DMA.

This stuff will all be handled internally, so there's probably no reason to be doing this yourself.

maple_dma_stop()

```
void maple_dma_stop (
            void  )
```

Stop a Maple DMA.

This stuff will all be handled internally, so there's probably no reason to be doing this yourself.

maple_driver_attach()

```
int maple_driver_attach (
            maple_frame_t * det )
```

Attach a maple device to a driver, if possible.

Parameters

det	The detection frame.

Return values

0	On success.
-1	If no driver is available.

maple_driver_detach()

```
int maple_driver_detach (
            int p,
            int u )
```

Detach an attached maple device.

Parameters

p	The port of the device to detach.
u	The unit of the device to detach.

Return values

0	On success.
-1	If the device wasn't valid.

maple_driver_foreach()

```
int maple_driver_foreach (
            maple_driver_t * drv,
            int(*)(maple_device_t *) callback )
```

For each device which the given driver controls, call the callback.

Parameters

drv	The driver to loop through devices of.
callback	The function to call. The parameter is the device that it is being called on. It should return 0 on success, and <0 on failure.

Return values

0	On success.
-1	If any callbacks return <0.

maple_driver_reg()

```
int maple_driver_reg (
            maple_driver_t * driver )
```

Register a maple device driver.

This should be done before calling maple_init().

Return values

0	On success (no error conditions defined).

maple_driver_unreg()

```
int maple_driver_unreg (
            maple_driver_t * driver )
```

Unregister a maple device driver.

Return values

0	On success (no error conditions defined).

maple_enum_count()

```
int maple_enum_count (
            void  )
```

Return the number of connected devices.

Returns

The number of devices connected.

maple_enum_dev()

```
maple_device_t * maple_enum_dev (
            int p,
            int u )
```

Get a raw device info struct for the given device.

Parameters

p	The port to look up.
u	The unit to look up.

Returns

The device at that address, or NULL if no device is there.

maple_enum_type()

```
maple_device_t * maple_enum_type (
            int n,
            uint32 func )
```

Get the Nth device of the requested type (where N is zero-indexed).

Parameters

n	The index to look up.
func	The function code to look for.

Returns

The device found, if any. NULL otherwise.

maple_enum_type_ex()

```
maple_device_t * maple_enum_type_ex (
            int n,
            uint32 func,
            uint32 cap )
```

Return the Nth device that is of the requested type and supports the list of capabilities given.

Note, this only currently makes sense for controllers, since some devices don't necessarily use the function data in the same manner that controllers do (and controllers are the only devices where we have a list of what all the bits mean at the moment).

Parameters

n	The index to look up.
func	The function code to look for.
cap	Capabilities bits to look for.

Returns

The device found, if any. NULL otherwise.

maple_frame_init()

```
void maple_frame_init (
            maple_frame_t * frame )
```

Initialize a new frame to prepare it to be placed on the queue.

You should call this before you fill in the frame data.

Parameters

frame	The frame to initialize.

maple_frame_lock()

```
int maple_frame_lock (
            maple_frame_t * frame )
```

Lock a frame so that someone else can't use it in the mean time.

Return values

0	On success.
-1	If the frame is already locked.

maple_frame_unlock()

```
void maple_frame_unlock (
            maple_frame_t * frame )
```

Unlock a frame.

maple_gun_disable()

```
void maple_gun_disable (
            void  )
```

Disable light gun mode.

There is probably very little reason to call this function. Light gun mode is ordinarily disabled and is automatically disabled after the data has been read from the device. The only reason to call this function is if you call the maple_gun_enable() function, and then change your mind during the same frame.

maple_gun_enable()

```
int maple_gun_enable (
            int port )
```

Enable light gun mode for this frame.

This function enables light gun processing for the current frame of data. Light gun mode will automatically be disabled when the data comes back for this frame.

Parameters

port	The port to enable light gun mode on.

Returns

MAPLE_EOK on success, MAPLE_EFAIL on error.

maple_gun_read_pos()

```
void maple_gun_read_pos (
            int * x,
            int * y )
```

Read the light gun position values.

This function fetches the gun position values from the video hardware and returns them via the parameters. These values are not normalized before returning.

Parameters

x	Storage for the horizontal position of the gun.
y	Storage for the vertical position of the gun.

Note

The values returned from this function are the raw H and V counter values from the video hardware where the gun registered its position. The values, however, need a bit of massaging before they correspond nicely to screen values. The y value is particularly odd in interlaced modes due to the fact that you really have half as many physical lines on the screen as you might expect.

maple_init()

```
void maple_init (
            void  )
```

Initialize Maple.

maple_pcaps()

```
const char * maple_pcaps (
            uint32 functions )
```

Return a string with the capabilities of a given function code.

This function is not re-entrant, and thus NOT THREAD SAFE.

Parameters

functions	The list of function codes.

Returns

A string containting the capabilities.

maple_perror()

```
const char * maple_perror (
            int response )
```

Return a string representing the maple response code.

Parameters

response	The response code returned from the function.

Returns

A string containing a textual respresentation of the response code.

maple_queue_flush()

```
void maple_queue_flush (
             void  )
```

Send all queued frames.

maple_queue_frame()

```
int maple_queue_frame (
             maple_frame_t * frame )
```

Submit a frame for queueing.

This will generally be called inside the periodic interrupt; however, if you need to do something asynchronously (e.g., VMU access) then it might cause some problems. In this case, the function will automatically do locking by disabling interrupts temporarily. In any case, the callback will be done inside an IRQ context.

Parameters

frame	The frame to queue up.

Return values

0	On success.
-1	If the frame is already queued.

maple_queue_remove()

```
int maple_queue_remove (
             maple_frame_t * frame )
```

Remove a used frame from the queue.

This will be done automatically when the frame is consumed.

Parameters

frame	The frame to remove from the queue.

Return values

0	On success.
-1	If the frame is not queued.

maple_raddr()

```
void maple_raddr (
            uint8 addr,
            int * port,
            int * unit )
```

Decompose a "maple address" into a port, unit pair.

WARNING: This function will not work with multi-cast addresses!

Parameters

addr	The input address.
port	Output space for the port of the address.
unit	Output space for the unit of the address.

maple_shutdown()

```
void maple_shutdown (
            void  )
```

Shutdown Maple.

maple_vbl_irq_hnd()

```
void maple_vbl_irq_hnd (
            uint32 code )
```

Called on every VBL (~60fps).

Parameters

code	The ASIC event code.

maple_wait_scan()

```
void maple_wait_scan (
             void  )
```

Wait for the initial bus scan to complete.

9.192.5 Variable Documentation

maple_state

```
maple_state_t maple_state  [extern]
```

Global state info.

Do not manipulate this state yourself, as it will likely break things if you do so.

9.193 maple.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/maple.h
00004    Copyright (C) 2002 Megan Potter
00005    Copyright (C) 2015 Lawrence Sebald
00006
00007    This new driver's design is based loosely on the LinuxDC maple
00008    bus driver.
00009 */
00010
00011 /** \file   dc/maple.h
00012     \brief  Maple Bus driver interface.
00013
00014     This file provides support for accessing the Maple bus on the Dreamcast.
00015     Maple is the bus that all of your controllers and memory cards and the like
00016     connect to, so this is one of those types of things that are quite important
00017     to know how to use.
00018
00019     Each peripheral device registers their driver within this system, and can be
00020     accessed through the functions here. Most of the drivers have their own
00021     functionality that is implemented in their header files, as well.
00022
00023     \author Megan Potter
00024     \author Lawrence Sebald
00025
00026     \see    dc/maple/controller.h
00027     \see    dc/maple/dreameye.h
00028     \see    dc/maple/keyboard.h
00029     \see    dc/maple/mouse.h
00030     \see    dc/maple/purupuru.h
00031     \see    dc/maple/sip.h
00032     \see    dc/maple/vmu.h
00033 */
00034
00035 #ifndef __DC_MAPLE_H
00036 #define __DC_MAPLE_H
00037
00038 #include <sys/cdefs.h>
00039 __BEGIN_DECLS
00040
00041 #include <arch/types.h>
00042 #include <sys/queue.h>
00043
00044 /** \brief  Enable Maple DMA debugging.
```

```
00045
00046     Changing this to a 1 will add massive amounts of processing time to the
00047     maple system in general, but it can help in verifying DMA errors. In
00048     general, for most purposes this should stay disabled.
00049 */
00050 #define MAPLE_DMA_DEBUG 0
00051
00052 /** \brief  Enable Maple IRQ debugging.
00053
00054     Changing this to a 1 will turn on intra-interrupt debugging messages, which
00055     may cause issues if you're using dcload rather than a raw serial debug
00056     terminal. You probably will never have a good reason to enable this, so keep
00057     it disabled for normal use.
00058 */
00059 #define MAPLE_IRQ_DEBUG 0
00060
00061 /** \defgroup maple_regs            Maple Bus register locations
00062
00063     These are various registers related to the Maple Bus. In general, you
00064     probably won't ever need to mess with these directly.
00065
00066     @{
00067 */
00068 #define MAPLE_BASE      0xa05f6c00          /**< \brief Maple register base */
00069 #define MAPLE_DMAADDR   (MAPLE_BASE+0x04)   /**< \brief DMA address register */
00070 #define MAPLE_RESET2    (MAPLE_BASE+0x10)   /**< \brief Reset register #2 */
00071 #define MAPLE_ENABLE    (MAPLE_BASE+0x14)   /**< \brief Enable register */
00072 #define MAPLE_STATE     (MAPLE_BASE+0x18)   /**< \brief Status register */
00073 #define MAPLE_SPEED     (MAPLE_BASE+0x80)   /**< \brief Speed register */
00074 #define MAPLE_RESET1    (MAPLE_BASE+0x8c)   /**< \brief Reset register #1 */
00075 /** @} */
00076
00077 /** \defgroup maple_reg_values      Values to write to Maple Bus registers
00078
00079     These are the values that are written to registers to get them to do their
00080     thing.
00081
00082     @{
00083 */
00084 #define MAPLE_RESET2_MAGIC      0               /**< \brief 2nd reset value */
00085 #define MAPLE_ENABLE_ENABLED    1               /**< \brief Enable Maple */
00086 #define MAPLE_ENABLE_DISABLED   0               /**< \brief Disable Maple */
00087 #define MAPLE_STATE_IDLE        0               /**< \brief Idle state */
00088 #define MAPLE_STATE_DMA         1               /**< \brief DMA in-progress */
00089 #define MAPLE_SPEED_2MBPS       0               /**< \brief 2Mbps bus speed */
00090 #define MAPLE_SPEED_TIMEOUT(n)  ((n) << 16)     /**< \brief Bus timeout macro */
00091
00092 #ifndef _arch_sub_naomi
00093 #define MAPLE_RESET1_MAGIC      0x6155404f      /**< \brief First reset value */
00094 #else
00095 #define MAPLE_RESET1_MAGIC      0x6155405f
00096 #endif
00097
00098 /** @} */
00099
00100 /** \defgroup maple_cmds            Maple commands and responses
00101
00102     These are all either commands or responses to commands sent to or from Maple
00103     in normal operation.
00104
00105     @{
00106 */
00107 #define MAPLE_RESPONSE_FILEERR      -5  /**< \brief File error */
00108 #define MAPLE_RESPONSE_AGAIN        -4  /**< \brief Try again later */
00109 #define MAPLE_RESPONSE_BADCMD       -3  /**< \brief Bad command sent */
00110 #define MAPLE_RESPONSE_BADFUNC      -2  /**< \brief Bad function code */
00111 #define MAPLE_RESPONSE_NONE         -1  /**< \brief No response */
00112 #define MAPLE_COMMAND_DEVINFO       1   /**< \brief Device info request */
00113 #define MAPLE_COMMAND_ALLINFO       2   /**< \brief All info request */
00114 #define MAPLE_COMMAND_RESET         3   /**< \brief Reset device request */
00115 #define MAPLE_COMMAND_KILL          4   /**< \brief Kill device request */
00116 #define MAPLE_RESPONSE_DEVINFO      5   /**< \brief Device info response */
00117 #define MAPLE_RESPONSE_ALLINFO      6   /**< \brief All info response */
00118 #define MAPLE_RESPONSE_OK           7   /**< \brief Command completed ok */
00119 #define MAPLE_RESPONSE_DATATRF      8   /**< \brief Data transfer */
00120 #define MAPLE_COMMAND_GETCOND       9   /**< \brief Get condition request */
00121 #define MAPLE_COMMAND_GETMINFO      10  /**< \brief Get memory information */
00122 #define MAPLE_COMMAND_BREAD         11  /**< \brief Block read */
00123 #define MAPLE_COMMAND_BWRITE        12  /**< \brief Block write */
00124 #define MAPLE_COMMAND_BSYNC         13  /**< \brief Block sync */
00125 #define MAPLE_COMMAND_SETCOND       14  /**< \brief Set condition request */
```

```
00126 #define MAPLE_COMMAND_MICCONTROL    15  /**< \brief Microphone control */
00127 #define MAPLE_COMMAND_CAMCONTROL    17  /**< \brief Camera control */
00128 /** @} */
00129
00130 /** \defgroup maple_functions       Maple device function codes
00131
00132     This is the list of maple device types (function codes). Each device must
00133     have at least one function to actually do anything.
00134
00135     @{
00136 */
00137
00138 /* Function codes; most sources claim that these numbers are little
00139    endian, and for all I know, they might be; but since it's a bitmask
00140    it doesn't really make much different. We'll just reverse our constants
00141    from the "big-endian" version. */
00142 #define MAPLE_FUNC_PURUPURU     0x00010000  /**< \brief Jump pack */
00143 #define MAPLE_FUNC_MOUSE        0x00020000  /**< \brief Mouse */
00144 #define MAPLE_FUNC_CAMERA       0x00080000  /**< \brief Camera (Dreameye) */
00145 #define MAPLE_FUNC_CONTROLLER   0x01000000  /**< \brief Controller */
00146 #define MAPLE_FUNC_MEMCARD      0x02000000  /**< \brief Memory card */
00147 #define MAPLE_FUNC_LCD          0x04000000  /**< \brief LCD screen */
00148 #define MAPLE_FUNC_CLOCK        0x08000000  /**< \brief Clock */
00149 #define MAPLE_FUNC_MICROPHONE   0x10000000  /**< \brief Microphone */
00150 #define MAPLE_FUNC_ARGUN        0x20000000  /**< \brief AR gun? */
00151 #define MAPLE_FUNC_KEYBOARD     0x40000000  /**< \brief Keyboard */
00152 #define MAPLE_FUNC_LIGHTGUN     0x80000000  /**< \brief Lightgun */
00153 /** @} */
00154
00155 /* \cond */
00156 /* Pre-define list/queue types */
00157 struct maple_frame;
00158 TAILQ_HEAD(maple_frame_queue, maple_frame);
00159
00160 struct maple_driver;
00161 LIST_HEAD(maple_driver_list, maple_driver);
00162 /* \endcond */
00163
00164 /** \brief  Maple frame to be queued for transport.
00165
00166     Internal representation of a frame to be queued up for sending.
00167
00168     \headerfile dc/maple.h
00169 */
00170 typedef struct maple_frame {
00171     /** \brief  Send queue handle. NOT A FUNCTION! */
00172     TAILQ_ENTRY(maple_frame)    frameq;
00173
00174     int                 cmd;        /**< \brief Command (see \ref maple_cmds) */
00175     int                 dst_port;   /**< \brief Destination port */
00176     int                 dst_unit;   /**< \brief Destination unit */
00177     int                 length;     /**< \brief Data transfer length in 32-bit words */
00178     volatile int        state;      /**< \brief Has this frame been sent / responded to? */
00179     volatile int        queued;     /**< \brief Are we on the queue? */
00180
00181     void                *send_buf;  /**< \brief The data which will be sent (if any) */
00182     uint8               *recv_buf;  /**< \brief Points into recv_buf_arr, but 32-byte aligned */
00183
00184     struct maple_device *dev;       /**< \brief Does this belong to a device? */
00185
00186     void (*callback)(struct maple_frame *);     /**< \brief Response callback */
00187
00188 #if MAPLE_DMA_DEBUG
00189     uint8   recv_buf_arr[1024 + 1024 + 32]; /**< \brief Response receive area */
00190 #else
00191     uint8   recv_buf_arr[1024 + 32];        /**< \brief Response receive area */
00192 #endif
00193 } maple_frame_t;
00194
00195 /** \defgroup maple_frame_states    States that frames can be in
00196     @{
00197 */
00198 #define MAPLE_FRAME_VACANT      0   /**< \brief Ready to be used */
00199 #define MAPLE_FRAME_UNSENT      1   /**< \brief Ready to be sent */
00200 #define MAPLE_FRAME_SENT        2   /**< \brief Frame has been sent, but no response yet */
00201 #define MAPLE_FRAME_RESPONDED   3   /**< \brief Frame has a response */
00202 /** @} */
00203
00204 /** \brief  Maple device info structure.
00205
00206     This structure is used by the hardware to deliver the response to the device
```

```
00207     info request.
00208
00209     \headerfile dc/maple.h
00210 */
00211 typedef struct maple_devinfo {
00212     uint32  functions;              /**< \brief Function codes supported */
00213     uint32  function_data[3];       /**< \brief Additional data per function */
00214     uint8   area_code;              /**< \brief Region code */
00215     uint8   connector_direction;    /**< \brief 0: UP (most controllers), 1: DOWN (lightgun, microphones)
      */
00216     char    product_name[30];       /**< \brief Name of device */
00217     char    product_license[60];    /**< \brief License statement */
00218     uint16  standby_power;          /**< \brief Power consumption (standby) */
00219     uint16  max_power;              /**< \brief Power consumption (max) */
00220 } maple_devinfo_t;
00221
00222 /** \brief  Maple response frame structure.
00223
00224     This structure is used to deliver the actual response to a request placed.
00225     The data field is where all the interesting stuff will be.
00226
00227     \headerfile dc/maple.h
00228 */
00229 typedef struct maple_response {
00230     int8    response;   /**< \brief Response */
00231     uint8   dst_addr;   /**< \brief Destination address */
00232     uint8   src_addr;   /**< \brief Source address */
00233     uint8   data_len;   /**< \brief Data length (in 32-bit words) */
00234     uint8   data[];     /**< \brief Data (if any) */
00235 } maple_response_t;
00236
00237 /** \brief  One maple device.
00238
00239     Note that we duplicate the port/unit info which is normally somewhat
00240     implicit so that we can pass around a pointer to a particular device struct.
00241
00242     \headerfile dc/maple.h
00243 */
00244 typedef struct maple_device {
00245     /* Public */
00246     int             valid;  /**< \brief Is this a valid device? */
00247     int             port;   /**< \brief Maple bus port connected to */
00248     int             unit;   /**< \brief Unit number, off of the port */
00249     maple_devinfo_t info;   /**< \brief Device info struct */
00250
00251     /* Private */
00252     int                     dev_mask;       /**< \brief Device-present mask for unit 0's */
00253     maple_frame_t           frame;          /**< \brief One rx/tx frame */
00254     struct maple_driver     *drv;           /**< \brief Driver which handles this device */
00255
00256     volatile int            status_valid;   /**< \brief Have we got our first status update? */
00257     uint8                   status[1024];   /**< \brief Status buffer (for pollable devices) */
00258 } maple_device_t;
00259
00260 #define MAPLE_PORT_COUNT    4   /**< \brief Number of ports on the bus */
00261 #define MAPLE_UNIT_COUNT    6   /**< \brief Max number of units per port */
00262
00263 /** \brief  Internal representation of a Maple port.
00264
00265     Each maple port can contain up to 6 devices, the first one of which is
00266     always the port itself.
00267
00268     \headerfile dc/maple.h
00269 */
00270 typedef struct maple_port {
00271     int             port;                       /**< \brief Port ID */
00272     maple_device_t  units[MAPLE_UNIT_COUNT];    /**< \brief Pointers to active units */
00273 } maple_port_t;
00274
00275 /** \brief  A maple device driver.
00276
00277     Anything which is added to this list is capable of handling one or more
00278     maple device types. When a device of the given type is connected (includes
00279     startup "connection"), the driver is invoked. This same process happens for
00280     disconnection, response receipt, and on a periodic interval (for normal
00281     updates).
00282
00283     \headerfile dc/maple.h
00284 */
00285 typedef struct maple_driver {
00286     /** \brief  Driver list handle. NOT A FUNCTION! */
```

```
00287     LIST_ENTRY(maple_driver)    drv_list;
00288
00289     uint32      functions;  /**< \brief One or more MAPLE_FUNCs ORed together */
00290     const char  *name;      /**< \brief The driver name */
00291
00292     /* Callbacks, to be filled in by the driver */
00293
00294     /** \brief  Periodic polling callback.
00295
00296         This callback will be called to update the status of connected devices
00297         periodically.
00298
00299         \param  drv         This structure for the driver.
00300     */
00301     void (*periodic)(struct maple_driver *drv);
00302
00303     /** \brief  Device attached callback.
00304
00305         This callback will be called when a new device of this driver is
00306         connected to the system.
00307
00308         \param  drv         This structure for the driver.
00309         \param  dev         The device that was connected.
00310         \return             0 on success, <0 on error.
00311     */
00312     int (*attach)(struct maple_driver *drv, maple_device_t *dev);
00313
00314     /** \brief  Device detached callback.
00315
00316         This callback will be called when a device of this driver is disconnected
00317         from the system.
00318
00319         \param  drv         This structure for the driver.
00320         \param  dev         The device that was detached.
00321     */
00322     void (*detach)(struct maple_driver *drv, maple_device_t *dev);
00323 } maple_driver_t;
00324
00325 /** \brief  Maple state structure.
00326
00327     We put everything in here to keep from polluting the global namespace too
00328     much.
00329
00330     \headerfile dc/maple.h
00331 */
00332 typedef struct maple_state_str {
00333     /** \brief  Maple device driver list. Do not manipulate directly! */
00334     struct maple_driver_list    driver_list;
00335
00336     /** \brief  Maple frame submission queue. Do not manipulate directly! */
00337     struct maple_frame_queue    frame_queue;
00338
00339     /** \brief  Maple device info structure */
00340     maple_port_t                ports[MAPLE_PORT_COUNT];
00341
00342     /** \brief  DMA interrupt counter */
00343     volatile int                dma_cntr;
00344
00345     /** \brief  VBlank interrupt counter */
00346     volatile int                vbl_cntr;
00347
00348     /** \brief  DMA send buffer */
00349     uint8                       *dma_buffer;
00350
00351     /** \brief  Is a DMA running now? */
00352     volatile int                dma_in_progress;
00353
00354     /** \brief  Next port that will be auto-detected */
00355     int                         detect_port_next;
00356
00357     /** \brief  Next unit which will be auto-detected */
00358     int                         detect_unit_next;
00359
00360     /** \brief  Did the detect wrap? */
00361     volatile int                detect_wrapped;
00362
00363     /** \brief  Our vblank handler handle */
00364     int                         vbl_handle;
00365
00366     /** \brief  The port to read for lightgun status, if any. */
00367     int                         gun_port;
```

```
00368
00369     /** \brief  The horizontal position of the lightgun signal. */
00370     int                         gun_x;
00371
00372     /** \brief  The vertical position of the lightgun signal. */
00373     int                         gun_y;
00374 } maple_state_t;
00375
00376 /** \brief  Maple DMA buffer size.
00377
00378     Increase if you do a _LOT_ of maple stuff on every periodic interrupt.
00379 */
00380 #define MAPLE_DMA_SIZE 16384
00381
00382 /* Maple memory read/write functions; these are just hooks in case
00383    we need to do something else later */
00384 /** \brief  Maple memory read macro. */
00385 #define maple_read(A) ( *((vuint32*)(A)) )
00386
00387 /** \brief  Maple memory write macro. */
00388 #define maple_write(A, V) ( *((vuint32*)(A)) = (V) )
00389
00390 /* Return codes from maple access functions */
00391 /** \defgroup maple_func_rvs        Return values from Maple functions
00392     @{
00393 */
00394 #define MAPLE_EOK       0   /**< \brief No error */
00395 #define MAPLE_EFAIL     -1  /**< \brief Command failed */
00396 #define MAPLE_EAGAIN    -2  /**< \brief Try again later */
00397 #define MAPLE_EINVALID  -3  /**< \brief Invalid command */
00398 #define MAPLE_ENOTSUPP  -4  /**< \brief Command not suppoted by device */
00399 #define MAPLE_ETIMEOUT  -5  /**< \brief Command timed out */
00400 /** @} */
00401
00402 /*****************************************************************************/
00403 /* maple_globals.c */
00404
00405 /** \brief  Global state info.
00406
00407     Do not manipulate this state yourself, as it will likely break things if you
00408     do so.
00409 */
00410 extern maple_state_t maple_state;
00411
00412 /*****************************************************************************/
00413 /* maple_utils.c */
00414
00415 /** \brief  Enable the Maple bus.
00416
00417     This will be done for you autmatically at init time, and there's probably
00418     not many reasons to be doing this during runtime.
00419 */
00420 void maple_bus_enable(void);
00421
00422 /** \brief  Disable the Maple bus.
00423
00424     There's really not many good reasons to be mucking with this at runtime.
00425 */
00426 void maple_bus_disable(void);
00427
00428 /** \brief  Start a Maple DMA.
00429
00430     This stuff will all be handled internally, so there's probably no reason to
00431     be doing this yourself.
00432 */
00433 void maple_dma_start(void);
00434
00435 /** \brief  Stop a Maple DMA.
00436
00437     This stuff will all be handled internally, so there's probably no reason to
00438     be doing this yourself.
00439 */
00440 void maple_dma_stop(void);
00441
00442 /** \brief  Is a Maple DMA in progress?
00443
00444     \return                 Non-zero if a DMA is in progress.
00445 */
00446 int maple_dma_in_progress(void);
00447
00448 /** \brief  Set the Maple DMA address.
```

```
00449
00450     Once again, you should not muck around with this in your programs.
00451 */
00452 void maple_dma_addr(void *ptr);
00453
00454 /** \brief  Return a "maple address" for a port, unit pair.
00455
00456     \param  port            The port to build the address for.
00457     \param  unit            The unit to build the address for.
00458     \return                 The Maple address of the pair.
00459 */
00460 uint8 maple_addr(int port, int unit);
00461
00462 /** \brief  Decompose a "maple address" into a port, unit pair.
00463
00464     WARNING: This function will not work with multi-cast addresses!
00465
00466     \param  addr            The input address.
00467     \param  port            Output space for the port of the address.
00468     \param  unit            Output space for the unit of the address.
00469 */
00470 void maple_raddr(uint8 addr, int * port, int * unit);
00471
00472 /** \brief  Return a string with the capabilities of a given function code.
00473
00474     This function is not re-entrant, and thus NOT THREAD SAFE.
00475
00476     \param  functions       The list of function codes.
00477     \return                 A string containting the capabilities.
00478 */
00479 const char * maple_pcaps(uint32 functions);
00480
00481 /** \brief  Return a string representing the maple response code.
00482
00483     \param  response        The response code returned from the function.
00484     \return                 A string containing a textual respresentation of the
00485                             response code.
00486 */
00487 const char * maple_perror(int response);
00488
00489 /** \brief  Determine if a given device is valid.
00490     \param  p               The port to check.
00491     \param  u               The unit to check.
00492     \return                 Non-zero if the device is valid.
00493 */
00494 int maple_dev_valid(int p, int u);
00495
00496 /** \brief  Enable light gun mode for this frame.
00497
00498     This function enables light gun processing for the current frame of data.
00499     Light gun mode will automatically be disabled when the data comes back for
00500     this frame.
00501
00502     \param  port            The port to enable light gun mode on.
00503     \return                 MAPLE_EOK on success, MAPLE_EFAIL on error.
00504 */
00505 int maple_gun_enable(int port);
00506
00507 /** \brief  Disable light gun mode.
00508
00509     There is probably very little reason to call this function. Light gun mode
00510     is ordinarily disabled and is automatically disabled after the data has been
00511     read from the device. The only reason to call this function is if you call
00512     the maple_gun_enable() function, and then change your mind during the same
00513     frame.
00514 */
00515 void maple_gun_disable(void);
00516
00517 /** \brief  Read the light gun position values.
00518
00519     This function fetches the gun position values from the video hardware and
00520     returns them via the parameters. These values are not normalized before
00521     returning.
00522
00523     \param  x               Storage for the horizontal position of the gun.
00524     \param  y               Storage for the vertical position of the gun.
00525
00526     \note   The values returned from this function are the raw H and V counter
00527             values from the video hardware where the gun registered its
00528             position. The values, however, need a bit of massaging before they
00529             correspond nicely to screen values. The y value is particularly odd
```

```
00530             in interlaced modes due to the fact that you really have half as
00531             many physical lines on the screen as you might expect.
00532 */
00533 void maple_gun_read_pos(int *x, int *y);
00534
00535 #if MAPLE_DMA_DEBUG
00536 /* Debugging help */
00537
00538 /** \brief  Setup a sentinel for debugging DMA issues.
00539     \param  buffer          The buffer to add the sentinel to.
00540     \param  bufsize         The size of the data in the buffer.
00541 */
00542 void maple_sentinel_setup(void * buffer, int bufsize);
00543
00544 /** \brief  Verify the presence of the sentine.
00545     \param  bufname         A string to recognize the buffer by.
00546     \param  buffer          The buffer to check.
00547     \param  bufsize         The size of the buffer.
00548 */
00549 void maple_sentinel_verify(const char * bufname, void * buffer, int bufsize);
00550 #endif
00551
00552 /*****************************************************************************/
00553 /* maple_queue.c */
00554
00555 /** \brief  Send all queued frames. */
00556 void maple_queue_flush(void);
00557
00558 /** \brief  Submit a frame for queueing.
00559
00560     This will generally be called inside the periodic interrupt; however, if you
00561     need to do something asynchronously (e.g., VMU access) then it might cause
00562     some problems. In this case, the function will automatically do locking by
00563     disabling interrupts temporarily. In any case, the callback will be done
00564     inside an IRQ context.
00565
00566     \param  frame           The frame to queue up.
00567     \retval 0               On success.
00568     \retval -1              If the frame is already queued.
00569 */
00570 int maple_queue_frame(maple_frame_t *frame);
00571
00572 /** \brief  Remove a used frame from the queue.
00573
00574     This will be done automatically when the frame is consumed.
00575
00576     \param  frame           The frame to remove from the queue.
00577     \retval 0               On success.
00578     \retval -1              If the frame is not queued.
00579 */
00580 int maple_queue_remove(maple_frame_t *frame);
00581
00582 /** \brief  Initialize a new frame to prepare it to be placed on the queue.
00583
00584     You should call this before you fill in the frame data.
00585
00586     \param  frame           The frame to initialize.
00587 */
00588 void maple_frame_init(maple_frame_t *frame);
00589
00590 /** \brief  Lock a frame so that someone else can't use it in the mean time.
00591     \retval 0               On success.
00592     \retval -1              If the frame is already locked.
00593 */
00594 int maple_frame_lock(maple_frame_t *frame);
00595
00596 /** \brief  Unlock a frame. */
00597 void maple_frame_unlock(maple_frame_t *frame);
00598
00599 /*****************************************************************************/
00600 /* maple_driver.c */
00601
00602 /** \brief  Register a maple device driver.
00603
00604     This should be done before calling maple_init().
00605
00606     \retval 0               On success (no error conditions defined).
00607 */
00608 int maple_driver_reg(maple_driver_t *driver);
00609
00610 /** \brief  Unregister a maple device driver.
```

```
00611     \retval 0               On success (no error conditions defined).
00612 */
00613 int maple_driver_unreg(maple_driver_t *driver);
00614
00615 /** \brief  Attach a maple device to a driver, if possible.
00616     \param  det             The detection frame.
00617     \retval 0               On success.
00618     \retval -1              If no driver is available.
00619 */
00620 int maple_driver_attach(maple_frame_t *det);
00621
00622 /** \brief  Detach an attached maple device.
00623     \param  p               The port of the device to detach.
00624     \param  u               The unit of the device to detach.
00625     \retval 0               On success.
00626     \retval -1              If the device wasn't valid.
00627 */
00628 int maple_driver_detach(int p, int u);
00629
00630 /** \brief  For each device which the given driver controls, call the callback.
00631     \param  drv             The driver to loop through devices of.
00632     \param  callback        The function to call. The parameter is the device
00633                             that it is being called on. It should return 0 on
00634                             success, and <0 on failure.
00635     \retval 0               On success.
00636     \retval -1              If any callbacks return <0.
00637 */
00638 int maple_driver_foreach(maple_driver_t *drv, int (*callback)(maple_device_t *));
00639
00640 /** \brief  Maple attach callback type.
00641
00642     Functions of this type can be set with maple_attach_callback() to respond
00643     automatically to the attachment of a maple device that supports specified
00644     functions.
00645 */
00646 typedef void (*maple_attach_callback_t)(maple_device_t *dev);
00647
00648 /** \brief  Set an automatic maple attach callback.
00649
00650     This function sets a callback function to be called when the specified
00651     maple device that supports functions has been attached.
00652
00653     \param  functions       The functions maple device must support. Set to
00654                             0 to support all maple devices.
00655     \param  cb              The callback to call when the maple is attached.
00656 */
00657 void maple_attach_callback(uint32 functions, maple_attach_callback_t cb);
00658
00659 /** \brief  Maple detach callback type.
00660
00661     Functions of this type can be set with maple_detach_callback() to respond
00662     automatically to the detachment of a maple device that supports specified
00663     functions.
00664 */
00665 typedef void (*maple_detach_callback_t)(maple_device_t *dev);
00666
00667 /** \brief  Set an automatic maple detach callback.
00668
00669     This function sets a callback function to be called when the specified
00670     maple device that supports functions has been detached.
00671
00672     \param  functions       The functions maple device must support. Set to
00673                             0 to support all maple devices.
00674     \param  cb              The callback to call when the maple is detached.
00675 */
00676 void maple_detach_callback(uint32 functions, maple_detach_callback_t cb);
00677
00678 /*****************************************************************************/
00679 /* maple_irq.c */
00680
00681 /** \brief  Called on every VBL (~60fps).
00682     \param  code            The ASIC event code.
00683 */
00684 void maple_vbl_irq_hnd(uint32 code);
00685
00686 /** \brief  Called after a Maple DMA send / receive pair completes.
00687     \param  code            The ASIC event code.
00688 */
00689 void maple_dma_irq_hnd(uint32 code);
00690
00691 /*****************************************************************************/
```

```
00692 /* maple_enum.c */
00693
00694 /** \brief  Return the number of connected devices.
00695     \return                 The number of devices connected.
00696 */
00697 int maple_enum_count(void);
00698
00699 /** \brief  Get a raw device info struct for the given device.
00700     \param  p               The port to look up.
00701     \param  u               The unit to look up.
00702     \return                 The device at that address, or NULL if no device is
00703                             there.
00704 */
00705 maple_device_t * maple_enum_dev(int p, int u);
00706
00707 /** \brief  Get the Nth device of the requested type (where N is zero-indexed).
00708     \param  n               The index to look up.
00709     \param  func            The function code to look for.
00710     \return                 The device found, if any. NULL otherwise.
00711 */
00712 maple_device_t * maple_enum_type(int n, uint32 func);
00713
00714 /** \brief  Return the Nth device that is of the requested type and supports the
00715             list of capabilities given.
00716
00717     Note, this only currently makes sense for controllers, since some devices
00718     don't necessarily use the function data in the same manner that controllers
00719     do (and controllers are the only devices where we have a list of what all
00720     the bits mean at the moment).
00721
00722     \param  n               The index to look up.
00723     \param  func            The function code to look for.
00724     \param  cap             Capabilities bits to look for.
00725     \return                 The device found, if any. NULL otherwise.
00726 */
00727 maple_device_t * maple_enum_type_ex(int n, uint32 func, uint32 cap);
00728
00729 /** \brief  Get the status struct for the requested maple device.
00730
00731     This function will wait until the status is valid before returning.
00732     You should cast to the appropriate type you're expecting.
00733
00734     \param  dev             The device to look up.
00735     \return                 The device's status.
00736 */
00737 void * maple_dev_status(maple_device_t *dev);
00738
00739 /*****************************************************************************/
00740 /* maple_init.c */
00741
00742 /** \brief  Initialize Maple. */
00743 void maple_init(void);
00744
00745 /** \brief  Shutdown Maple. */
00746 void maple_shutdown(void);
00747
00748 /** \brief  Wait for the initial bus scan to complete. */
00749 void maple_wait_scan(void);
00750
00751 /*****************************************************************************/
00752 /* Convienence macros */
00753
00754 /* A "foreach" loop to scan all maple devices of a given type. It is used
00755    like this:
00756
00757    MAPLE_FOREACH_BEGIN(MAPLE_FUNC_CONTROLLER, cont_state_t, st)
00758     if(st->buttons & CONT_START)
00759         return -1;
00760    MAPLE_FOREACH_END()
00761
00762    The peripheral index can be obtained with __i, and the raw device struct
00763    with __dev. The code inside the loop is guaranteed to be inside a block
00764    (i.e., { code })
00765  */
00766
00767 /** \brief  Begin a foreach loop over Maple devices.
00768
00769     This macro (along with the MAPLE_FOREACH_END() one) implements a simple
00770     foreach-style loop over the given type of devices. Essentially, it grabs the
00771     status of the device, and leaves it to you to figure out what to do with it.
00772
```

```
00773     The most common use of this would be to look for input on any controller.
00774
00775     \param  TYPE            The function code of devices to look at.
00776     \param  VARTYPE         The type to cast the return value of
00777                             maple_dev_status() to.
00778     \param  VAR             The name of the result of maple_dev_status().
00779 */
00780 #define MAPLE_FOREACH_BEGIN(TYPE, VARTYPE, VAR) \
00781     do { \
00782         maple_device_t  * __dev; \
00783         VARTYPE * VAR; \
00784         int __i; \
00785         \
00786         __i = 0; \
00787         while( (__dev = maple_enum_type(__i, TYPE)) ) { \
00788             VAR = (VARTYPE *)maple_dev_status(__dev); \
00789             do {
00790
00791 /** \brief  End a foreach loop over Maple devices.
00792
00793     Each MAPLE_FOREACH_BEGIN() must be paired with one of these after the loop
00794     body.
00795 */
00796 #define MAPLE_FOREACH_END() \
00797     } while(0); \
00798     __i++; \
00799     } \
00800     } while(0);
00801
00802 __END_DECLS
00803
00804 #endif /* __DC_MAPLE_H */
```

9.194 kernel/arch/dreamcast/include/dc/maple/controller.h File Reference

Definitions for using the controller device.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <stdint.h>
```

Data Structures

- struct cont_state_t

  Controller state structure.

Macros

- #define CONT_C (1<<0)

  C button Mask.
- #define CONT_B (1<<1)

  B button Mask.
- #define CONT_A (1<<2)

  A button Mask.
- #define CONT_START (1<<3)

  Start button Mask.
- #define CONT_DPAD_UP (1<<4)

Main Dpad Up button Mask.

- #define CONT_DPAD_DOWN (1<<5)

  Main Dpad Down button Mask.
- #define CONT_DPAD_LEFT (1<<6)

  Main Dpad Left button Mask.
- #define CONT_DPAD_RIGHT (1<<7)

  Main Dpad right button Mask.
- #define CONT_Z (1<<8)

  Z button Mask.
- #define CONT_Y (1<<9)

  Y button Mask.
- #define CONT_X (1<<10)

  X button Mask.
- #define CONT_D (1<<11)

  D button Mask.
- #define CONT_DPAD2_UP (1<<12)

  Secondary Dpad Up button Mask.
- #define CONT_DPAD2_DOWN (1<<13)

  Secondary Dpad Down button Mask.
- #define CONT_DPAD2_LEFT (1<<14)

  Secondary Dpad Left button Mask.
- #define CONT_DPAD2_RIGHT (1<<15)

  Secondary Dpad Right button Mask.
- #define CONT_RESET_BUTTONS (CONT_A | CONT_B | CONT_X | CONT_Y | CONT_START)

  Controller buttons for standard reset action.
- #define CONT_CAPABILITY_C (1<<24)

  C button capability mask.
- #define CONT_CAPABILITY_B (1<<25)

  B button capability mask.
- #define CONT_CAPABILITY_A (1<<26)

  A button capability mask.
- #define CONT_CAPABILITY_START (1<<27)

  Start button capability mask.
- #define CONT_CAPABILITY_DPAD_UP (1<<28)

  First Dpad up capability mask.
- #define CONT_CAPABILITY_DPAD_DOWN (1<<29)

  First Dpad down capability mask.
- #define CONT_CAPABILITY_DPAD_LEFT (1<<30)

  First Dpad left capability mask.
- #define CONT_CAPABILITY_DPAD_RIGHT (1<<31)

  First Dpad right capability mask.
- #define CONT_CAPABILITY_Z (1<<16)

  Z button capability mask.
- #define CONT_CAPABILITY_Y (1<<17)

  Y button capability mask.
- #define CONT_CAPABILITY_X (1<<18)

  X button capability mask.

- #define CONT_CAPABILITY_D (1<<19)

  D button capability mask.
- #define CONT_CAPABILITY_DPAD2_UP (1<<20)

  Second Dpad up capability mask.
- #define CONT_CAPABILITY_DPAD2_DOWN (1<<21)

  Second Dpad down capability mask.
- #define CONT_CAPABILITY_DPAD2_LEFT (1<<22)

  Second Dpad left capability mask.
- #define CONT_CAPABILITY_DPAD2_RIGHT (1<<23)

  Second Dpad right capability mask.
- #define CONT_CAPABILITY_RTRIG (1<<8)

  Right trigger capability mask.
- #define CONT_CAPABILITY_LTRIG (1<<9)

  Left trigger capability mask.
- #define CONT_CAPABILITY_ANALOG_X (1<<10)

  First analog X axis capability mask.
- #define CONT_CAPABILITY_ANALOG_Y (1<<11)

  First analog Y axis capability mask.
- #define CONT_CAPABILITY_ANALOG2_X (1<<12)

  Second analog X axis capability mask.
- #define CONT_CAPABILITY_ANALOG2_Y (1<<13)

  Second analog Y axis capability mask.
- #define CONT_CAPABILITIES_STANDARD_BUTTONS

  Standard button (A, B, X, Y, Start) controller capabilities.
- #define CONT_CAPABILITIES_DPAD

  Directional pad (up, down, left right) controller capabilities.
- #define CONT_CAPABILITIES_ANALOG

  Analog stick (X, Y axes) controller capabilities.
- #define CONT_CAPABILITIES_TRIGGERS

  Trigger (L, R lever) controller capabilities.
- #define CONT_CAPABILITIES_EXTENDED_BUTTONS

  Extended button (C, Z) controller capabilities.
- #define CONT_CAPABILITIES_SECONDARY_DPAD

  Secondary directional pad (up, down, left, right) controller capabilities.
- #define CONT_CAPABILITIES_SECONDARY_ANALOG

  Secondary analog stick (X, Y axes) controller capabilities.
- #define CONT_CAPABILITIES_DUAL_DPAD

  Both directional pads (up, down, left right) controller capabilities.
- #define CONT_CAPABILITIES_DUAL_ANALOG

  Both analog sticks (X, Y axes) controller capabilities.
- #define CONT_TYPE_STANDARD_CONTROLLER

  Standard controller type.
- #define CONT_TYPE_DUAL_ANALOG_CONTROLLER

  Dual analog controller type.
- #define CONT_TYPE_ASCII_PAD

  ASCII fighting pad controller type.
- #define CONT_TYPE_ARCADE_STICK

Arcade stick controller type.

- #define CONT_TYPE_TWIN_STICK

  Twin stick joystick controller type.
- #define CONT_TYPE_ASCII_MISSION_STICK

  ASCII Mission Stick controller type.
- #define CONT_TYPE_RACING_CONTROLLER

  Racing wheel/controller type.
- #define CONT_TYPE_MARACAS

  Samba De Amigo maraca controller type.
- #define CONT_TYPE_DANCE_MAT

  Dance Dance Revolution mat controller type.
- #define CONT_TYPE_FISHING_ROD

  Fishing rod controller type.
- #define CONT_TYPE_POP_N_MUSIC

  Pop'n'Music controller type.
- #define CONT_TYPE_DENSHA_DE_GO

  Densha de Go! controller type.

Typedefs

- typedef void(∗ cont_btn_callback_t) (uint8_t addr, uint32_t btns)

  Controller automatic callback type.

Functions

- void cont_btn_callback (uint8_t addr, uint32_t btns, cont_btn_callback_t cb)

  Set an automatic button press callback.
- int cont_has_capabilities (const struct maple_device ∗cont, uint32_t capabilities)

  Check for controller capabilities.
- int cont_is_type (const struct maple_device ∗cont, uint32_t type)

  Check for controller type.

9.194.1 Detailed Description

Definitions for using the controller device.

This file contains the definitions needed to access the Maple controller device. Obviously, this corresponds to the MAPLE_FUNC_CONTROLLER function code.

Author

Jordan DeLong

Megan Potter

Falco Girgis

9.195 controller.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/maple/controller.h
00004    Copyright (C) 2000-2002 Jordan DeLong
00005    Copyright (C) 2000-2002 Megan Potter
00006    Copyright (C) 2023 Falco Girgis
00007
00008    Thanks to Marcus Comstedt for information on the controller.
00009 */
00010
00011 /** \file    dc/maple/controller.h
00012     \brief   Definitions for using the controller device.
00013     \ingroup controller
00014
00015     This file contains the definitions needed to access the Maple controller
00016     device. Obviously, this corresponds to the MAPLE_FUNC_CONTROLLER function
00017     code.
00018
00019     \author Jordan DeLong
00020     \author Megan Potter
00021     \author Falco Girgis
00022 */
00023
00024 #ifndef __DC_MAPLE_CONTROLLER_H
00025 #define __DC_MAPLE_CONTROLLER_H
00026
00027 #include <sys/cdefs.h>
00028 __BEGIN_DECLS
00029
00030 #include <arch/types.h>
00031 #include <stdint.h>
00032
00033 /** \defgroup controller Controller
00034     \brief    Controller Maple Device API
00035     \ingroup  maple
00036
00037     This module contains the public API for the controller
00038     maple driver.
00039
00040     A standard, first-party Dreamcast controller has
00041     the following button configuration:
00042
00043                                  ____________
00044                                /  |   __   |  \
00045           L trigger ------|   /   | |  | |    \  |----- R trigger
00046                           _|__/    | |__| |     \_|__
00047                         |  _        \____/           |
00048            Joystick ----|-/ \                 (Y)    |
00049                         | \_/              (X)   (B)|
00050                         |    _                 (A)   |
00051                         |  _|  |_                     |
00052               D-Pad ----|-|_   _|                    |
00053                         |   |_|     /\              |
00054                          \          /__\            /
00055                           \    _____|_____        /
00056                            \  /     |       \   /
00057                             \/      |        \/
00058                                Start button
00059
00060    You can grab a pointer to a connected controller by
00061    using the following:
00062
00063         maple_device_t *device = maple_enum_type(N, MAPLE_FUNC_CONTROLLER);
00064
00065         if(device) printf("Controller found!\n");
00066         else printf("Controller not found!\n");
00067
00068     where N is the controller number. 0 would be the first
00069     controller found, which may not necessarily be on port A.
00070 */
00071
00072 /** \defgroup controller_inputs Querying Inputs
00073     \brief    API used to query for input state
00074     \ingroup  controller
00075
00076     The following API is used to check for a controller's input state.
```

```
00077
00078     You can grab a controller's state structure, containing the state
00079     of all of its inputs by using:
00080
00081         cont_state_t *state = (cont_state_t *)maple_dev_status(device);
00082
00083     Next you can check for the state of a particular button with:
00084
00085         if(state->a)                // Check via bitfield
00086             printf("Pressed A".);
00087
00088     or
00089
00090         if(state->buttons & CONT_A) // Check via applying bitmask
00091             printf("Pressed A.")
00092 */
00093
00094 /** \defgroup controller_input_masks Inputs
00095     \brief    Collection of all status masks for checking input
00096     \ingroup  controller_inputs
00097
00098     A set of bitmasks representing each input source on a controller, used to
00099     check its status.
00100
00101     @{
00102 */
00103 #define CONT_C              (1<<0)      /**< \brief C button Mask. */
00104 #define CONT_B              (1<<1)      /**< \brief B button Mask. */
00105 #define CONT_A              (1<<2)      /**< \brief A button Mask. */
00106 #define CONT_START          (1<<3)      /**< \brief Start button Mask. */
00107 #define CONT_DPAD_UP        (1<<4)      /**< \brief Main Dpad Up button Mask. */
00108 #define CONT_DPAD_DOWN      (1<<5)      /**< \brief Main Dpad Down button Mask. */
00109 #define CONT_DPAD_LEFT      (1<<6)      /**< \brief Main Dpad Left button Mask. */
00110 #define CONT_DPAD_RIGHT     (1<<7)      /**< \brief Main Dpad right button Mask. */
00111 #define CONT_Z              (1<<8)      /**< \brief Z button Mask. */
00112 #define CONT_Y              (1<<9)      /**< \brief Y button Mask. */
00113 #define CONT_X              (1<<10)     /**< \brief X button Mask. */
00114 #define CONT_D              (1<<11)     /**< \brief D button Mask. */
00115 #define CONT_DPAD2_UP       (1<<12)     /**< \brief Secondary Dpad Up button Mask. */
00116 #define CONT_DPAD2_DOWN     (1<<13)     /**< \brief Secondary Dpad Down button Mask. */
00117 #define CONT_DPAD2_LEFT     (1<<14)     /**< \brief Secondary Dpad Left button Mask. */
00118 #define CONT_DPAD2_RIGHT    (1<<15)     /**< \brief Secondary Dpad Right button Mask. */
00119 /** @} */
00120
00121 /** \brief   Controller buttons for standard reset action
00122     \ingroup controller_inputs
00123
00124     Convenience macro providing the standard button combination
00125     used as a reset mechanism by most retail games.
00126 */
00127 #define CONT_RESET_BUTTONS  (CONT_A | CONT_B | CONT_X | CONT_Y | CONT_START)
00128
00129 /** \brief   Controller state structure.
00130     \ingroup controller_inputs
00131
00132     This structure contains information about the status of the controller
00133     device and can be fetched by casting the result of maple_dev_status() to
00134     this structure.
00135
00136     A 1 bit in the buttons' bitfield indicates that a button is pressed, and the
00137     joyx, joyy, joy2x, joy2 values are all 0 based (0 is centered).
00138
00139     \note
00140     Whether a particular field or button is actually used by the controller
00141     depends upon its capabilities. See \ref controller_query_caps.
00142
00143     \sa maple_dev_status
00144 */
00145 typedef struct cont_state {
00146     union {
00147         /** \brief  bit-packed controller button states
00148             \sa     controller_buttons
00149         */
00150         uint32_t buttons;
00151         struct {
00152             uint32_t c: 1;              /**< \brief C button value. */
00153             uint32_t b: 1;              /**< \brief B button value. */
00154             uint32_t a: 1;              /**< \brief A button value. */
00155             uint32_t start: 1;          /**< \brief Start button value. */
00156             uint32_t dpad_up: 1;        /**< \brief Main Dpad Up button value. */
00157             uint32_t dpad_down: 1;      /**< \brief Main Dpad Down button value. */
```

```
00158             uint32_t dpad_left: 1;      /**< \brief Main Dpad Left button value. */
00159             uint32_t dpad_right: 1;     /**< \brief Main Dpad Right button value. */
00160             uint32_t z: 1;              /**< \brief Z button value. */
00161             uint32_t y: 1;              /**< \brief Y button value. */
00162             uint32_t x: 1;              /**< \brief X button value. */
00163             uint32_t d: 1;              /**< \brief D button value. */
00164             uint32_t dpad2_up: 1;       /**< \brief Secondary Dpad Up button value. */
00165             uint32_t dpad2_down: 1;     /**< \brief Secondary Dpad Down button value. */
00166             uint32_t dpad2_left: 1;     /**< \brief Secondary Dpad Left button value. */
00167             uint32_t dpad2_right: 1;    /**< \brief Secondary Dpad Right button value. */
00168             uint32_t: 16;
00169         };
00170     };
00171
00172     int ltrig;    /**< \brief Left trigger value (0-255). */
00173     int rtrig;    /**< \brief Right trigger value (0-255). */
00174     int joyx;     /**< \brief Main joystick x-axis value (-128 - 127). */
00175     int joyy;     /**< \brief Main joystick y-axis value. */
00176     int joy2x;    /**< \brief Secondary joystick x-axis value. */
00177     int joy2y;    /**< \brief Secondary joystick y-axis value. */
00178 } cont_state_t;
00179
00180 /** \brief   Controller automatic callback type.
00181     \ingroup controller_inputs
00182
00183     Functions of this type can be set with cont_btn_callback() to respond
00184     automatically to the specified set of buttons being pressed. This can be
00185     used, for instance, to implement the standard A+B+X+Y+Start method of ending
00186     the program running.
00187
00188     \warning
00189     Your callback will be invoked within a context with interrupts disabled.
00190     See cont_btn_callback for more information.
00191
00192     \param addr             Maple BUS address to poll for the button mask
00193                             on, or 0 for all ports.
00194     \param btns             Mask of all buttons which should be pressed to
00195                             trigger the callback.
00196
00197     \sa cont_btn_callback
00198 */
00199 typedef void (*cont_btn_callback_t)(uint8_t addr, uint32_t btns);
00200
00201 /** \brief   Set an automatic button press callback.
00202     \ingroup controller_inputs
00203
00204     This function sets a callback function to be called when the specified
00205     controller has the set of buttons given pressed.
00206
00207     \note
00208     The callback gets invoked for the given maple port; however, providing
00209     an address of '0' will cause it to be invoked for any port with a
00210     device pressing the given buttons. Since you are passed back the address
00211     of this device, You are free to implement your own filtering logic within
00212     your callback.
00213
00214     \warning
00215     The provided callback function is invoked within a context which has
00216     interrupts disabled. This means that you should not do any sort of complex
00217     processing or make any API calls which depend on interrupts to complete,
00218     such as Maple or etherent processing whichy rely on packet transmission,
00219     any sleeping or threading calls, blocking on any sort of file I/O, etc.
00220     This mechanism is typically used to quickly terminate the application
00221     and should be used with caution.
00222
00223     \param  addr            The controller to listen on (or 0 for all ports).
00224                             This value can be obtained by using maple_addr().
00225     \param  btns            The buttons bitmask to match.
00226     \param  cb              The callback to call when the buttons are pressed.
00227 */
00228 void cont_btn_callback(uint8_t addr, uint32_t btns, cont_btn_callback_t cb);
00229
00230 /** \defgroup controller_query_caps Querying Capabilities
00231     \brief    API used to query for a controller's capabilities
00232     \ingroup  controller
00233
00234     The following API is used to query for the support of individual
00235     or groups of capabilities by a particular device.
00236 */
00237
00238 /** \defgroup controller_caps Capabilities
```

```
00239     \brief    Bit masks used to identify controller capabilities
00240     \ingroup  controller_query_caps
00241
00242     These bits will be set in the function_data for the controller's deviceinfo
00243     if the controller supports the corresponding button/axis capability.
00244
00245     \note
00246     The ordering here is so that they match the order found in
00247     \ref controller_input_masks.
00248
00249     @{
00250 */
00251 #define CONT_CAPABILITY_C               (1<<24)     /**< \brief C button capability mask. */
00252 #define CONT_CAPABILITY_B               (1<<25)     /**< \brief B button capability mask. */
00253 #define CONT_CAPABILITY_A               (1<<26)     /**< \brief A button capability mask. */
00254 #define CONT_CAPABILITY_START           (1<<27)     /**< \brief Start button capability mask. */
00255 #define CONT_CAPABILITY_DPAD_UP         (1<<28)     /**< \brief First Dpad up capability mask. */
00256 #define CONT_CAPABILITY_DPAD_DOWN       (1<<29)     /**< \brief First Dpad down capability mask. */
00257 #define CONT_CAPABILITY_DPAD_LEFT       (1<<30)     /**< \brief First Dpad left capability mask. */
00258 #define CONT_CAPABILITY_DPAD_RIGHT      (1<<31)     /**< \brief First Dpad right capability mask. */
00259 #define CONT_CAPABILITY_Z               (1<<16)     /**< \brief Z button capability mask. */
00260 #define CONT_CAPABILITY_Y               (1<<17)     /**< \brief Y button capability mask. */
00261 #define CONT_CAPABILITY_X               (1<<18)     /**< \brief X button capability mask. */
00262 #define CONT_CAPABILITY_D               (1<<19)     /**< \brief D button capability mask. */
00263 #define CONT_CAPABILITY_DPAD2_UP        (1<<20)     /**< \brief Second Dpad up capability mask. */
00264 #define CONT_CAPABILITY_DPAD2_DOWN      (1<<21)     /**< \brief Second Dpad down capability mask. */
00265 #define CONT_CAPABILITY_DPAD2_LEFT      (1<<22)     /**< \brief Second Dpad left capability mask. */
00266 #define CONT_CAPABILITY_DPAD2_RIGHT     (1<<23)     /**< \brief Second Dpad right capability mask. */
00267 #define CONT_CAPABILITY_RTRIG           (1<<8)      /**< \brief Right trigger capability mask. */
00268 #define CONT_CAPABILITY_LTRIG           (1<<9)      /**< \brief Left trigger capability mask. */
00269 #define CONT_CAPABILITY_ANALOG_X        (1<<10)     /**< \brief First analog X axis capability mask. */
00270 #define CONT_CAPABILITY_ANALOG_Y        (1<<11)     /**< \brief First analog Y axis capability mask. */
00271 #define CONT_CAPABILITY_ANALOG2_X       (1<<12)     /**< \brief Second analog X axis capability mask. */
00272 #define CONT_CAPABILITY_ANALOG2_Y       (1<<13)     /**< \brief Second analog Y axis capability mask. */
00273 /** @} */
00274
00275 /** \defgroup controller_caps_groups Capability Groups
00276     \brief    Bit masks representing common groups of capabilities
00277     \ingroup  controller_query_caps
00278
00279     These are a sets of capabilties providing a
00280     convenient way to test for high-level features,
00281     such as dual-analog sticks or extra buttons.
00282
00283     @{
00284 */
00285 /** \brief Standard button (A, B, X, Y, Start) controller capabilities */
00286 #define CONT_CAPABILITIES_STANDARD_BUTTONS    (CONT_CAPABILITY_A | \
00287                                                CONT_CAPABILITY_B | \
00288                                                CONT_CAPABILITY_X | \
00289                                                CONT_CAPABILITY_Y | \
00290                                                CONT_CAPABILITY_START)
00291
00292 /** \brief Directional pad (up, down, left right) controller capabilities */
00293 #define CONT_CAPABILITIES_DPAD                (CONT_CAPABILITY_DPAD_UP | \
00294                                                CONT_CAPABILITY_DPAD_DOWN | \
00295                                                CONT_CAPABILITY_DPAD_LEFT | \
00296                                                CONT_CAPABILITY_DPAD_RIGHT)
00297
00298 /** \brief Analog stick (X, Y axes) controller capabilities */
00299 #define CONT_CAPABILITIES_ANALOG              (CONT_CAPABILITY_ANALOG_X | \
00300                                                CONT_CAPABILITY_ANALOG_Y)
00301
00302 /** \brief Trigger (L, R lever) controller capabilities */
00303 #define CONT_CAPABILITIES_TRIGGERS            (CONT_CAPABILITY_LTRIG | \
00304                                                CONT_CAPABILITY_RTRIG)
00305
00306 /** \brief Extended button (C, Z) controller capabilities */
00307 #define CONT_CAPABILITIES_EXTENDED_BUTTONS    (CONT_CAPABILITY_C | \
00308                                                CONT_CAPABILITY_Z)
00309
00310 /** \brief Secondary directional pad (up, down, left, right) controller capabilities */
00311 #define CONT_CAPABILITIES_SECONDARY_DPAD      (CONT_CAPABILITY_DPAD2_UP | \
00312                                                CONT_CAPABILITY_DPAD2_DOWN | \
00313                                                CONT_CAPABILITY_DPAD2_LEFT | \
00314                                                CONT_CAPABILITY_DPAD2_RIGHT)
00315
00316 /** \brief Secondary analog stick (X, Y axes) controller capabilities */
00317 #define CONT_CAPABILITIES_SECONDARY_ANALOG    (CONT_CAPABILITY_ANALOG2_X | \
00318                                                CONT_CAPABILITY_ANALOG2_Y)
00319
```

```
00320 /** \brief Both directional pads (up, down, left right) controller capabilities */
00321 #define CONT_CAPABILITIES_DUAL_DPAD           (CONT_CAPABILITIES_DPAD | \
00322                                                CONT_CAPABILITIES_SECONDARY_DPAD)
00323
00324 /** \brief Both analog sticks (X, Y axes) controller capabilities */
00325 #define CONT_CAPABILITIES_DUAL_ANALOG         (CONT_CAPABILITIES_ANALOG | \
00326                                                CONT_CAPABILITIES_SECONDARY_ANALOG)
00327
00328 /* Forward declaration */
00329 struct maple_device;
00330
00331 /** \brief   Check for controller capabilities
00332     \ingroup controller_query_caps
00333
00334     Checks whether or not a controller implements the capabilities
00335     associated with the given type.
00336
00337     \note
00338     Controller capability reporting is an extremely generic mechanism,
00339     such that many peripherals may implement the same capability in
00340     completely different ways. For example, the Samba De Amigo maraca
00341     controller will advertise itself as a dual-analog device, with each
00342     maraca being an analog stick.
00343
00344     \param cont            Pointer to a Maple device structure which
00345                            implements the CONTROLLER function.
00346     \param capabilities    Capability mask the controller is expected
00347                            to implement
00348
00349     \retval 1              The controller supports the given capabilities.
00350     \retval 0              The controller doesn't support the given capabilities.
00351     \retval -1             Invalid controller.
00352
00353     \sa cont_is_type
00354 */
00355 int cont_has_capabilities(const struct maple_device *cont, uint32_t capabilities);
00356 /** @} */
00357
00358 /** \defgroup controller_query_types Querying Types
00359     \brief    API for determining controller types
00360     \ingroup  controller
00361
00362     The following API is for detecting between different types
00363     of standard controllers. These controllers are not identified
00364     by specific model but are instead identified solely by capabilities,
00365     so that homebrew software can remain generic and future-proof to later
00366     homebrew controllers or exotic, untested 3rd party peripherals.
00367
00368     \warning
00369     Usually you want to check if a controller <i>supports the
00370     capabilities</i> of another controller, not whether it is has
00371     the <i>exact</i> same capabilities of a controller. For example,
00372     a controller that happens to come along supporting a dual analog
00373     stick but is otherwise the same layout as a standard controller
00374     would not match the standard controller type; however, it would
00375     implement its capabilities. There exist 3rd party adapters for
00376     connecting dual-analog PS2 controllers to DC which operate
00377     like this today.
00378
00379     \note
00380     If you really want to hard-code the detection of a certain
00381     exact model or brand of controller, instead of basing your
00382     detection upon capabilities, check for its product_name
00383     or license within the maple_devinfo structure.
00384
00385     \sa cont_has_capabilities, maple_devinfo
00386 */
00387
00388 /** \defgroup controller_types Types
00389     \brief    Preconfigured capabilities for standard controllers
00390     \ingroup  controller_query_types
00391
00392     Aggregate capability mask containing all capabilities
00393     which are implemented for a particular controller type.
00394     For example, the standard controller type is simply a
00395     combination of the following capabilities:
00396         - Standard buttons
00397         - Triggers
00398         - Dpad
00399         - Analog
00400
```

```
00401     \note
00402     Because these are technically just capability masks,
00403     a type may also be passed to cont_has_capabilities()
00404     for detecting whether something has <i>at least</i>
00405     the capabilities of a type.
00406
00407     @{
00408 */
00409 /** \brief Standard controller type */
00410 #define CONT_TYPE_STANDARD_CONTROLLER       (CONT_CAPABILITIES_STANDARD_BUTTONS | \
00411                                              CONT_CAPABILITIES_TRIGGERS | \
00412                                              CONT_CAPABILITIES_DPAD | \
00413                                              CONT_CAPABILITIES_ANALOG)
00414
00415 /** \brief Dual analog controller type */
00416 #define CONT_TYPE_DUAL_ANALOG_CONTROLLER    (CONT_CAPABILITIES_STANDARD_BUTTONS | \
00417                                              CONT_CAPABILITIES_TRIGGERS | \
00418                                              CONT_CAPABILITIES_DPAD | \
00419                                              CONT_CAPABILITIES_DUAL_ANALOG)
00420
00421 /** \brief ASCII fighting pad controller type */
00422 #define CONT_TYPE_ASCII_PAD                 (CONT_CAPABILITIES_STANDARD_BUTTONS | \
00423                                              CONT_CAPABILITIES_EXTENDED_BUTTONS | \
00424                                              CONT_CAPABILITIES_DPAD)
00425
00426 /** \brief Arcade stick controller type */
00427 #define CONT_TYPE_ARCADE_STICK              (CONT_CAPABILITIES_STANDARD_BUTTONS | \
00428                                              CONT_CAPABILITIES_EXTENDED_BUTTONS | \
00429                                              CONT_CAPABILITIES_DPAD)
00430
00431 /** \brief Twin stick joystick controller type */
00432 #define CONT_TYPE_TWIN_STICK                (CONT_CAPABILITIES_STANDARD_BUTTONS | \
00433                                              CONT_CAPABILITIES_EXTENDED_BUTTONS | \
00434                                              CONT_CAPABILITY_D | \
00435                                              CONT_CAPABILITIES_DUAL_DPAD)
00436
00437 /** \brief ASCII Mission Stick controller type */
00438 #define CONT_TYPE_ASCII_MISSION_STICK       (CONT_CAPABILITIES_STANDARD_BUTTONS | \
00439                                              CONT_CAPABILITIES_DUAL_DPAD | \
00440                                              CONT_CAPABILITIES_TRIGGERS | \
00441                                              CONT_CAPABILITIES_ANALOG)
00442
00443 /** \brief Racing wheel/controller type */
00444 #define CONT_TYPE_RACING_CONTROLLER         (CONT_CAPABILITY_DPAD_UP | \
00445                                              CONT_CAPABILITY_DPAD_DOWN | \
00446                                              CONT_CAPABILITY_A | \
00447                                              CONT_CAPABILITY_B | \
00448                                              CONT_CAPABILITY_START | \
00449                                              CONT_CAPABILITIES_TRIGGERS | \
00450                                              CONT_CAPABILITY_ANALOG_X \
00451                                              CONT_CAPABILITIES_SECONDARY_ANALOG)
00452
00453 /** \brief Samba De Amigo maraca controller type */
00454 #define CONT_TYPE_MARACAS                   (CONT_CAPABILITY_A | \
00455                                              CONT_CAPABILITY_B | \
00456                                              CONT_CAPABILITY_D | \
00457                                              CONT_CAPABILITY_START | \
00458                                              CONT_CAPABILITIES_EXTENDED_BUTTONS | \
00459                                              CONT_CAPABILITIES_DUAL_ANALOG)
00460
00461 /** \brief Dance Dance Revolution mat controller type */
00462 #define CONT_TYPE_DANCE_MAT                 (CONT_CAPABILITY_A | \
00463                                              CONT_CAPABILITY_B | \
00464                                              CONT_CAPABILITY_START | \
00465                                              CONT_CAPABILITIES_DPAD)
00466
00467 /** \brief Fishing rod controller type */
00468 #define CONT_TYPE_FISHING_ROD               (CONT_CAPABILITIES_STANDARD_BUTTONS | \
00469                                              CONT_CAPABILITIES_DPAD | \
00470                                              CONT_CAPABILITIES_TRIGGERS | \
00471                                              CONT_CAPABILITIES_DUAL_ANALOG)
00472
00473 /** \brief Pop'n'Music controller type */
00474 #define CONT_TYPE_POP_N_MUSIC               (CONT_CAPABILITIES_STANDARD_BUTTONS | \
00475                                              CONT_CAPABILITY_C | \
00476                                              CONT_CAPABILITIES_DPAD)
00477
00478 /** \brief Densha de Go! controller type */
00479 #define CONT_TYPE_DENSHA_DE_GO              (CONT_CAPABILITIES_STANDARD_BUTTONS | \
00480                                              CONT_CAPABILITIES_EXTENDED_BUTTONS | \
00481                                              CONT_CAPABILITY_D | \
```

```
00482                                           CONT_CAPABILITIES_DPAD)
00483 /** @} */
00484
00485 /** \brief   Check for controller type
00486     \ingroup controller_query_types
00487
00488     Checks whether or not a controller has the <i>exact</i>
00489     capabilities associated with the given type.
00490
00491     \warning
00492     Just becase a controller has all of the same capabilities of a
00493     type does not mean that it's that exact type. For example, the
00494     ASCII Pad and Arcade Stick both implement the same capabilties,
00495     although they are not the same controllers. They would be
00496     indistinguishable here, by design, so that you are able to
00497     generalize to a collection of 1st or 3rd party controllers
00498     easily.
00499
00500     \param cont            Pointer to a Maple device structure which
00501                            implements the CONTROLLER function.
00502     \param type            Type identifier or capability mask the
00503                            controller is expected to match
00504
00505     \retval 1              The controller matches the given type.
00506     \retval 0              The controller doesn't match the given type.
00507     \retval -1             Invalid controller.
00508
00509     \sa cont_has_capabilities
00510 */
00511 int cont_is_type(const struct maple_device *cont, uint32_t type);
00512
00513 /* \cond */
00514 /* Init / Shutdown */
00515 void cont_init(void);
00516 void cont_shutdown(void);
00517 /* \endcond */
00518
00519 __END_DECLS
00520
00521 #endif  /* __DC_MAPLE_CONTROLLER_H */
00522
```

9.196 kernel/arch/dreamcast/include/dc/maple/dreameye.h File Reference

Definitions for using the Dreameye Camera device.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <dc/maple.h>
```

Data Structures

- struct dreameye_state_t

  Dreameye status structure.

Macros

- #define DREAMEYE_GETCOND_NUM_IMAGES 0x81

  Get the number of images on the device.
- #define DREAMEYE_GETCOND_TRANSFER_COUNT 0x83

  Get the number of transfers to copy an image.

- #define DREAMEYE_SUBCOMMAND_IMAGEREQ 0x04

  Get an image from the device.
- #define DREAMEYE_SUBCOMMAND_ERASE 0x05

  Erase an image from the device.
- #define DREAMEYE_SUBCOMMAND_ERROR 0xFF

  Error return command.
- #define DREAMEYE_IMAGEREQ_CONTINUE 0x00

  Continue transferring an image.
- #define DREAMEYE_IMAGEREQ_START 0x40

  Start transferring an image from its start.

Functions

- int dreameye_get_image_count (maple_device_t *dev, int block)

  Get the number of images on the Dreameye.
- int dreameye_get_image (maple_device_t *dev, uint8 image, uint8 **data, int *img_sz)

  Transfer an image from the Dreameye.
- int dreameye_erase_image (maple_device_t *dev, uint8 image, int block)

  Erase an image from the Dreameye.

9.196.1 Detailed Description

Definitions for using the Dreameye Camera device.

This file contains the definitions needed to access the Maple Camera type device (aka, the Dreameye). Currently, this driver allows you to download the still pictures that are saved on the camera and delete them. It does not allow you to use the camera for video input currently.

Author

Lawrence Sebald

9.196.2 Macro Definition Documentation

DREAMEYE_GETCOND_NUM_IMAGES

```
#define DREAMEYE_GETCOND_NUM_IMAGES 0x81
```

Get the number of images on the device.

This constant is used with the MAPLE_COMMAND_GETCOND command to fetch the number of images on the device.

DREAMEYE_GETCOND_TRANSFER_COUNT

```
#define DREAMEYE_GETCOND_TRANSFER_COUNT 0x83
```

Get the number of transfers to copy an image.

This constant is used with the MAPLE_COMMAND_GETCOND command to fetch the number of times a transfer command must be sent to get the image specified.

DREAMEYE_IMAGEREQ_CONTINUE

```
#define DREAMEYE_IMAGEREQ_CONTINUE 0x00
```

Continue transferring an image.

DREAMEYE_IMAGEREQ_START

```
#define DREAMEYE_IMAGEREQ_START 0x40
```

Start transferring an image from its start.

DREAMEYE_SUBCOMMAND_ERASE

```
#define DREAMEYE_SUBCOMMAND_ERASE 0x05
```

Erase an image from the device.

This subcommand is used with the MAPLE_COMMAND_CAMCONTROL command to remove an image from the device.

DREAMEYE_SUBCOMMAND_ERROR

```
#define DREAMEYE_SUBCOMMAND_ERROR 0xFF
```

Error return command.

This subcommand is used by the dreameye with the MAPLE_COMMAND_CAMCONTROL command to indicate an error occurred in a subcommand.

DREAMEYE_SUBCOMMAND_IMAGEREQ

```
#define DREAMEYE_SUBCOMMAND_IMAGEREQ 0x04
```

Get an image from the device.

This subcommand is used with the MAPLE_COMMAND_CAMCONTROL command to fetch part of image data from the specified image.

9.196.3 Function Documentation

dreameye_erase_image()

```
int dreameye_erase_image (
            maple_device_t * dev,
            uint8 image,
            int block )
```

Erase an image from the Dreameye.

This function erases the specified image from the Dreameye device. This command can be sent to any of the subdevices of the MAPLE_FUNC_CONTROLLER root device of the Dreameye.

Parameters

dev	The device to erase from.
image	The image number to erase (0xFF to erase all).
block	Set to 1 to wait for the Dreameye to respond.

Return values

MAPLE_EOK	On success.
MAPLE_EAGAIN	Couldn't send the command, try again.
MAPLE_ETIMEOUT	Timeout on blocking.
MAPLE_EINVALID	Invalid image number specified.

dreameye_get_image()

```
int dreameye_get_image (
            maple_device_t * dev,
            uint8 image,
            uint8 ** data,
            int * img_sz )
```

Transfer an image from the Dreameye.

This function fetches a single image from the specified Dreameye device. This function will block, and can take a little while to execute. You must use the first subdevice of the MAPLE_FUNC_CONTROLLER root device of the Dreameye as the dev parameter.

Parameters

dev	The device to get an image from.
image	The image number to download.
data	A pointer to a buffer to store things in. This will be allocated by the function and you are responsible for freeing the data when you are done.
img_sz	A pointer to storage for the size of the image, in bytes.

Return values

MAPLE_EOK	On success.
MAPLE_EFAIL	On error.

dreameye_get_image_count()

```
int dreameye_get_image_count (
            maple_device_t * dev,
            int block )
```

Get the number of images on the Dreameye.

This function fetches the number of saved images on the specified Dreameye device. It can be sent to any of the subdevices of the MAPLE_FUNC_CONTROLLER root device of the Dreameye. When the response comes from the device, the image_count field of the dreameye_state_t for the specified device will have the number of images on the device, and image_count_valid will be set to 1.

Parameters

dev	The device to query.
block	Set to 1 to wait for the Dreameye to respond.

Return values

MAPLE_EOK	On success.
MAPLE_ETIMEOUT	The command timed out while blocking.
MAPLE_EAGAIN	Could not send the command to the device, try again.

9.197 dreameye.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/maple/dreameye.h
00004    Copyright (C) 2005, 2009, 2010 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   dc/maple/dreameye.h
00009     \brief  Definitions for using the Dreameye Camera device.
00010
00011     This file contains the definitions needed to access the Maple Camera type
00012     device (aka, the Dreameye). Currently, this driver allows you to download
00013     the still pictures that are saved on the camera and delete them. It does not
00014     allow you to use the camera for video input currently.
00015
00016     \author Lawrence Sebald
00017 */
00018
00019 #ifndef __DC_MAPLE_DREAMEYE_H
00020 #define __DC_MAPLE_DREAMEYE_H
```

```
00021
00022 #include <sys/cdefs.h>
00023 __BEGIN_DECLS
00024
00025 #include <arch/types.h>
00026 #include <dc/maple.h>
00027
00028 /** \brief  Dreameye status structure.
00029
00030     This structure contains information about the status of the Camera device
00031     and can be fetched with maple_dev_status(). You should not change any of
00032     this information, it should all be considered read-only. Most of the fields
00033     in here are related to image transfers, and messing with them during a
00034     transfer could screw things up.
00035
00036     \headerfile dc/maple/dreameye.h
00037 */
00038 typedef struct dreameye_state {
00039     /** \brief  The number of images on the device. */
00040     int             image_count;
00041
00042     /** \brief  Is the image_count field valid? */
00043     int             image_count_valid;
00044
00045     /** \brief  The number of transfer operations required for the selected
00046                 image. */
00047     int             transfer_count;
00048
00049     /** \brief  Is an image transferring now? */
00050     int             img_transferring;
00051
00052     /** \brief  Storage for image data. */
00053     uint8           *img_buf;
00054
00055     /** \brief  The size of the image in bytes. */
00056     int             img_size;
00057
00058     /** \brief  The image number currently being transferred. */
00059     uint8           img_number;
00060 } dreameye_state_t;
00061
00062 /** \brief  Get the number of images on the device.
00063
00064     This constant is used with the MAPLE_COMMAND_GETCOND command to fetch the
00065     number of images on the device.
00066 */
00067 #define DREAMEYE_GETCOND_NUM_IMAGES     0x81
00068
00069 /** \brief  Get the number of transfers to copy an image.
00070
00071      This constant is used with the MAPLE_COMMAND_GETCOND command to fetch the
00072      number of times a transfer command must be sent to get the image specified.
00073 */
00074 #define DREAMEYE_GETCOND_TRANSFER_COUNT 0x83
00075
00076 /** \brief  Get an image from the device.
00077
00078     This subcommand is used with the MAPLE_COMMAND_CAMCONTROL command to fetch
00079     part of image data from the specified image.
00080 */
00081 #define DREAMEYE_SUBCOMMAND_IMAGEREQ    0x04
00082
00083 /** \brief  Erase an image from the device.
00084
00085     This subcommand is used with the MAPLE_COMMAND_CAMCONTROL command to remove
00086     an image from the device.
00087 */
00088 #define DREAMEYE_SUBCOMMAND_ERASE       0x05
00089
00090 /** \brief  Error return command.
00091
00092      This subcommand is used by the dreameye with the MAPLE_COMMAND_CAMCONTROL
00093      command to indicate an error occurred in a subcommand.
00094 */
00095 #define DREAMEYE_SUBCOMMAND_ERROR       0xFF
00096
00097 /** \brief  Continue transferring an image. */
00098 #define DREAMEYE_IMAGEREQ_CONTINUE      0x00
00099
00100 /** \brief  Start transferring an image from its start. */
00101 #define DREAMEYE_IMAGEREQ_START         0x40
```

```
00102
00103 /** \brief  Get the number of images on the Dreameye.
00104
00105     This function fetches the number of saved images on the specified Dreameye
00106     device. It can be sent to any of the subdevices of the MAPLE_FUNC_CONTROLLER
00107     root device of the Dreameye. When the response comes from the device, the
00108     image_count field of the dreameye_state_t for the specified device will have
00109     the number of images on the device, and image_count_valid will be set to 1.
00110
00111     \param  dev             The device to query.
00112     \param  block           Set to 1 to wait for the Dreameye to respond.
00113     \retval MAPLE_EOK       On success.
00114     \retval MAPLE_ETIMEOUT  The command timed out while blocking.
00115     \retval MAPLE_EAGAIN    Could not send the command to the device, try again.
00116 */
00117 int dreameye_get_image_count(maple_device_t *dev, int block);
00118
00119 /** \brief  Transfer an image from the Dreameye.
00120
00121     This function fetches a single image from the specified Dreameye device.
00122     This function will block, and can take a little while to execute. You must
00123     use the first subdevice of the MAPLE_FUNC_CONTROLLER root device of the
00124     Dreameye as the dev parameter.
00125
00126     \param  dev             The device to get an image from.
00127     \param  image           The image number to download.
00128     \param  data            A pointer to a buffer to store things in. This
00129                             will be allocated by the function and you are
00130                             responsible for freeing the data when you are done.
00131     \param  img_sz          A pointer to storage for the size of the image, in
00132                             bytes.
00133     \retval MAPLE_EOK       On success.
00134     \retval MAPLE_EFAIL     On error.
00135 */
00136 int dreameye_get_image(maple_device_t *dev, uint8 image, uint8 **data,
00137                        int *img_sz);
00138
00139 /** \brief  Erase an image from the Dreameye.
00140
00141     This function erases the specified image from the Dreameye device. This
00142     command can be sent to any of the subdevices of the MAPLE_FUNC_CONTROLLER
00143     root device of the Dreameye.
00144
00145     \param  dev             The device to erase from.
00146     \param  image           The image number to erase (0xFF to erase all).
00147     \param  block           Set to 1 to wait for the Dreameye to respond.
00148     \retval MAPLE_EOK       On success.
00149     \retval MAPLE_EAGAIN    Couldn't send the command, try again.
00150     \retval MAPLE_ETIMEOUT  Timeout on blocking.
00151     \retval MAPLE_EINVALID  Invalid image number specified.
00152 */
00153 int dreameye_erase_image(maple_device_t *dev, uint8 image, int block);
00154
00155 /* \cond */
00156 /* Init / Shutdown */
00157 void dreameye_init(void);
00158 void dreameye_shutdown(void);
00159 /* \endcond */
00160
00161 __END_DECLS
00162
00163 #endif  /* __DC_MAPLE_DREAMEYE_H */
```

9.198 kernel/arch/dreamcast/include/dc/maple/keyboard.h File Reference

Definitions for using the keyboard device.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <dc/maple.h>
```

Data Structures

- struct kbd_keymap_t

  Keyboard keymap.
- struct kbd_cond_t

  Keyboard raw condition structure.
- struct kbd_state_t

  Keyboard status structure.

Macros

- #define KBD_MOD_LCTRL (1<<0)
- #define KBD_MOD_LSHIFT (1<<1)
- #define KBD_MOD_LALT (1<<2)
- #define KBD_MOD_S1 (1<<3)
- #define KBD_MOD_RCTRL (1<<4)
- #define KBD_MOD_RSHIFT (1<<5)
- #define KBD_MOD_RALT (1<<6)
- #define KBD_MOD_S2 (1<<7)
- #define KBD_LED_NUMLOCK (1<<0)
- #define KBD_LED_CAPSLOCK (1<<1)
- #define KBD_LED_SCRLOCK (1<<2)
- #define KBD_KEY_NONE 0x00
- #define KBD_KEY_ERROR 0x01
- #define KBD_KEY_ERR2 0x02
- #define KBD_KEY_ERR3 0x03
- #define KBD_KEY_A 0x04
- #define KBD_KEY_B 0x05
- #define KBD_KEY_C 0x06
- #define KBD_KEY_D 0x07
- #define KBD_KEY_E 0x08
- #define KBD_KEY_F 0x09
- #define KBD_KEY_G 0x0a
- #define KBD_KEY_H 0x0b
- #define KBD_KEY_I 0x0c
- #define KBD_KEY_J 0x0d
- #define KBD_KEY_K 0x0e
- #define KBD_KEY_L 0x0f
- #define KBD_KEY_M 0x10
- #define KBD_KEY_N 0x11
- #define KBD_KEY_O 0x12
- #define KBD_KEY_P 0x13
- #define KBD_KEY_Q 0x14
- #define KBD_KEY_R 0x15
- #define KBD_KEY_S 0x16
- #define KBD_KEY_T 0x17
- #define KBD_KEY_U 0x18
- #define KBD_KEY_V 0x19
- #define KBD_KEY_W 0x1a

- #define KBD_KEY_X 0x1b
- #define KBD_KEY_Y 0x1c
- #define KBD_KEY_Z 0x1d
- #define KBD_KEY_1 0x1e
- #define KBD_KEY_2 0x1f
- #define KBD_KEY_3 0x20
- #define KBD_KEY_4 0x21
- #define KBD_KEY_5 0x22
- #define KBD_KEY_6 0x23
- #define KBD_KEY_7 0x24
- #define KBD_KEY_8 0x25
- #define KBD_KEY_9 0x26
- #define KBD_KEY_0 0x27
- #define KBD_KEY_ENTER 0x28
- #define KBD_KEY_ESCAPE 0x29
- #define KBD_KEY_BACKSPACE 0x2a
- #define KBD_KEY_TAB 0x2b
- #define KBD_KEY_SPACE 0x2c
- #define KBD_KEY_MINUS 0x2d
- #define KBD_KEY_PLUS 0x2e
- #define KBD_KEY_LBRACKET 0x2f
- #define KBD_KEY_RBRACKET 0x30
- #define KBD_KEY_BACKSLASH 0x31
- #define KBD_KEY_SEMICOLON 0x33
- #define KBD_KEY_QUOTE 0x34
- #define KBD_KEY_TILDE 0x35
- #define KBD_KEY_COMMA 0x36
- #define KBD_KEY_PERIOD 0x37
- #define KBD_KEY_SLASH 0x38
- #define KBD_KEY_CAPSLOCK 0x39
- #define KBD_KEY_F1 0x3a
- #define KBD_KEY_F2 0x3b
- #define KBD_KEY_F3 0x3c
- #define KBD_KEY_F4 0x3d
- #define KBD_KEY_F5 0x3e
- #define KBD_KEY_F6 0x3f
- #define KBD_KEY_F7 0x40
- #define KBD_KEY_F8 0x41
- #define KBD_KEY_F9 0x42
- #define KBD_KEY_F10 0x43
- #define KBD_KEY_F11 0x44
- #define KBD_KEY_F12 0x45
- #define KBD_KEY_PRINT 0x46
- #define KBD_KEY_SCRLOCK 0x47
- #define KBD_KEY_PAUSE 0x48
- #define KBD_KEY_INSERT 0x49
- #define KBD_KEY_HOME 0x4a
- #define KBD_KEY_PGUP 0x4b
- #define KBD_KEY_DEL 0x4c
- #define KBD_KEY_END 0x4d
- #define KBD_KEY_PGDOWN 0x4e

- #define KBD_KEY_RIGHT 0x4f
- #define KBD_KEY_LEFT 0x50
- #define KBD_KEY_DOWN 0x51
- #define KBD_KEY_UP 0x52
- #define KBD_KEY_PAD_NUMLOCK 0x53
- #define KBD_KEY_PAD_DIVIDE 0x54
- #define KBD_KEY_PAD_MULTIPLY 0x55
- #define KBD_KEY_PAD_MINUS 0x56
- #define KBD_KEY_PAD_PLUS 0x57
- #define KBD_KEY_PAD_ENTER 0x58
- #define KBD_KEY_PAD_1 0x59
- #define KBD_KEY_PAD_2 0x5a
- #define KBD_KEY_PAD_3 0x5b
- #define KBD_KEY_PAD_4 0x5c
- #define KBD_KEY_PAD_5 0x5d
- #define KBD_KEY_PAD_6 0x5e
- #define KBD_KEY_PAD_7 0x5f
- #define KBD_KEY_PAD_8 0x60
- #define KBD_KEY_PAD_9 0x61
- #define KBD_KEY_PAD_0 0x62
- #define KBD_KEY_PAD_PERIOD 0x63
- #define KBD_KEY_S3 0x65
- #define KBD_REGION_JP 1

  Japanese keyboard.
- #define KBD_REGION_US 2

  US keyboard.
- #define KBD_REGION_UK 3

  UK keyboard.
- #define KBD_REGION_DE 4

  German keyboard.
- #define KBD_REGION_FR 5

  French keyboard (not supported yet)
- #define KBD_REGION_IT 6

  Italian keyboard (not supported yet)
- #define KBD_REGION_ES 7

  Spanish keyboard.
- #define KEY_STATE_NONE 0
- #define KEY_STATE_WAS_PRESSED 1
- #define KEY_STATE_PRESSED 2
- #define MAX_PRESSED_KEYS 6

  Maximum number of keys the DC can read simultaneously. This is a hardware constant. The define prevents the magic number '6' from appearing.
- #define MAX_KBD_KEYS 256

  Maximum number of keys a DC keyboard can have. This is a hardware constant. The define prevents the magic number '256' from appearing.
- #define KBD_QUEUE_SIZE 16

  Size of a keyboard queue.

Functions

- void kbd_set_queue (int active) __attribute__((deprecated))

  Activate or deactivate global key queueing.
- int kbd_get_key (void) __attribute__((deprecated))

  Pop a key off the global keyboard queue.
- int kbd_queue_pop (maple_device_t *dev, int xlat)

  Pop a key off a specific keyboard's queue.

9.198.1 Detailed Description

Definitions for using the keyboard device.

This file contains the definitions needed to access the Maple keyboard device. Obviously, this corresponds to the MAPLE_FUNC_KEYBOARD function code.

Author

Jordan DeLong

Megan Potter

Lawrence Sebald

9.198.2 Macro Definition Documentation

KBD_QUEUE_SIZE

```
#define KBD_QUEUE_SIZE 16
```

Size of a keyboard queue.

Each keyboard queue will hold this many elements. Once the queue fills, no new elements will be placed on the queue. As long as you check the queue relatively frequently, the default of 16 should be plenty.

Note

This **MUST** be a power of two.

MAX_KBD_KEYS

```
#define MAX_KBD_KEYS 256
```

Maximum number of keys a DC keyboard can have. This is a hardware constant. The define prevents the magic number '256' from appearing.

MAX_PRESSED_KEYS

```
#define MAX_PRESSED_KEYS 6
```

Maximum number of keys the DC can read simultaneously. This is a hardware constant. The define prevents the magic number '6' from appearing.

9.198.3 Function Documentation

kbd_get_key()

```
int kbd_get_key (
            void  )
```

Pop a key off the global keyboard queue.

This function pops the front off of the keyboard queue, and returns the value to the caller. The value returned will be the ASCII value of the key pressed (accounting for the shift keys being pressed).

If a key does not have an ASCII value associated with it, the raw key code will be returned, shifted up by 8 bits.

Returns

The value at the front of the queue, or -1 if there are no keys in the queue or queueing is off.

Note

This function does not account for non-US keyboard layouts properly (for compatibility with old code), and is deprecated. Use the individual keyboard queues instead to properly account for non-US keyboards.

See also

kbd_queue_pop()

kbd_queue_pop()

```
int kbd_queue_pop (
            maple_device_t * dev,
            int xlat )
```

Pop a key off a specific keyboard's queue.

This function pops the front element off of the specified keyboard queue, and returns the value of that key to the caller.

If the xlat parameter is non-zero and the key represents an ISO-8859-1 character, that is the value that will be returned from this function. Otherwise if xlat is non-zero, it will be the raw key code, shifted up by 8 bits.

If the xlat parameter is zero, the lower 8 bits of the returned value will be the raw key code. The next 8 bits will be the modifier keys that were down when the key was pressed (a bitfield of KBD_MOD_* values). The next 3 bits will be the lock key status (a bitfield of KBD_LED_* values).

Parameters

dev	The keyboard device to read from.
xlat	Set to non-zero to do key translation. Otherwise, you'll simply get the raw key value. Raw key values are not mapped at all, so you are responsible for figuring out what it is by the region.

Returns

The value at the front of the queue, or -1 if there are no keys in the queue.

kbd_set_queue()

```
void kbd_set_queue (
            int active )
```

Activate or deactivate global key queueing.

This function will turn the internal keyboard queueing on or off. Note that there is only one queue for the whole system, no matter how many keyboards are attached, and the queue is of fairly limited length. Turning queueing off is useful (for instance) in a game where individual keypresses don't mean as much as having the keys up or down does.

You can clear the queue (without popping all the keys off) by setting the active value to a different value than it was.

The queue is by default on, unless you turn it off.

Parameters

active	Set to non-zero to activate the queue.

Note

The global queue does not account for non-US keyboard layouts and is deprecated. Please use the individual queues instead for future code.

9.199 keyboard.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/maple/keyboard.h
00004    Copyright (C) 2000-2002 Jordan DeLong and Megan Potter
00005    Copyright (C) 2012 Lawrence Sebald
00006
00007 */
00008
00009 /** \file   dc/maple/keyboard.h
00010     \brief  Definitions for using the keyboard device.
00011
00012     This file contains the definitions needed to access the Maple keyboard
00013     device. Obviously, this corresponds to the MAPLE_FUNC_KEYBOARD function
00014     code.
```

```
00015
00016     \author Jordan DeLong
00017     \author Megan Potter
00018     \author Lawrence Sebald
00019 */
00020
00021 #ifndef __DC_MAPLE_KEYBOARD_H
00022 #define __DC_MAPLE_KEYBOARD_H
00023
00024 #include <sys/cdefs.h>
00025 __BEGIN_DECLS
00026
00027 #include <arch/types.h>
00028 #include <dc/maple.h>
00029
00030 /** \defgroup   kbd_mods    Keyboard modifier keys
00031
00032     These are the various modifiers that can be pressed on the keyboard, and are
00033     reflected in the modifiers field of kbd_cond_t.
00034     @{
00035 */
00036 #define KBD_MOD_LCTRL       (1<<0)
00037 #define KBD_MOD_LSHIFT      (1<<1)
00038 #define KBD_MOD_LALT        (1<<2)
00039 #define KBD_MOD_S1          (1<<3)
00040 #define KBD_MOD_RCTRL       (1<<4)
00041 #define KBD_MOD_RSHIFT      (1<<5)
00042 #define KBD_MOD_RALT        (1<<6)
00043 #define KBD_MOD_S2          (1<<7)
00044 /** @} */
00045
00046 /** \defgroup   kbd_leds    Keyboard LEDs
00047
00048     This is the LEDs that can be turned on and off on the keyboard. This list
00049     may not be exhaustive. Think of these sorta like an extension of the
00050     modifiers list.
00051     @{
00052 */
00053 #define KBD_LED_NUMLOCK     (1<<0)
00054 #define KBD_LED_CAPSLOCK    (1<<1)
00055 #define KBD_LED_SCRLOCK     (1<<2)
00056 /** @} */
00057
00058 /** \defgroup   kbd_keys    Keyboard keys
00059
00060     This is the list of keys that are on the keyboard that may be pressed. The
00061     keyboard returns keys in this format.
00062
00063     These are the raw keycodes returned by the US keyboard, and thus only cover
00064     the keys on US keyboards.
00065     @{
00066 */
00067 #define KBD_KEY_NONE            0x00
00068 #define KBD_KEY_ERROR           0x01
00069 #define KBD_KEY_ERR2            0x02
00070 #define KBD_KEY_ERR3            0x03
00071 #define KBD_KEY_A               0x04
00072 #define KBD_KEY_B               0x05
00073 #define KBD_KEY_C               0x06
00074 #define KBD_KEY_D               0x07
00075 #define KBD_KEY_E               0x08
00076 #define KBD_KEY_F               0x09
00077 #define KBD_KEY_G               0x0a
00078 #define KBD_KEY_H               0x0b
00079 #define KBD_KEY_I               0x0c
00080 #define KBD_KEY_J               0x0d
00081 #define KBD_KEY_K               0x0e
00082 #define KBD_KEY_L               0x0f
00083 #define KBD_KEY_M               0x10
00084 #define KBD_KEY_N               0x11
00085 #define KBD_KEY_O               0x12
00086 #define KBD_KEY_P               0x13
00087 #define KBD_KEY_Q               0x14
00088 #define KBD_KEY_R               0x15
00089 #define KBD_KEY_S               0x16
00090 #define KBD_KEY_T               0x17
00091 #define KBD_KEY_U               0x18
00092 #define KBD_KEY_V               0x19
00093 #define KBD_KEY_W               0x1a
00094 #define KBD_KEY_X               0x1b
00095 #define KBD_KEY_Y               0x1c
```

```
00096 #define KBD_KEY_Z               0x1d
00097 #define KBD_KEY_1               0x1e
00098 #define KBD_KEY_2               0x1f
00099 #define KBD_KEY_3               0x20
00100 #define KBD_KEY_4               0x21
00101 #define KBD_KEY_5               0x22
00102 #define KBD_KEY_6               0x23
00103 #define KBD_KEY_7               0x24
00104 #define KBD_KEY_8               0x25
00105 #define KBD_KEY_9               0x26
00106 #define KBD_KEY_0               0x27
00107 #define KBD_KEY_ENTER           0x28
00108 #define KBD_KEY_ESCAPE          0x29
00109 #define KBD_KEY_BACKSPACE       0x2a
00110 #define KBD_KEY_TAB             0x2b
00111 #define KBD_KEY_SPACE           0x2c
00112 #define KBD_KEY_MINUS           0x2d
00113 #define KBD_KEY_PLUS            0x2e
00114 #define KBD_KEY_LBRACKET        0x2f
00115 #define KBD_KEY_RBRACKET        0x30
00116 #define KBD_KEY_BACKSLASH       0x31
00117 #define KBD_KEY_SEMICOLON       0x33
00118 #define KBD_KEY_QUOTE           0x34
00119 #define KBD_KEY_TILDE           0x35
00120 #define KBD_KEY_COMMA           0x36
00121 #define KBD_KEY_PERIOD          0x37
00122 #define KBD_KEY_SLASH           0x38
00123 #define KBD_KEY_CAPSLOCK        0x39
00124 #define KBD_KEY_F1              0x3a
00125 #define KBD_KEY_F2              0x3b
00126 #define KBD_KEY_F3              0x3c
00127 #define KBD_KEY_F4              0x3d
00128 #define KBD_KEY_F5              0x3e
00129 #define KBD_KEY_F6              0x3f
00130 #define KBD_KEY_F7              0x40
00131 #define KBD_KEY_F8              0x41
00132 #define KBD_KEY_F9              0x42
00133 #define KBD_KEY_F10             0x43
00134 #define KBD_KEY_F11             0x44
00135 #define KBD_KEY_F12             0x45
00136 #define KBD_KEY_PRINT           0x46
00137 #define KBD_KEY_SCRLOCK         0x47
00138 #define KBD_KEY_PAUSE           0x48
00139 #define KBD_KEY_INSERT          0x49
00140 #define KBD_KEY_HOME            0x4a
00141 #define KBD_KEY_PGUP            0x4b
00142 #define KBD_KEY_DEL             0x4c
00143 #define KBD_KEY_END             0x4d
00144 #define KBD_KEY_PGDOWN          0x4e
00145 #define KBD_KEY_RIGHT           0x4f
00146 #define KBD_KEY_LEFT            0x50
00147 #define KBD_KEY_DOWN            0x51
00148 #define KBD_KEY_UP              0x52
00149 #define KBD_KEY_PAD_NUMLOCK     0x53
00150 #define KBD_KEY_PAD_DIVIDE      0x54
00151 #define KBD_KEY_PAD_MULTIPLY    0x55
00152 #define KBD_KEY_PAD_MINUS       0x56
00153 #define KBD_KEY_PAD_PLUS        0x57
00154 #define KBD_KEY_PAD_ENTER       0x58
00155 #define KBD_KEY_PAD_1           0x59
00156 #define KBD_KEY_PAD_2           0x5a
00157 #define KBD_KEY_PAD_3           0x5b
00158 #define KBD_KEY_PAD_4           0x5c
00159 #define KBD_KEY_PAD_5           0x5d
00160 #define KBD_KEY_PAD_6           0x5e
00161 #define KBD_KEY_PAD_7           0x5f
00162 #define KBD_KEY_PAD_8           0x60
00163 #define KBD_KEY_PAD_9           0x61
00164 #define KBD_KEY_PAD_0           0x62
00165 #define KBD_KEY_PAD_PERIOD      0x63
00166 #define KBD_KEY_S3              0x65
00167 /** @} */
00168
00169 /** \defgroup   kbd_regions Keyboard region codes
00170
00171     This is the list of possible values for the "region" field in the
00172     kbd_state_t structure.
00173     @{
00174 */
00175 #define KBD_REGION_JP       1           /**< \brief Japanese keyboard */
00176 #define KBD_REGION_US       2           /**< \brief US keyboard */
```

```
00177 #define KBD_REGION_UK       3           /**< \brief UK keyboard */
00178 #define KBD_REGION_DE       4           /**< \brief German keyboard */
00179 #define KBD_REGION_FR       5           /**< \brief French keyboard (not supported yet) */
00180 #define KBD_REGION_IT       6           /**< \brief Italian keyboard (not supported yet) */
00181 #define KBD_REGION_ES       7           /**< \brief Spanish keyboard */
00182 /** @} */
00183
00184 /** \defgroup   key_states  States each key can be in.
00185
00186     These are the different 'states' each key can be in. They are stored in
00187     kbd_state_t->matrix, and manipulated/checked by kbd_check_poll.
00188
00189     none-> pressed or none
00190     was pressed-> pressed or none
00191     pressed-> was_pressed
00192     @{
00193 */
00194 #define KEY_STATE_NONE        0
00195 #define KEY_STATE_WAS_PRESSED 1
00196 #define KEY_STATE_PRESSED     2
00197 /** @} */
00198
00199 /** \brief Maximum number of keys the DC can read simultaneously.
00200     This is a hardware constant. The define prevents the magic number '6' from appearing.
00201 **/
00202 #define MAX_PRESSED_KEYS 6
00203
00204 /** \brief Maximum number of keys a DC keyboard can have.
00205     This is a hardware constant. The define prevents the magic number '256' from appearing.
00206 **/
00207 #define MAX_KBD_KEYS 256
00208
00209 /** \brief  Size of a keyboard queue.
00210
00211     Each keyboard queue will hold this many elements. Once the queue fills, no
00212     new elements will be placed on the queue. As long as you check the queue
00213     relatively frequently, the default of 16 should be plenty.
00214
00215     \note   This <strong>MUST</strong> be a power of two.
00216 */
00217 #define KBD_QUEUE_SIZE 16
00218
00219 /** \brief  Keyboard keymap.
00220
00221     This structure represents a mapping from raw key values to ASCII values, if
00222     appropriate. This handles base values as well as shifted ("shift" and "Alt"
00223     keys) values.
00224
00225     \headerfile dc/maple/keyboard.h
00226 */
00227 typedef struct kbd_keymap {
00228     uint8 base[MAX_KBD_KEYS];
00229     uint8 shifted[MAX_KBD_KEYS];
00230     uint8 alt[MAX_KBD_KEYS];
00231 } kbd_keymap_t;
00232
00233 /** \brief  Keyboard raw condition structure.
00234
00235     This structure is what the keyboard responds with as its current status.
00236
00237     \headerfile dc/maple/keyboard.h
00238 */
00239 typedef struct {
00240     uint8 modifiers;    /**< \brief Bitmask of set modifiers. */
00241     uint8 leds;         /**< \brief Bitmask of set LEDs */
00242     uint8 keys[MAX_PRESSED_KEYS];      /**< \brief Key codes for currently pressed keys. */
00243 } kbd_cond_t;
00244
00245 /** \brief  Keyboard status structure.
00246
00247     This structure holds information about the current status of the keyboard
00248     device. This is what maple_dev_status() will return.
00249
00250     \headerfile dc/maple/keyboard.h
00251 */
00252 typedef struct kbd_state {
00253     /** \brief  The lastest raw condition of the keyboard. */
00254     kbd_cond_t cond;
00255
00256     /** \brief  Key array.
00257
```

```
00258         This array lists the state of all possible keys on the keyboard. It can
00259         be used for key repeat and debouncing. This will be non-zero if the key
00260         is currently being pressed.
00261
00262         \see    kbd_keys
00263     */
00264     uint8 matrix[MAX_KBD_KEYS];
00265
00266     /** \brief  Modifier key status. */
00267     int shift_keys;
00268
00269     /** \brief  Keyboard type/region. */
00270     int region;
00271
00272     /** \brief  Individual keyboard queue.
00273         You should not access this variable directly. Please use the appropriate
00274         function to access it. */
00275     uint32 key_queue[KBD_QUEUE_SIZE];
00276     int queue_tail;                     /**< \brief Key queue tail. */
00277     int queue_head;                     /**< \brief Key queue head. */
00278     int queue_len;                      /**< \brief Current length of queue. */
00279
00280     uint8 kbd_repeat_key;           /**< \brief Key that is repeating. */
00281     uint64 kbd_repeat_timer;        /**< \brief Time that the next repeat will trigger. */
00282 } kbd_state_t;
00283
00284 /** \brief  Activate or deactivate global key queueing.
00285
00286     This function will turn the internal keyboard queueing on or off. Note that
00287     there is only one queue for the whole system, no matter how many keyboards
00288     are attached, and the queue is of fairly limited length. Turning queueing
00289     off is useful (for instance) in a game where individual keypresses don't
00290     mean as much as having the keys up or down does.
00291
00292     You can clear the queue (without popping all the keys off) by setting the
00293     active value to a different value than it was.
00294
00295     The queue is by default on, unless you turn it off.
00296
00297     \param  active          Set to non-zero to activate the queue.
00298     \note                   The global queue does not account for non-US
00299                             keyboard layouts and is deprecated. Please use the
00300                             individual queues instead for future code.
00301 */
00302 void kbd_set_queue(int active) __attribute__((deprecated));
00303
00304 /** \brief  Pop a key off the global keyboard queue.
00305
00306     This function pops the front off of the keyboard queue, and returns the
00307     value to the caller. The value returned will be the ASCII value of the key
00308     pressed (accounting for the shift keys being pressed).
00309
00310     If a key does not have an ASCII value associated with it, the raw key code
00311     will be returned, shifted up by 8 bits.
00312
00313     \return                 The value at the front of the queue, or -1 if there
00314                             are no keys in the queue or queueing is off.
00315     \note                   This function does not account for non-US keyboard
00316                             layouts properly (for compatibility with old code),
00317                             and is deprecated. Use the individual keyboard
00318                             queues instead to properly account for non-US
00319                             keyboards.
00320     \see                    kbd_queue_pop()
00321 */
00322 int kbd_get_key(void) __attribute__((deprecated));
00323
00324 /** \brief  Pop a key off a specific keyboard's queue.
00325
00326     This function pops the front element off of the specified keyboard queue,
00327     and returns the value of that key to the caller.
00328
00329     If the xlat parameter is non-zero and the key represents an ISO-8859-1
00330     character, that is the value that will be returned from this function.
00331     Otherwise if xlat is non-zero, it will be the raw key code, shifted up by 8
00332     bits.
00333
00334     If the xlat parameter is zero, the lower 8 bits of the returned value will
00335     be the raw key code. The next 8 bits will be the modifier keys that were
00336     down when the key was pressed (a bitfield of KBD_MOD_* values). The next 3
00337     bits will be the lock key status (a bitfield of KBD_LED_* values).
00338
```

```
00339     \param  dev             The keyboard device to read from.
00340     \param  xlat            Set to non-zero to do key translation. Otherwise,
00341                             you'll simply get the raw key value. Raw key values
00342                             are not mapped at all, so you are responsible for
00343                             figuring out what it is by the region.
00344
00345     \return                 The value at the front of the queue, or -1 if there
00346                             are no keys in the queue.
00347 */
00348 int kbd_queue_pop(maple_device_t *dev, int xlat);
00349
00350 /* \cond */
00351 /* Init / Shutdown */
00352 void kbd_init(void);
00353 void kbd_shutdown(void);
00354 /* \endcond */
00355
00356 __END_DECLS
00357
00358 #endif  /* __DC_MAPLE_KEYBOARD_H */
```

9.200 kernel/arch/dreamcast/include/dc/maple/lightgun.h File Reference

Definitions for using the light gun.

```
#include <sys/cdefs.h>
```

9.200.1 Detailed Description

Definitions for using the light gun.

This file contains the definitions needed to access maple light gun devices. There's really no user-serviceable parts in here, as there's very little to this driver.

Author

Lawrence Sebald

9.201 lightgun.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/maple/lightgun.h
00004    Copyright (C) 2015 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   dc/maple/lightgun.h
00009     \brief  Definitions for using the light gun.
00010
00011     This file contains the definitions needed to access maple light gun devices.
00012     There's really no user-serviceable parts in here, as there's very little to
00013     this driver.
00014
00015     \author Lawrence Sebald
00016 */
00017
00018 #ifndef __DC_MAPLE_LIGHTGUN_H
00019 #define __DC_MAPLE_LIGHTGUN_H
00020
```

```
00021 #include <sys/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 /* \cond */
00025 /* Init / Shutdown */
00026 void lightgun_init(void);
00027 void lightgun_shutdown(void);
00028 /* \endcond */
00029
00030 __END_DECLS
00031
00032 #endif  /* __DC_MAPLE_LIGHTGUN_H */
```

9.202 kernel/arch/dreamcast/include/dc/maple/mouse.h File Reference

Definitions for using the mouse device.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Data Structures

- struct mouse_cond_t
- struct mouse_state_t

  Mouse status structure.

Macros

- #define MOUSE_RIGHTBUTTON (1<<1)

  Right mouse button.
- #define MOUSE_LEFTBUTTON (1<<2)

  Left mouse button.
- #define MOUSE_SIDEBUTTON (1<<3)

  Side mouse button.
- #define MOUSE_DELTA_CENTER 0x200

  Mouse center value in the raw condition structure.

9.202.1 Detailed Description

Definitions for using the mouse device.

This file contains the definitions needed to access the Maple mouse type device.

Author

Jordan DeLong

Megan Potter

9.202.2 Macro Definition Documentation

MOUSE_DELTA_CENTER

```
#define MOUSE_DELTA_CENTER 0x200
```

Mouse center value in the raw condition structure.

9.203 mouse.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/maple/mouse.h
00004    (C)2000-2002 Jordan DeLong and Megan Potter
00005
00006 */
00007
00008 /** \file   dc/maple/mouse.h
00009     \brief  Definitions for using the mouse device.
00010
00011     This file contains the definitions needed to access the Maple mouse type
00012     device.
00013
00014     \author Jordan DeLong
00015     \author Megan Potter
00016 */
00017
00018 #ifndef __DC_MAPLE_MOUSE_H
00019 #define __DC_MAPLE_MOUSE_H
00020
00021 #include <sys/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 #include <arch/types.h>
00025
00026 /** \defgroup   mouse_buttons   Mouse button codes
00027
00028     These are the possible buttons to press on a maple bus mouse.
00029
00030     @{
00031 */
00032 #define MOUSE_RIGHTBUTTON   (1<<1)  /**< \brief Right mouse button */
00033 #define MOUSE_LEFTBUTTON    (1<<2)  /**< \brief Left mouse button */
00034 #define MOUSE_SIDEBUTTON    (1<<3)  /**< \brief Side mouse button */
00035 /** @} */
00036
00037 /** \brief  Mouse center value in the raw condition structure. */
00038 #define MOUSE_DELTA_CENTER      0x200
00039
00040 /* Raw mouse condition structure */
00041 typedef struct {
00042     uint16 buttons;
00043     uint16 dummy1;
00044     int16 dx;
00045     int16 dy;
00046     int16 dz;
00047     uint16 dummy2;
00048     uint32 dummy3;
00049     uint32 dummy4;
00050 } mouse_cond_t;
00051
00052 /* More civilized mouse structure. There are several significant
00053    differences in data interpretation between the "cooked" and
00054    the old "raw" structs:
00055
00056    - buttons are zero-based: a 1-bit means the button is PRESSED
00057    - no dummy values
00058
00059    Note that this is what maple_dev_status() will return.
00060  */
```

```
00061
00062 /** \brief  Mouse status structure.
00063
00064     This structure contains information about the status of the mouse device,
00065     and can be fetched with maple_dev_status().
00066
00067     \headerfile dc/maple/mouse.h
00068 */
00069 typedef struct {
00070     /** \brief  Buttons pressed bitmask.
00071         \see    mouse_buttons
00072     */
00073     uint32  buttons;
00074
00075     /** \brief  X movement value */
00076     int dx;
00077
00078     /** \brief  Y movement value */
00079     int dy;
00080
00081     /** \brief  Z movement value */
00082     int dz;
00083 } mouse_state_t;
00084
00085 /* \cond */
00086 /* Init / Shutdown */
00087 void mouse_init(void);
00088 void mouse_shutdown(void);
00089 /* \endcond */
00090
00091 __END_DECLS
00092
00093 #endif  /* __DC_MAPLE_MOUSE_H */
00094
```

9.204 kernel/arch/dreamcast/include/dc/maple/purupuru.h File Reference

Definitions for using the Puru Puru (Jump) Pack.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <dc/maple.h>
```

Data Structures

- struct purupuru_effect_t

  Effect generation structure.

Macros

- #define PURUPURU_EFFECT2_UINTENSITY(x) (x << 4)

  Upper-nibble of effect2 convenience macro.
- #define PURUPURU_EFFECT2_LINTENSITY(x) (x)

  Lower-nibble of effect2 convenience macro.
- #define PURUPURU_EFFECT2_DECAY (8 << 4)

  Give a decay effect to the rumble on some packs.
- #define PURUPURU_EFFECT2_PULSE (8)

  Give a pulse effect to the rumble.

- #define PURUPURU_EFFECT1_INTENSITY(x) (x << 4)

  Upper nibble of effect1 convenience macro.
- #define PURUPURU_EFFECT1_PULSE (8 << 4)

  Give a pulse effect to the rumble.
- #define PURUPURU_EFFECT1_POWERSAVE (15)

  Ignore this command.
- #define PURUPURU_SPECIAL_MOTOR1 (1 << 4)

  Select motor #1.
- #define PURUPURU_SPECIAL_MOTOR2 (1 << 7)

  Select motor #2.
- #define PURUPURU_SPECIAL_PULSE (1)

  Yet another pulse effect.

Functions

- int purupuru_rumble (maple_device_t *dev, purupuru_effect_t *effect)

  Send an effect to a jump pack.
- int purupuru_rumble_raw (maple_device_t *dev, uint32 effect)

  Send a raw effect to a jump pack.

9.204.1 Detailed Description

Definitions for using the Puru Puru (Jump) Pack.

This file contains the definitions needed to access maple jump pack devices. Puru Puru was Sega's internal name for the device, hence why its referred to in this way here.

This driver is largely based off of information provided by Kamjin on the DCEmulation forums. See http://dcemulation.org/phpBB/viewtopic.php?f=29&t=48462 if you're interested in the original documentation.

Also, its important to note that not all Jump Packs are created equal. Some of the stuff in here does not do what it seems like it should on many devices. The "decay" setting, for instance, does not seem to work on Sega Puru Purus, and actually makes most (if not all) effects do absolutely nothing. Basically, its all a big guess-and-test game to get things to work the way you might like. Don't be surprised if you manage to set up something that does absolutely nothing on the first try.

Author

Lawrence Sebald

9.204.2 Macro Definition Documentation

PURUPURU_EFFECT1_INTENSITY

```
#define PURUPURU_EFFECT1_INTENSITY(
              x ) (x << 4)
```

Upper nibble of effect1 convenience macro.

This macro is for setting the upper nibble of the effect1 field of the purupuru_effect_t. This value works with the lower nibble of the effect2 field to increase the intensity of the rumble effect. Valid values are 0-7.

See also

PURUPURU_EFFECT2_LINTENSITY

PURUPURU_EFFECT1_POWERSAVE

```
#define PURUPURU_EFFECT1_POWERSAVE (15)
```

Ignore this command.

Most jump packs will ignore commands with this set in effect1, apparently.

PURUPURU_EFFECT1_PULSE

```
#define PURUPURU_EFFECT1_PULSE (8 << 4)
```

Give a pulse effect to the rumble.

This probably should be used with PURUPURU_EFFECT2_PULSE as well.

See also

PURUPURU_EFFECT2_PULSE

PURUPURU_EFFECT2_DECAY

```
#define PURUPURU_EFFECT2_DECAY (8 << 4)
```

Give a decay effect to the rumble on some packs.

PURUPURU_EFFECT2_LINTENSITY

```
#define PURUPURU_EFFECT2_LINTENSITY(
              x ) (x)
```

Lower-nibble of effect2 convenience macro.

This macro is for setting the lower nibble of the effect2 field of the purupuru_effect_t. This value works with the upper nibble of the effect1 field to increase the intensity of the rumble effect. Valid values are 0-7.

See also

PURUPURU_EFFECT1_INTENSITY

PURUPURU_EFFECT2_PULSE

```
#define PURUPURU_EFFECT2_PULSE (8)
```

Give a pulse effect to the rumble.

This probably should be used with PURUPURU_EFFECT1_PULSE as well.

See also

PURUPURU_EFFECT1_PULSE

PURUPURU_EFFECT2_UINTENSITY

```
#define PURUPURU_EFFECT2_UINTENSITY(
              x ) (x << 4)
```

Upper-nibble of effect2 convenience macro.

This macro is for setting the upper nibble of the effect2 field of the purupuru_effect_t. This apparently lowers the rumble's intensity somewhat. Valid values are 0-7.

PURUPURU_SPECIAL_MOTOR1

```
#define PURUPURU_SPECIAL_MOTOR1 (1 << 4)
```

Select motor #1.

Most jump packs only have one motor, but on things that do have more than one motor (like PS1->Dreamcast controller adapters that support rumble), this selects the first motor.

PURUPURU_SPECIAL_MOTOR2

```
#define PURUPURU_SPECIAL_MOTOR2 (1 << 7)
```

Select motor #2.

Most jump packs only have one motor, but on things that do have more than one motor (like PS1->Dreamcast controller adapters that support rumble), this selects the second motor.

PURUPURU_SPECIAL_PULSE

```
#define PURUPURU_SPECIAL_PULSE (1)
```

Yet another pulse effect.

This supposedly creates a sharp pulse effect.

9.204.3 Function Documentation

purupuru_rumble()

```
int purupuru_rumble (
            maple_device_t * dev,
            purupuru_effect_t * effect )
```

Send an effect to a jump pack.

This function sends an effect created with the purupuru_effect_t structure to a jump pack to be executed.

Parameters

dev	The device to send the command to.
effect	The effect to send.

Return values

MAPLE_EOK	On success.
MAPLE_EAGAIN	If the command couldn't be sent. Try again later.
MAPLE_ETIMEOUT	If the command timed out while blocking.

purupuru_rumble_raw()

```
int purupuru_rumble_raw (
            maple_device_t * dev,
            uint32 effect )
```

Send a raw effect to a jump pack.

This function sends an effect to a jump pack to be executed. This is for if you (for some reason) don't want to use purupuru_effect_t to build the effect up.

Parameters

dev	The device to send the command to.
effect	The effect to send.

Return values

MAPLE_EOK	On success.
MAPLE_EAGAIN	If the command couldn't be sent. Try again later.
MAPLE_ETIMEOUT	If the command timed out while blocking.

9.205 purupuru.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/maple/purupuru.h
00004    Copyright (C) 2003 Megan Potter
00005    Copyright (C) 2005, 2010 Lawrence Sebald
00006
00007 */
00008
00009 /** \file   dc/maple/purupuru.h
00010     \brief  Definitions for using the Puru Puru (Jump) Pack.
00011
00012     This file contains the definitions needed to access maple jump pack devices.
00013     Puru Puru was Sega's internal name for the device, hence why its referred to
00014     in this way here.
00015
00016     This driver is largely based off of information provided by Kamjin on the
00017     DCEmulation forums. See
00018     http://dcemulation.org/phpBB/viewtopic.php?f=29&t=48462 if you're interested
00019     in the original documentation.
00020
00021     Also, its important to note that not all Jump Packs are created equal. Some
00022     of the stuff in here does not do what it seems like it should on many
00023     devices. The "decay" setting, for instance, does not seem to work on Sega
00024     Puru Purus, and actually makes most (if not all) effects do absolutely
00025     nothing. Basically, its all a big guess-and-test game to get things to work
00026     the way you might like. Don't be surprised if you manage to set up something
00027     that does absolutely nothing on the first try.
00028
00029     \author Lawrence Sebald
00030 */
00031
00032 #ifndef __DC_MAPLE_PURUPURU_H
00033 #define __DC_MAPLE_PURUPURU_H
00034
00035 #include <sys/cdefs.h>
00036 __BEGIN_DECLS
00037
00038 #include <arch/types.h>
00039 #include <dc/maple.h>
00040
00041 /** \brief  Effect generation structure.
00042
00043     This structure is used for convenience to send an effect to the jump pack.
00044     This, along with the various macros in this file can give a slightly better
00045     idea of the effect being generated than using the raw values.
```

```
00046 */
00047 typedef struct purupuru_effect  {
00048     /** \brief  The duration of the effect. No idea on units... */
00049     uint8 duration;
00050 
00051     /** \brief  2nd effect field. */
00052     uint8 effect2;
00053 
00054     /** \brief  1st effect field. */
00055     uint8 effect1;
00056 
00057     /** \brief  Special effects field. */
00058     uint8 special;
00059 } purupuru_effect_t;
00060 
00061 /* Set one of each of the following in the effect2 field of the
00062    purupuru_effect_t. Valid values for each are 0-7. The LINTENSITY
00063    value works with the INTENSITY of effect1 to increase the intensity
00064    of the rumble, where UINTENSITY apparently lowers the rumble's
00065    intensity somewhat. */
00066 
00067 /** \brief  Upper-nibble of effect2 convenience macro.
00068 
00069     This macro is for setting the upper nibble of the effect2 field of the
00070     purupuru_effect_t. This apparently lowers the rumble's intensity somewhat.
00071     Valid values are 0-7.
00072 */
00073 #define PURUPURU_EFFECT2_UINTENSITY(x) (x << 4)
00074 
00075 /** \brief  Lower-nibble of effect2 convenience macro.
00076 
00077     This macro is for setting the lower nibble of the effect2 field of the
00078     purupuru_effect_t. This value works with the upper nibble of the effect1
00079     field to increase the intensity of the rumble effect. Valid values are 0-7.
00080 
00081     \see    PURUPURU_EFFECT1_INTENSITY
00082 */
00083 #define PURUPURU_EFFECT2_LINTENSITY(x) (x)
00084 
00085 /* OR these in with your effect2 value if you feel so inclined.
00086    if you or the PULSE effect in here, you probably should also
00087    do so with the effect1 one below. */
00088 /** \brief  Give a decay effect to the rumble on some packs. */
00089 #define PURUPURU_EFFECT2_DECAY          (8 << 4)
00090 
00091 /** \brief  Give a pulse effect to the rumble.
00092 
00093     This probably should be used with PURUPURU_EFFECT1_PULSE as well.
00094 
00095     \see    PURUPURU_EFFECT1_PULSE
00096 */
00097 #define PURUPURU_EFFECT2_PULSE          (8)
00098 
00099 /* Set one value for this in the effect1 field of the effect structure. */
00100 /** \brief  Upper nibble of effect1 convenience macro.
00101 
00102     This macro is for setting the upper nibble of the effect1 field of the
00103     purupuru_effect_t. This value works with the lower nibble of the effect2
00104     field to increase the intensity of the rumble effect. Valid values are 0-7.
00105 
00106     \see    PURUPURU_EFFECT2_LINTENSITY
00107 */
00108 #define PURUPURU_EFFECT1_INTENSITY(x)   (x << 4)
00109 
00110 /* OR these in with your effect1 value, if you need them. PULSE
00111    should probably be used with the PULSE in effect2, as well.
00112    POWERSAVE will probably make your purupuru ignore that command. */
00113 /** \brief  Give a pulse effect to the rumble.
00114 
00115     This probably should be used with PURUPURU_EFFECT2_PULSE as well.
00116 
00117     \see    PURUPURU_EFFECT2_PULSE
00118 */
00119 #define PURUPURU_EFFECT1_PULSE          (8 << 4)
00120 
00121 /** \brief  Ignore this command.
00122 
00123     Most jump packs will ignore commands with this set in effect1, apparently.
00124 */
00125 #define PURUPURU_EFFECT1_POWERSAVE      (15)
00126 
```

```
00127 /* Special Effects and motor select. The normal purupuru packs will
00128    only have one motor. Selecting MOTOR2 for these is probably not
00129    a good idea. The PULSE setting here supposably creates a sharp
00130    pulse effect, when ORed with the special field. */
00131 /** \brief  Select motor #1.
00132
00133     Most jump packs only have one motor, but on things that do have more than
00134     one motor (like PS1->Dreamcast controller adapters that support rumble),
00135     this selects the first motor.
00136 */
00137 #define PURUPURU_SPECIAL_MOTOR1         (1 « 4)
00138
00139 /** \brief  Select motor #2.
00140
00141     Most jump packs only have one motor, but on things that do have more than
00142     one motor (like PS1->Dreamcast controller adapters that support rumble),
00143     this selects the second motor.
00144 */
00145 #define PURUPURU_SPECIAL_MOTOR2         (1 « 7)
00146
00147 /** \brief  Yet another pulse effect.
00148
00149     This supposedly creates a sharp pulse effect.
00150 */
00151 #define PURUPURU_SPECIAL_PULSE          (1)
00152
00153 /** \brief  Send an effect to a jump pack.
00154
00155     This function sends an effect created with the purupuru_effect_t structure
00156     to a jump pack to be executed.
00157
00158     \param  dev             The device to send the command to.
00159     \param  effect          The effect to send.
00160     \retval MAPLE_EOK       On success.
00161     \retval MAPLE_EAGAIN    If the command couldn't be sent. Try again later.
00162     \retval MAPLE_ETIMEOUT  If the command timed out while blocking.
00163 */
00164 int purupuru_rumble(maple_device_t *dev, purupuru_effect_t *effect);
00165
00166 /** \brief  Send a raw effect to a jump pack.
00167
00168     This function sends an effect to a jump pack to be executed. This is for if
00169     you (for some reason) don't want to use purupuru_effect_t to build the
00170     effect up.
00171
00172     \param  dev             The device to send the command to.
00173     \param  effect          The effect to send.
00174     \retval MAPLE_EOK       On success.
00175     \retval MAPLE_EAGAIN    If the command couldn't be sent. Try again later.
00176     \retval MAPLE_ETIMEOUT  If the command timed out while blocking.
00177 */
00178 int purupuru_rumble_raw(maple_device_t *dev, uint32 effect);
00179
00180 /* \cond */
00181 /* Init / Shutdown */
00182 void purupuru_init(void);
00183 void purupuru_shutdown(void);
00184 /* \endcond */
00185
00186 __END_DECLS
00187
00188 #endif  /* __DC_MAPLE_PURUPURU_H */
00189
```

9.206 kernel/arch/dreamcast/include/dc/maple/sip.h File Reference

Definitions for using the Sound Input Peripheral.

```
#include <sys/cdefs.h>
#include <sys/types.h>
#include <dc/maple.h>
```

Data Structures

- struct sip_state_t

  SIP status structure.

Macros

- #define SIP_SUBCOMMAND_GET_SAMPLES 0x01

  Get recorded samples from the microphone device.
- #define SIP_SUBCOMMAND_BASIC_CTRL 0x02

  Start and stop sampling.
- #define SIP_MIN_GAIN 0x00

  Minimum microphone gain.
- #define SIP_DEFAULT_GAIN 0x0F

  Default microphone gain.
- #define SIP_MAX_GAIN 0x1F

  Maximum microphone gain.
- #define SIP_SAMPLE_16BIT_SIGNED 0x00

  Record 16-bit signed integer samples.
- #define SIP_SAMPLE_8BIT_ULAW 0x01

  Record 8-bit ulaw samples.
- #define SIP_SAMPLE_11KHZ 0x00

  Record samples at 11.025kHz.
- #define SIP_SAMPLE_8KHZ 0x01

  Record samples at 8kHz.

Typedefs

- typedef void(∗ sip_sample_cb) (maple_device_t ∗dev, uint8 ∗samples, size_t len)

  Type for a microphone sample callback.

Functions

- int sip_set_gain (maple_device_t ∗dev, unsigned int g)

  Set the microphone's gain value.
- int sip_set_sample_type (maple_device_t ∗dev, unsigned int type)

  Set the sample type to be recorded by the microphone.
- int sip_set_frequency (maple_device_t ∗dev, unsigned int freq)

  Set the sample frequency to be recorded by the microphone.
- int sip_start_sampling (maple_device_t ∗dev, sip_sample_cb cb, int block)

  Start sampling on a microphone.
- int sip_stop_sampling (maple_device_t ∗dev, int block)

  Stop sampling on a microphone.

9.206.1 Detailed Description

Definitions for using the Sound Input Peripheral.

This file contains the definitions needed to access the Maple microphone type device (the Seaman mic). Many thanks go out to ZeZu who pointed me toward what some of the commands actually do in the original version of this driver.

As a note, the device itself is actually referred to by the system as the Sound Input Peripheral, so hence why this driver is named as it is.

Author

Lawrence Sebald

9.206.2 Macro Definition Documentation

SIP_DEFAULT_GAIN

```
#define SIP_DEFAULT_GAIN 0x0F
```

Default microphone gain.

SIP_MAX_GAIN

```
#define SIP_MAX_GAIN 0x1F
```

Maximum microphone gain.

SIP_MIN_GAIN

```
#define SIP_MIN_GAIN 0x00
```

Minimum microphone gain.

SIP_SAMPLE_11KHZ

```
#define SIP_SAMPLE_11KHZ 0x00
```

Record samples at 11.025kHz.

SIP_SAMPLE_16BIT_SIGNED

```
#define SIP_SAMPLE_16BIT_SIGNED 0x00
```

Record 16-bit signed integer samples.

SIP_SAMPLE_8BIT_ULAW

```
#define SIP_SAMPLE_8BIT_ULAW 0x01
```

Record 8-bit ulaw samples.

SIP_SAMPLE_8KHZ

```
#define SIP_SAMPLE_8KHZ 0x01
```

Record samples at 8kHz.

SIP_SUBCOMMAND_BASIC_CTRL

```
#define SIP_SUBCOMMAND_BASIC_CTRL 0x02
```

Start and stop sampling.

This subcommand is used with the MAPLE_COMMAND_MICCONTROL command to start and stop sampling on the microphone.

SIP_SUBCOMMAND_GET_SAMPLES

```
#define SIP_SUBCOMMAND_GET_SAMPLES 0x01
```

Get recorded samples from the microphone device.

This subcommand is used with the MAPLE_COMMAND_MICCONTROL command to fetch samples from the microphone.

9.206.3 Typedef Documentation

sip_sample_cb

```
typedef void(* sip_sample_cb) (maple_device_t *dev, uint8 *samples, size_t len)
```

Type for a microphone sample callback.

This is the signature that is required for a function to accept samples from the microphone as it is sampling. This function will be called about once per frame, and in an interrupt context (so it should be pretty quick to execute). Basically, all you should do in one of these is copy the samples out to your own buffer – do not do any processing on the samples in your callback other than to copy them out!

Parameters

dev	The device the samples are coming from.
samples	Pointer to the sample buffer.
len	The number of bytes in the sample buffer.

9.206.4 Function Documentation

sip_set_frequency()

```
int sip_set_frequency (
            maple_device_t * dev,
            unsigned int freq )
```

Set the sample frequency to be recorded by the microphone.

This function sets the sample frequency that the microphone will record. The default value for this is about 11.025kHz samples. You must call this prior to sip_start_sampling() if you wish to change it from the default.

Parameters

dev	The microphone device to set sample type on.
freq	The type of samples requested.

Return values

MAPLE_EOK	On success.
MAPLE_EINVALID	If freq is invalid.
MAPLE_EFAIL	If the microphone is sampling.

See also

SIP_SAMPLE_11KHZ
SIP_SAMPLE_8KHZ

sip_set_gain()

```
int sip_set_gain (
            maple_device_t * dev,
            unsigned int g )
```

Set the microphone's gain value.

This function sets the gain value of the specified microphone device to the value given. This should only be called prior to sampling so as to keep the amplification constant throughout the sampling process, but can be changed on the fly if you really want to.

Parameters

dev	The microphone device to set gain on.
g	The value to set as the gain.

Return values

MAPLE_EOK	On success.
MAPLE_EINVALID	If g is out of range.

See also

SIP_MIN_GAIN

SIP_DEFAULT_GAIN

SIP_MAX_GAIN

sip_set_sample_type()

```
int sip_set_sample_type (
            maple_device_t * dev,
            unsigned int type )
```

Set the sample type to be recorded by the microphone.

This function sets the sample type that the microphone will return. The default value for this is 16-bit signed integer samples. You must call this prior to sip_start_sampling() if you wish to change it from the default.

Parameters

dev	The microphone device to set sample type on.
type	The type of samples requested.

Return values

MAPLE_EOK	On success.
MAPLE_EINVALID	If type is invalid.
MAPLE_EFAIL	If the microphone is sampling.

See also

SIP_SAMPLE_16BIT_SIGNED

SIP_SAMPLE_8BIT_ULAW

sip_start_sampling()

```
int sip_start_sampling (
            maple_device_t * dev,
            sip_sample_cb cb,
            int block )
```

Start sampling on a microphone.

This function informs a microphone it should start recording samples.

Parameters

dev	The device to start sampling on.
cb	A callback to call when samples are ready.
block	Set to 1 to wait for the SIP to start sampling. Otherwise check the is_sampling member of the status for dev to know when it has started.

Return values

MAPLE_EOK	On success.
MAPLE_EAGAIN	If the command couldn't be sent, try again later.
MAPLE_EFAIL	If the microphone is already sampling or the callback function is NULL.
MAPLE_ETIMEOUT	If the command timed out while blocking.

sip_stop_sampling()

```
int sip_stop_sampling (
            maple_device_t * dev,
            int block )
```

Stop sampling on a microphone.

This function informs a microphone it should stop recording samples.

Parameters

dev	The device to stop sampling on.
block	Set to 1 to wait for the SIP to stop sampling. Otherwise check the is_sampling member of the status for dev to know when it has finished.

Return values

MAPLE_EOK	On success.
MAPLE_EAGAIN	If the command couldn't be sent, try again later.
MAPLE_EFAIL	If the microphone is not sampling.
MAPLE_ETIMEOUT	If the command timed out while blocking.

9.207 sip.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/maple/sip.h
00004    Copyright (C) 2005, 2008, 2010, 2013 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   dc/maple/sip.h
00009     \brief  Definitions for using the Sound Input Peripheral.
00010
00011     This file contains the definitions needed to access the Maple microphone
00012     type device (the Seaman mic). Many thanks go out to ZeZu who pointed me
00013     toward what some of the commands actually do in the original version of this
00014     driver.
00015
00016     As a note, the device itself is actually referred to by the system as the
00017     Sound Input Peripheral, so hence why this driver is named as it is.
00018
00019     \author Lawrence Sebald
00020 */
00021
00022 #ifndef __DC_MAPLE_SIP_H
00023 #define __DC_MAPLE_SIP_H
00024
00025 #include <sys/cdefs.h>
00026 __BEGIN_DECLS
00027
00028 #include <sys/types.h>
00029 #include <dc/maple.h>
00030
00031 /** \brief  Type for a microphone sample callback.
00032
00033     This is the signature that is required for a function to accept samples
00034     from the microphone as it is sampling. This function will be called about
00035     once per frame, and in an interrupt context (so it should be pretty quick
00036     to execute). Basically, all you should do in one of these is copy the
00037     samples out to your own buffer -- do not do any processing on the samples
00038     in your callback other than to copy them out!
00039
00040     \param  dev             The device the samples are coming from.
00041     \param  samples         Pointer to the sample buffer.
00042     \param  len             The number of bytes in the sample buffer.
00043
00044     \headerfile dc/maple/sip.h
00045 */
00046 typedef void (*sip_sample_cb)(maple_device_t *dev, uint8 *samples, size_t len);
00047
00048 /** \brief  SIP status structure.
00049
00050     This structure contains information about the status of the microphone
00051     device and can be fetched with maple_dev_status(). You should not modify
00052     any of the values in here, it is all "read-only" to your programs. Modifying
00053     any of this, especially while the microphone is sampling could really screw
00054     things up.
00055
00056     \headerfile dc/maple/sip.h
00057 */
00058 typedef struct  sip_state {
00059     /** \brief  The gain value for the microphone amp. */
00060     int             amp_gain;
00061
00062     /** \brief  The type of samples that are being recorded. */
00063     int             sample_type;
00064
00065     /** \brief  What frequency are we sampling at? */
00066     int             frequency;
00067
00068     /** \brief  Is the mic currently sampling? */
00069     int             is_sampling;
00070
00071     /** \brief  Sampling callback. */
00072     sip_sample_cb   callback;
00073 } sip_state_t;
00074
00075 /** \brief  Get recorded samples from the microphone device.
00076
```

```
00077     This subcommand is used with the MAPLE_COMMAND_MICCONTROL command to fetch
00078     samples from the microphone.
00079 */
00080 #define SIP_SUBCOMMAND_GET_SAMPLES 0x01
00081
00082 /** \brief  Start and stop sampling.
00083
00084     This subcommand is used with the MAPLE_COMMAND_MICCONTROL command to start
00085     and stop sampling on the microphone.
00086 */
00087 #define SIP_SUBCOMMAND_BASIC_CTRL  0x02
00088
00089 /** \brief  Minimum microphone gain. */
00090 #define SIP_MIN_GAIN     0x00
00091
00092 /** \brief  Default microphone gain. */
00093 #define SIP_DEFAULT_GAIN 0x0F
00094
00095 /** \brief  Maximum microphone gain. */
00096 #define SIP_MAX_GAIN     0x1F
00097
00098 /** \brief  Set the microphone's gain value.
00099
00100     This function sets the gain value of the specified microphone device to
00101     the value given. This should only be called prior to sampling so as to keep
00102     the amplification constant throughout the sampling process, but can be
00103     changed on the fly if you really want to.
00104
00105     \param  dev             The microphone device to set gain on.
00106     \param  g               The value to set as the gain.
00107     \retval MAPLE_EOK       On success.
00108     \retval MAPLE_EINVALID  If g is out of range.
00109     \see    SIP_MIN_GAIN
00110     \see    SIP_DEFAULT_GAIN
00111     \see    SIP_MAX_GAIN
00112 */
00113 int sip_set_gain(maple_device_t *dev, unsigned int g);
00114
00115 /* Sample types. These two values are the only defined types of samples that
00116    the SIP can output. 16-bit signed is your standard 16-bit signed samples,
00117    where 8-bit ulaw is obvously encoded as ulaw. */
00118
00119 /** \brief  Record 16-bit signed integer samples. */
00120 #define SIP_SAMPLE_16BIT_SIGNED 0x00
00121
00122 /** \brief  Record 8-bit ulaw samples. */
00123 #define SIP_SAMPLE_8BIT_ULAW    0x01
00124
00125 /** \brief  Set the sample type to be recorded by the microphone.
00126
00127     This function sets the sample type that the microphone will return. The
00128     default value for this is 16-bit signed integer samples. You must call this
00129     prior to sip_start_sampling() if you wish to change it from the default.
00130
00131     \param  dev             The microphone device to set sample type on.
00132     \param  type            The type of samples requested.
00133     \retval MAPLE_EOK       On success.
00134     \retval MAPLE_EINVALID  If type is invalid.
00135     \retval MAPLE_EFAIL     If the microphone is sampling.
00136     \see    SIP_SAMPLE_16BIT_SIGNED
00137     \see    SIP_SAMPLE_8BIT_ULAW
00138 */
00139 int sip_set_sample_type(maple_device_t *dev, unsigned int type);
00140
00141 /* Sampling frequencies. The SIP supports sampling at either 8kHz or 11.025 kHz.
00142    One of these values should be passed to the sip_set_frequency function. */
00143 /** \brief  Record samples at 11.025kHz. */
00144 #define SIP_SAMPLE_11KHZ 0x00
00145
00146 /** \brief  Record samples at 8kHz. */
00147 #define SIP_SAMPLE_8KHZ  0x01
00148
00149 /** \brief  Set the sample frequency to be recorded by the microphone.
00150
00151     This function sets the sample frequency that the microphone will record. The
00152     default value for this is about 11.025kHz samples. You must call this prior
00153     to sip_start_sampling() if you wish to change it from the default.
00154
00155     \param  dev             The microphone device to set sample type on.
00156     \param  freq            The type of samples requested.
00157     \retval MAPLE_EOK       On success.
```

```
00158     \retval MAPLE_EINVALID  If freq is invalid.
00159     \retval MAPLE_EFAIL     If the microphone is sampling.
00160     \see    SIP_SAMPLE_11KHZ
00161     \see    SIP_SAMPLE_8KHZ
00162 */
00163 int sip_set_frequency(maple_device_t *dev, unsigned int freq);
00164
00165 /** \brief  Start sampling on a microphone.
00166
00167     This function informs a microphone it should start recording samples.
00168
00169     \param  dev             The device to start sampling on.
00170     \param  cb              A callback to call when samples are ready.
00171     \param  block           Set to 1 to wait for the SIP to start sampling.
00172                             Otherwise check the is_sampling member of the status
00173                             for dev to know when it has started.
00174     \retval MAPLE_EOK       On success.
00175     \retval MAPLE_EAGAIN    If the command couldn't be sent, try again later.
00176     \retval MAPLE_EFAIL     If the microphone is already sampling or the
00177                             callback function is NULL.
00178     \retval MAPLE_ETIMEOUT  If the command timed out while blocking.
00179 */
00180 int sip_start_sampling(maple_device_t *dev, sip_sample_cb cb, int block);
00181
00182 /** \brief  Stop sampling on a microphone.
00183
00184     This function informs a microphone it should stop recording samples.
00185
00186     \param  dev             The device to stop sampling on.
00187     \param  block           Set to 1 to wait for the SIP to stop sampling.
00188                             Otherwise check the is_sampling member of the status
00189                             for dev to know when it has finished.
00190     \retval MAPLE_EOK       On success.
00191     \retval MAPLE_EAGAIN    If the command couldn't be sent, try again later.
00192     \retval MAPLE_EFAIL     If the microphone is not sampling.
00193     \retval MAPLE_ETIMEOUT  If the command timed out while blocking.
00194 */
00195 int sip_stop_sampling(maple_device_t *dev, int block);
00196
00197 /* \cond */
00198 /* Init / Shutdown */
00199 void sip_init(void);
00200 void sip_shutdown(void);
00201 /* \endcond */
00202
00203 __END_DECLS
00204
00205 #endif  /* __DC_MAPLE_SIP_H */
```

9.208 kernel/arch/dreamcast/include/dc/maple/vmu.h File Reference

Definitions for using the VMU device.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <dc/maple.h>
#include <stdint.h>
#include <time.h>
```

Data Structures

- union vmu_state_t

  VMU's "civilized" state data: 0 = RELEASED, 1 = PRESSED.

Macros

- #define VMU_SCREEN_WIDTH 48

  Pixel width of a standard VMU screen.
- #define VMU_SCREEN_HEIGHT 32

  Pixel height of a standard VMU screen.
- #define VMU_DPAD_UP (0<<1)

  Up Dpad button on the VMU.
- #define VMU_DPAD_DOWN (1<<1)

  Down Dpad button on the VMU.
- #define VMU_DPAD_LEFT (2<<1)

  Left Dpad button on the VMU.
- #define VMU_DPAD_RIGHT (3<<1)

  Right Dpad button on the VMU.
- #define VMU_A (4<<1)

  'A' button on the VMU
- #define VMU_B (5<<1)

  'B' button on the VMU
- #define VMU_MODE (6<<1)

  Mode button on the VMU.
- #define VMU_SLEEP (7<<1)

  Sleep button on the VMU.

Typedefs

- typedef uint8_t vmu_cond_t

  VMU's raw condition data: 0 = PRESSED, 1 = RELEASED.

Functions

- int vmu_has_241_blocks (maple_device_t *dev)

  Get the status of a VMUs extra 41 blocks.
- int vmu_toggle_241_blocks (maple_device_t *dev, int enable)

  Enable the extra 41 blocks of a VMU.
- int vmu_use_custom_color (maple_device_t *dev, int enable)

  Enable custom color of a VMU.
- int vmu_set_custom_color (maple_device_t *dev, uint8_t red, uint8_t green, uint8_t blue, uint8_t alpha)

  Set custom color of a VMU.
- int vmu_get_custom_color (maple_device_t *dev, uint8_t *red, uint8_t *green, uint8_t *blue, uint8_t *alpha)

  Get custom color of a VMU.
- int vmu_set_icon_shape (maple_device_t *dev, uint8_t icon_shape)

  Set icon shape of a VMU.
- int vmu_get_icon_shape (maple_device_t *dev, uint8_t *icon_shape)

  Get icon shape of a VMU.
- int vmu_draw_lcd (maple_device_t *dev, const void *bitmap)

  Display a 1bpp bitmap on a VMU screen.

- int vmu_draw_lcd_rotated (maple_device_t *dev, const void *bitmap)
  Display a 1bpp bitmap on a VMU screen.
- int vmu_draw_lcd_xbm (maple_device_t *dev, const char *vmu_icon)
  Display a Xwindows XBM image on a VMU screen.
- void vmu_set_icon (const char *vmu_icon)
  Display a Xwindows XBM on all VMUs.
- int vmu_block_read (maple_device_t *dev, uint16_t blocknum, uint8_t *buffer)
  Read a block from a memory card.
- int vmu_block_write (maple_device_t *dev, uint16_t blocknum, const uint8_t *buffer)
  Write a block to a memory card.

Buzzer

Methods for tone generation.

- int vmu_beep_raw (maple_device_t *dev, uint32_t beep)
  Make a VMU beep (low-level).
- int vmu_beep_waveform (maple_device_t *dev, uint8_t period1, uint8_t duty_cycle1, uint8_t period2, uint8_t duty_cycle2)
  Play VMU Buzzer tone.

Date/Time

Methods for managing date and time.

- int vmu_set_datetime (maple_device_t *dev, time_t unix)
  Set the date and time on the VMU.
- int vmu_get_datetime (maple_device_t *dev, time_t *unix)
  Get the date and time on the VMU.

Input

Methods for polling button states.

- void vmu_set_buttons_enabled (int enable)
  Enable/Disable polling for VMU input.
- int vmu_get_buttons_enabled (void)
  Check whether polling for VMU input has been enabled.

9.208.1 Detailed Description

Definitions for using the VMU device.

This file provides an API around the various Maple function types (LCD, MEMCARD, CLOCK) provided by the Visual Memory Unit. Each API can also be used independently for devices which aren't VMUs, such as using MEMCARD functionality with a standard memory card that lacks a screen or buzzer.

Author

Jordan DeLong

Megan Potter

Donald Haase

Falco Girgis

9.209 vmu.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/maple/vmu.h
00004    Copyright (C) 2000-2002 Jordan DeLong, Megan Potter
00005    Copyright (C) 2008 Donald Haase
00006    Copyright (C) 2023 Falco Girgis, Andy Barajas
00007
00008 */
00009
00010 /** \file    dc/maple/vmu.h
00011     \brief   Definitions for using the VMU device.
00012     \ingroup vmu
00013
00014     This file provides an API around the various Maple function
00015     types (LCD, MEMCARD, CLOCK) provided by the Visual Memory Unit.
00016     Each API can also be used independently for devices which aren't
00017     VMUs, such as using MEMCARD functionality with a standard memory
00018     card that lacks a screen or buzzer.
00019
00020     \author Jordan DeLong
00021     \author Megan Potter
00022     \author Donald Haase
00023     \author Falco Girgis
00024 */
00025
00026 #ifndef __DC_MAPLE_VMU_H
00027 #define __DC_MAPLE_VMU_H
00028
00029 #include <sys/cdefs.h>
00030 __BEGIN_DECLS
00031
00032 #include <arch/types.h>
00033 #include <dc/maple.h>
00034
00035 #include <stdint.h>
00036 #include <time.h>
00037
00038 /** \defgroup vmu Visual Memory Unit
00039     \brief    VMU/VMS Maple Peripheral API
00040
00041     The Sega Dreamcast's Visual Memory Unit (VMU)
00042     is an 8-Bit gaming device which, when plugged into
00043     the controller, communicates with the Dreamcast
00044     as a Maple peripheral.
00045
00046                 Visual Memory Unit
00047                  _________________
00048                 /                 \
00049                 |   @ Dreamcast   |
00050                 |   ___________   |
00051                 |  |           |  |
00052                 |  |           |  |
00053                 |  |           |  |
00054                 |  |           |  |
00055         Sleep   |  |___________|  |   Mode
00056           ------|---------\    /--|-------
00057                 |   |¯|   *   *   |
00058              /--|-|¯   ¯| /¯\ /¯\_|____
00059             /   |  ¯|_|¯  \_/ \_/ |    \
00060           |     |           |      |    B
00061          D-pad  \___________|______/
00062                             |
00063                             A
00064
00065     As a Maple peripheral, the VMU implements the
00066     following functions:
00067     - <b>MEMCARD</b>: Storage device used for saving and
00068                       loading game files.
00069     - <b>LCD</b>:     Secondary LCD display on which additonal
00070                       information may be presented to the player.
00071     - <b>CLOCK</b>:   A device which maintains the current date
00072                       and time, provides at least one buzzer for
00073                       playing tones, and also has buttons used
00074                       for input.
00075
00076     Each Maple function has a corresponding set of C functions
```

```
00077     providing a high-level API around its functionality.
00078
00079 */
00080 /** \defgroup vmu_settings Settings
00081     \brief    Customizable configuration data
00082     \ingroup  vmu
00083
00084     This module provides a high-level abstraction around various
00085     features and settings which can be modified on the VMU. Many
00086     of these operations are provided by the Dreamcast's BIOS when
00087     a VMU has been formatted.
00088 */
00089
00090 /** \brief   Get the status of a VMUs extra 41 blocks
00091     \ingroup vmu_settings
00092
00093     This function checks if the extra 41 blocks of a VMU have been
00094     enabled.
00095
00096     \param  dev             The device to check the status of.
00097
00098     \retval 1               On success: extra blocks are enabled
00099     \retval 0               On success: extra blocks are disabled
00100     \retval -1              On failure
00101 */
00102 int vmu_has_241_blocks(maple_device_t *dev);
00103
00104 /** \brief   Enable the extra 41 blocks of a VMU
00105     \ingroup vmu_settings
00106
00107     This function enables/disables the extra 41 blocks of a specific VMU.
00108
00109     \warning    Enabling the extra blocks of a VMU may render it unusable
00110                 for a very few commercial games.
00111
00112     \param  dev             The device to enable/disable 41 blocks.
00113     \param  enable          Values other than 0 enables. Equal to 0 disables.
00114
00115     \retval 0               On success
00116     \retval -1              On failure
00117 */
00118 int vmu_toggle_241_blocks(maple_device_t *dev, int enable);
00119
00120 /** \brief   Enable custom color of a VMU
00121     \ingroup vmu_settings
00122
00123     This function enables/disables the custom color of a specific VMU.
00124     This color is only displayed in the Dreamcast's file manager.
00125
00126     \param  dev             The device to enable/disable custom color.
00127     \param  enable          Values other than 0 enables. Equal to 0 disables.
00128
00129     \retval 0               On success
00130     \retval -1              On failure
00131
00132     \sa vmu_set_custom_color
00133 */
00134 int vmu_use_custom_color(maple_device_t *dev, int enable);
00135
00136 /** \brief   Set custom color of a VMU
00137     \ingroup vmu_settings
00138
00139     This function sets the custom color of a specific VMU. This color is only
00140     displayed in the Dreamcast's file manager. This function also enables the
00141     use of the custom color. Otherwise it wouldn't show up.
00142
00143     \param  dev             The device to change the color of.
00144     \param  red             The red component. 0-255
00145     \param  green           The green component. 0-255
00146     \param  blue            The blue component. 0-255
00147     \param  alpha           The alpha component. 0-255; 100-255 Recommended
00148
00149     \retval 0               On success
00150     \retval -1              On failure
00151
00152     \sa vmu_get_custom_color, vmu_use_custom_color
00153 */
00154 int vmu_set_custom_color(maple_device_t *dev, uint8_t red, uint8_t green, uint8_t blue, uint8_t alpha);
00155
00156 /** \brief   Get custom color of a VMU
00157     \ingroup vmu_settings
```

```
00158
00159     This function gets the custom color of a specific VMU. This color is only
00160     displayed in the Dreamcast's file manager. This function also returns whether
00161     the custom color is currently enabled.
00162
00163     \param  dev             The device to change the color of.
00164     \param  red             The red component. 0-255
00165     \param  green           The green component. 0-255
00166     \param  blue            The blue component. 0-255
00167     \param  alpha           The alpha component. 0-255; 100-255 Recommended
00168
00169     \retval 1               On success: custom color is enabled
00170     \retval 0               On success: custom color is disabled
00171     \retval -1              On failure
00172
00173     \sa vmu_set_custom_color, vmu_use_custom_color
00174 */
00175 int vmu_get_custom_color(maple_device_t *dev, uint8_t *red, uint8_t *green, uint8_t *blue, uint8_t
    *alpha);
00176
00177 /** \brief   Set icon shape of a VMU
00178     \ingroup vmu_settings
00179
00180     This function sets the icon shape of a specific VMU. The icon shape is a
00181     VMU icon that is displayed on the LCD screen while navigating the Dreamcast
00182     BIOS menu and is the GUI representation of the VMU in the menu's file manager.
00183     The Dreamcast BIOS provides a set of 124 icons to choose from.
00184
00185     \note
00186     When a custom file named "ICONDATA_VMS" is present on a VMU, it overrides this
00187     icon by providing custom icons for both the DC BIOS menu and the VMU's LCD screen.
00188
00189     \param  dev             The device to change the icon shape of.
00190     \param  icon_shape      One of the values found in \ref vmu_icons.
00191
00192     \retval 0               On success
00193     \retval -1              On failure
00194
00195     \sa vmu_icons, vmu_get_icon_shape
00196 */
00197 int vmu_set_icon_shape(maple_device_t *dev, uint8_t icon_shape);
00198
00199 /** \brief   Get icon shape of a VMU
00200     \ingroup vmu_settings
00201
00202     This function gets the icon shape of a specific VMU. The icon shape is a
00203     VMU icon that is displayed on the LCD screen while navigating the Dreamcast
00204     BIOS menu and is the GUI representation of the VMU in the menu's file manager.
00205     The Dreamcast BIOS provides a set of 124 icons to choose from.
00206
00207     \note
00208     When a custom file named "ICONDATA_VMS" is present on a VMU, it overrides this
00209     icon by providing custom icons for both the DC BIOS menu and the VMU's LCD screen.
00210
00211     \param  dev             The device to change the icon shape of.
00212     \param  icon_shape      One of the values found in \ref vmu_icons.
00213
00214     \retval 0               On success
00215     \retval -1              On failure
00216
00217     \sa vmu_icons, vmu_set_icon_shape
00218 */
00219 int vmu_get_icon_shape(maple_device_t *dev, uint8_t *icon_shape);
00220
00221 /** \defgroup maple_lcd LCD Function
00222     \brief API for features of the LCD Maple Function
00223     \ingroup  vmu
00224
00225     The LCD Maple function is for exposing a secondary LCD screen
00226     that gets attached to a controller, which can be used to display
00227     additional game information, or information you only want visible
00228     to a single player.
00229 */
00230
00231 /**
00232     \brief   Pixel width of a standard VMU screen
00233     \ingroup maple_lcd
00234 */
00235 #define VMU_SCREEN_WIDTH    48
00236
00237 /**
```

```
00238     \brief Pixel height of a standard VMU screen
00239     \ingroup maple_lcd
00240 */
00241 #define VMU_SCREEN_HEIGHT   32
00242
00243 /** \brief   Display a 1bpp bitmap on a VMU screen.
00244     \ingroup maple_lcd
00245
00246     This function sends a raw bitmap to a VMU to display on its screen. This
00247     bitmap is 1bpp, and is 48x32 in size.
00248
00249     \param  dev             The device to draw to.
00250     \param  bitmap          The bitmap to show.
00251
00252     \retval MAPLE_EOK       On success.
00253     \retval MAPLE_EAGAIN    If the command couldn't be sent. Try again later.
00254     \retval MAPLE_ETIMEOUT  If the command timed out while blocking.
00255
00256     \sa vmu_draw_lcd_rotated, vmu_draw_lcd_xbm, vmu_set_icon
00257 */
00258 int vmu_draw_lcd(maple_device_t *dev, const void *bitmap);
00259
00260 /** \brief   Display a 1bpp bitmap on a VMU screen.
00261     \ingroup maple_lcd
00262
00263     This function sends a raw bitmap to a VMU to display on its screen. This
00264     bitmap is 1bpp, and is 48x32 in size. This function is equivalent to
00265     vmu_draw_lcd(), but the image is rotated 180° so that the first byte of the
00266     bitmap corresponds to the top-left corner, instead of the bottom-right one.
00267
00268     \warning    This function is optimized by an assembly routine which operates
00269                 on 32 bits at a time. As such, the given bitmap must be 4-byte
00270                 aligned.
00271
00272     \param  dev             The device to draw to.
00273     \param  bitmap          The bitmap to show.
00274     \retval MAPLE_EOK       On success.
00275     \retval MAPLE_EAGAIN    If the command couldn't be sent. Try again later.
00276     \retval MAPLE_ETIMEOUT  If the command timed out while blocking.
00277
00278     \sa vmu_draw_lcd, vmu_draw_lcd_xbm, vmu_set_icon
00279 */
00280 int vmu_draw_lcd_rotated(maple_device_t *dev, const void *bitmap);
00281
00282 /** \brief   Display a Xwindows XBM image on a VMU screen.
00283     \ingroup maple_lcd
00284
00285     This function takes in a Xwindows XBM, converts it to a raw bitmap, and sends
00286     it to a VMU to display on its screen. This XBM image is 48x32 in size.
00287
00288     \param  dev             The device to draw to.
00289     \param  vmu_icon        The icon to set.
00290
00291     \retval MAPLE_EOK       On success.
00292     \retval MAPLE_EAGAIN    If the command couldn't be sent. Try again later.
00293     \retval MAPLE_ETIMEOUT  If the command timed out while blocking.
00294
00295     \sa vmu_draw_lcd, vmu_set_icon
00296 */
00297 int vmu_draw_lcd_xbm(maple_device_t *dev, const char *vmu_icon);
00298
00299 /** \brief   Display a Xwindows XBM on all VMUs.
00300     \ingroup maple_lcd
00301
00302     This function takes in a Xwindows XBM and displays the image on all VMUs.
00303
00304     \note
00305     This is a convenience function for vmu_draw_lcd() to broadcast across all VMUs.
00306
00307     \todo
00308     Prevent this routine from broadcasting to rear VMUs.
00309
00310     \param  vmu_icon        The icon to set.
00311
00312     \sa vmu_draw_lcd_xbm
00313 */
00314 void vmu_set_icon(const char *vmu_icon);
00315
00316 /** \defgroup maple_memcard Memory Card Function
00317     \brief    API for features of the Memory Card Maple Function
00318     \ingroup  vmu
```

```
00319
00320     The Memory Card Maple function is for exposing a low-level,
00321     block-based API that allows you to read from and write to
00322     random blocks within the memory card's filesytem.
00323
00324     \note
00325     A standard memory card has a block size of 512 bytes; however,
00326     the block size is a configurable parameter in the "root" block,
00327     which can be queried to cover supporting homebrew memory
00328     cards with larger block sizes.
00329
00330     \warning
00331     You should never use these functions directly, unless you
00332     <i>really</i> know what you're doing, as you can easily corrupt
00333     the filesystem by writing incorrect data. Instead, you should
00334     favor the high-level filesystem API found in vmufs.h, or just
00335     use the native C standard filesystem API within the virtual
00336     `/vmu/' root directory to operate on VMU data.
00337 */
00338
00339 /** \brief   Read a block from a memory card.
00340     \ingroup maple_memcard
00341
00342     This function performs a low-level raw block read from a memory card.
00343
00344     \param  dev             The device to read from.
00345     \param  blocknum        The block number to read.
00346     \param  buffer          The buffer to read into (512 bytes).
00347
00348     \retval MAPLE_EOK       On success.
00349     \retval MAPLE_ETIMEOUT  If the command timed out while blocking.
00350     \retval MAPLE_EFAIL     On errors other than timeout.
00351
00352     \sa vmu_block_write
00353 */
00354 int vmu_block_read(maple_device_t *dev, uint16_t blocknum, uint8_t *buffer);
00355
00356 /** \brief   Write a block to a memory card.
00357     \ingroup maple_memcard
00358
00359     This function performs a low-level raw block write to a memory card.
00360
00361     \param  dev             The device to write to.
00362     \param  blocknum        The block number to write.
00363     \param  buffer          The buffer to write from (512 bytes).
00364
00365     \retval MAPLE_EOK       On success.
00366     \retval MAPLE_ETIMEOUT  If the command timed out while blocking.
00367     \retval MAPLE_EFAIL     On errors other than timeout.
00368
00369     \sa vmu_block_read
00370 */
00371 int vmu_block_write(maple_device_t *dev, uint16_t blocknum, const uint8_t *buffer);
00372
00373 /** \defgroup maple_clock Clock Function
00374     \brief    API for features of the Clock Maple Function
00375     \ingroup  vmu
00376
00377     The Clock Maple function provides a high-level API for the
00378     following functionality:
00379         - buzzer tone generation
00380         - date/time management
00381         - input/button status
00382 */
00383
00384 /** \name Buzzer
00385     \brief Methods for tone generation.
00386     @{
00387 */
00388
00389 /** \brief   Make a VMU beep (low-level).
00390     \ingroup maple_clock
00391
00392     This function sends a raw beep to a VMU, causing the speaker to emit a tone
00393     noise.
00394
00395     \note
00396     See http://dcemulation.org/phpBB/viewtopic.php?f=29&t=97048 for the
00397     original information about beeping.
00398
00399     \warning
```

```
00400     This function is submitting raw, encoded values to the VMU. For a more
00401     user-friendly API built around generating simple tones, see vmu_beep_waveform().
00402
00403     \param  dev             The device to attempt to beep.
00404     \param  beep            The tone to generate. Byte values are as follows:
00405                                 1. period of square wave 1
00406                                 2. duty cycle of square wave 1
00407                                 3. period of square wave 2 (ignored by
00408                                    standard mono VMUs)
00409                                 4. duty cycle of square wave 2 (ignored by
00410                                    standard mono VMUs)
00411
00412     \retval MAPLE_EOK       On success.
00413     \retval MAPLE_EAGAIN    If the command couldn't be sent. Try again later.
00414     \retval MAPLE_ETIMEOUT  If the command timed out while blocking.
00415
00416     \sa vmu_beep_waveform
00417 */
00418 int vmu_beep_raw(maple_device_t *dev, uint32_t beep);
00419
00420 /** \brief   Play VMU Buzzer tone.
00421     \ingroup maple_clock
00422
00423     Sends two different square waves to generate tone(s) on the VMU. Each
00424     waveform is configured as shown by the following diagram. On a standard
00425     VMU, there is only one piezoelectric buzzer, so waveform 2 is ignored;
00426     however, the parameters do support dual-channel stereo in case such a
00427     VMU ever does come along.
00428
00429                               Period
00430                         +---------------------+
00431                         |                     |
00432         HIGH ___________                 ___________
00433                         |           |     |           |           |
00434                         |           |     |           |           |
00435                         |___________|     |___________|
00436         LOW
00437                                     |           |
00438                                     +-----------+
00439                                      Duty Cycle
00440
00441                               WAVEFORM
00442
00443     To stop an active tone, one can simply generate a flat wave, such as by
00444     submitting both values as 0s.
00445
00446     \warning
00447     Any submitted waveform which has a duty cycle of greater than or equal to
00448     its period will result in an invalid waveform being generated and is
00449     going to mute or end the tone.
00450
00451     \note
00452     Note that there are no units given for the waveform, so any 3rd party VMU
00453     is free to use any base clock rate, potentially resulting in different
00454     frequencies (or tones) being generated for the same parameters on different
00455     devices.
00456
00457     \note
00458     On the VMU-side, this tone is generated using the VMU's Timer1 peripheral
00459     as a pulse generator, which is then fed into its piezoelectric buzzer. The
00460     calculated range of the standard VMU, given its 6MHz CF clock running with a
00461     divisor of 6 is driving the Timer1 counter, is approximately 3.9KHz-500Khz;
00462     however, due to physical characteristics of the buzzer, not every frequency
00463     can be produced at a decent volume, so it's recommended that you test your
00464     values, using the KOS example found at `/examples/dreamcast/vmu/beep`.
00465
00466     \param  dev                 The VMU device to play the tone on
00467     \param  period1             The period or total interval of the first waveform
00468     \param  duty_cycle1         The duty cycle or active interval of the first waveform
00469     \param  period2             The period or total interval of the second waveform
00470                                 (ignored by standard first-party VMUs).
00471     \param  duty_cycle2         The duty cycle or active interval of the second waveform
00472                                 (ignored by standard first-party VMUs).
00473
00474     \retval MAPLE_EOK           On success.
00475     \retval MAPLE_EAGAIN        If the command couldn't be sent. Try again later.
00476     \retval MAPLE_ETIMEOUT      If the command timed out while blocking.
00477 */
00478 int vmu_beep_waveform(maple_device_t *dev, uint8_t period1, uint8_t duty_cycle1, uint8_t period2, uint8_t
      duty_cycle2);
00479
```

```
00480 /** @} */
00481
00482 /** \name Date/Time
00483     \brief Methods for managing date and time.
00484     @{
00485 */
00486
00487 /** \brief  Set the date and time on the VMU.
00488     \ingroup maple_clock
00489
00490     This function sets the VMU's date and time values to
00491     the given standard C Unix timestamp.
00492
00493     \param  dev             The device to write to.
00494     \param  unix            Seconds since Unix epoch
00495
00496     \retval MAPLE_EOK       On success.
00497     \retval MAPLE_ETIMEOUT  If the command timed out while blocking.
00498     \retval MAPLE_EFAIL     On errors other than timeout.
00499
00500     \sa vmu_get_datetime
00501 */
00502 int vmu_set_datetime(maple_device_t *dev, time_t unix);
00503
00504 /** \brief   Get the date and time on the VMU.
00505     \ingroup maple_clock
00506
00507     This function gets the VMU's date and time values
00508     as a single standard C Unix timestamp.
00509
00510     \note
00511     This is the VMU equivalent of calling `time(unix)`.
00512
00513     \param  dev             The device to write to.
00514     \param  unix            Seconds since Unix epoch (set to -1 upon failure)
00515
00516     \retval MAPLE_EOK       On success.
00517     \retval MAPLE_ETIMEOUT  If the command timed out while blocking.
00518     \retval MAPLE_EFAIL     On errors other than timeout.
00519
00520     \sa vmu_set_datetime
00521 */
00522 int vmu_get_datetime(maple_device_t *dev, time_t *unix);
00523
00524 /** @} */
00525
00526 /** \defgroup vmu_buttons VMU Buttons
00527     \brief    VMU button masks
00528     \ingroup  maple_clock
00529
00530     VMU's button state/cond masks, same as capability masks
00531
00532     \note
00533     The MODE and SLEEP button states are not pollable on
00534     a standard VMU.
00535
00536     @{
00537 */
00538
00539 #define VMU_DPAD_UP     (0<<1)   /**< \brief Up Dpad button on the VMU */
00540 #define VMU_DPAD_DOWN   (1<<1)   /**< \brief Down Dpad button on the VMU */
00541 #define VMU_DPAD_LEFT   (2<<1)   /**< \brief Left Dpad button on the VMU */
00542 #define VMU_DPAD_RIGHT  (3<<1)   /**< \brief Right Dpad button on the VMU */
00543 #define VMU_A           (4<<1)   /**< \brief 'A' button on the VMU */
00544 #define VMU_B           (5<<1)   /**< \brief 'B' button on the VMU */
00545 #define VMU_MODE        (6<<1)   /**< \brief Mode button on the VMU */
00546 #define VMU_SLEEP       (7<<1)   /**< \brief Sleep button on the VMU */
00547
00548 /** \brief VMU's raw condition data: 0 = PRESSED, 1 = RELEASED */
00549 typedef uint8_t vmu_cond_t;
00550
00551 /** \brief  VMU's "civilized" state data: 0 = RELEASED, 1 = PRESSED
00552
00553     \note
00554     The Dpad buttons are automatically reoriented for you depending on
00555     which direction the VMU is facing in a particular type of controller.
00556  */
00557 typedef union vmu_state {
00558     uint8_t buttons;            /**< \brief Combined button state mask */
00559     struct {
00560         uint8_t dpad_up:    1;  /**< \brief Dpad Up button state */
```

```
00561         uint8_t dpad_down:  1;  /**< \brief Dpad Down button state */
00562         uint8_t dpad_left:  1;  /**< \brief Dpad Left button state */
00563         uint8_t dpad_right: 1;  /**< \brief Dpad Right button state */
00564         uint8_t a:          1;  /**< \brief 'A' button state */
00565         uint8_t b:          1;  /**< \brief 'B' button state */
00566         uint8_t mode:       1;  /**< \brief Mode button state */
00567         uint8_t sleep:      1;  /**< \brief Sleep button state */
00568     };
00569 } vmu_state_t;
00570
00571 /** @} */
00572
00573 /** \name Input
00574     \brief Methods for polling button states.
00575     @{
00576 */
00577
00578 /** \brief   Enable/Disable polling for VMU input
00579     \ingroup maple_clock
00580
00581     This function is used to either enable or disable polling the
00582     VMU buttons' states for input each frame.
00583
00584     \note
00585     These buttons are not usually accessible to the player; however,
00586     several devices, such as the ASCII pad, the arcade pad, and
00587     the Retro Fighters controller leave the VMU partially exposed,
00588     so that these buttons remain accessible, allowing them to be used
00589     as extended controller inputs.
00590
00591     \note
00592     Polling for VMU input is disabled by default to reduce unecessary
00593     Maple BUS traffic.
00594
00595     \sa vmu_get_buttons_enabled
00596 */
00597 void vmu_set_buttons_enabled(int enable);
00598
00599 /** \brief   Check whether polling for VMU input has been enabled
00600     \ingroup maple_clock
00601
00602     This function is used to check whether per-frame polling of
00603     the VMU's button states has been enabled in the driver.
00604
00605     \note
00606     Polling for VMU input is disabled by default to reduce unecessary
00607     Maple BUS traffic.
00608
00609     \sa vmu_set_buttons_enabled
00610 */
00611 int vmu_get_buttons_enabled(void);
00612
00613 /** @} */
00614
00615 /** \cond */
00616 /* Init / Shutdown -- Managed internally by KOS */
00617 void vmu_init(void);
00618 void vmu_shutdown(void);
00619 /** \endcond */
00620
00621 __END_DECLS
00622
00623 #endif  /* __DC_MAPLE_VMU_H */
00624
```

9.210 kernel/arch/dreamcast/include/dc/math.h File Reference

Prototypes for optimized math functions written in ASM.

```
#include <sys/cdefs.h>
```

Functions

- unsigned int bit_reverse (unsigned int value)

  Returns the bit-reverse value of the argument (where MSB becomes LSB and vice-versa).

9.210.1 Detailed Description

Prototypes for optimized math functions written in ASM.

Author

Paul Cercueil

9.210.2 Function Documentation

bit_reverse()

```
unsigned int bit_reverse (
            unsigned int value )
```

Returns the bit-reverse value of the argument (where MSB becomes LSB and vice-versa).

Returns

the bit-reverse value of the argument.

9.211 math.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/math.h
00004    Copyright (C) 2023 Paul Cercueil
00005
00006 */
00007
00008 #ifndef __DC_MATH_H
00009 #define __DC_MATH_H
00010
00011 #include <sys/cdefs.h>
00012 __BEGIN_DECLS
00013
00014 /**
00015     \file   dc/math.h
00016     \brief  Prototypes for optimized math functions written in ASM
00017     \author Paul Cercueil
00018 */
00019
00020 /**
00021     \brief  Returns the bit-reverse value of the argument (where MSB
00022             becomes LSB and vice-versa).
00023     \return the bit-reverse value of the argument.
00024 */
00025 unsigned int bit_reverse(unsigned int value);
00026
00027 #endif /* __DC_MATH_H */
```

9.212 kernel/arch/dreamcast/include/dc/matrix.h File Reference

Basic matrix operations.

```
#include <sys/cdefs.h>
#include <dc/vector.h>
```

Macros

- #define mat_trans_single(x, y, z)

  Macro to transform a single vertex by the internal matrix.
- #define mat_trans_single4(x, y, z, w)

  Macro to transform a single vertex by the internal matrix.
- #define mat_trans_single3(x, y, z)

  Macro to transform a single vertex by the internal matrix.
- #define mat_trans_nodiv(x, y, z, w)

  Macro to transform a single vertex by the internal matrix with no perspective division.
- #define mat_trans_single3_nodiv(x, y, z)

  Macro to transform a single 3d vertex coordinate by the internal matrix with no perspective division.
- #define mat_trans_single3_nomod(x, y, z, x2, y2, z2)

  Macro to transform a single 3d vertex coordinate by the internal matrix with perspective division.
- #define mat_trans_single3_nodiv_nomod(x, y, z, x2, y2, z2)

  Macro to transform a single 3d vertex coordinate by the internal matrix.
- #define mat_trans_single3_nodivw(x, y, z, w)

  Macro to transform a single 3d vertex coordinate by the internal matrix.
- #define mat_trans_single3_nodiv_div(x, y, z, xd, yd, zd)

  Macro to transform a single 3d vertex coordinate by the internal matrix both with and without perspective division.
- #define mat_trans_normal3(x, y, z)

  Macro to transform a single vertex normal by the internal matrix.
- #define mat_trans_normal3_nomod(x, y, z, x2, y2, z2)

  Macro to transform a single vertex normal by the internal matrix.

Functions

- void mat_store (matrix_t *out)

  Copy the internal matrix to a memory one.
- void mat_load (matrix_t *out)

  Copy a memory matrix into the internal one.
- void mat_identity (void)

  Clear the internal matrix to identity.
- void mat_apply (matrix_t *src)

  Apply a matrix.
- void mat_transform (vector_t *invecs, vector_t *outvecs, int veccnt, int vecskip)

  Transform vectors by the internal matrix.
- void mat_transform_sq (void *input, void *output, int veccnt)

  Transform vectors by the internal matrix into the store queues.

9.212.1 Detailed Description

Basic matrix operations.

This file contains various basic matrix math functionality for using the SH4's matrix transformation unit. Higher level functionality, like the 3D functionality is built off of these operations.

Author

Megan Potter

Josh "PH3NOM" Pearson

See also

dc/matrix3d.h

9.212.2 Macro Definition Documentation

mat_trans_nodiv

```
#define mat_trans_nodiv(
              x,
              y,
              z,
              w )
```

Value:

```
        { \
        register float __x __asm__("fr0") = (x); \
        register float __y __asm__("fr1") = (y); \
        register float __z __asm__("fr2") = (z); \
        register float __w __asm__("fr3") = (w); \
        __asm__ __volatile__( \
                              "ftrv   xmtrx,fv0\n" \
                              : "=f" (__x), "=f" (__y), "=f" (__z), "=f" (__w) \
                              : "0" (__x), "1" (__y), "2" (__z), "3" (__w) ); \
        x = __x; y = __y; z = __z; w = __w; \
    }
```

Macro to transform a single vertex by the internal matrix with no perspective division.

This macro is an inline assembly operation to transform a single vertex. It works most efficiently if the x value is in fr0, y is in fr1, z is in fr2, and w is in fr3 before using the macro. This macro is similar to mat_trans_single(), but this one does not do any perspective division.

Parameters

x	The X coordinate to transform.
y	The Y coordinate to transform.
z	The Z coordinate to transform.
w	The W coordinate to transform.

mat_trans_normal3

```
#define mat_trans_normal3(
              x,
              y,
              z )
```

Value:

```
        { \
        register float __x __asm__("fr8") = (x); \
        register float __y __asm__("fr9") = (y); \
        register float __z __asm__("fr10") = (z); \
        __asm__ __volatile__( \
                              "fldi0 fr11\n" \
                              "ftrv  xmtrx, fv8\n" \
                              : "=f" (__x), "=f" (__y), "=f" (__z) \
                              : "0" (__x), "1" (__y), "2" (__z) \
                              : "fr11" ); \
        x = __x; y = __y; z = __z; \
    }
```

Macro to transform a single vertex normal by the internal matrix.

This macro is an inline assembly operation to transform a 3 float vertex normal. It works most efficiently if the x value is in fr8, y is in fr9, and z is in fr10 before using the macro. This macro is similar to mat_trans_nodiv(), but this one sets the W component to 0 in order to transform a vertex normal, rather than 1 for a vertex position.

Parameters

x	The X normal to transform.
y	The Y normal to transform.
z	The Z normal to transform.

mat_trans_normal3_nomod

```
#define mat_trans_normal3_nomod(
              x,
              y,
              z,
              x2,
              y2,
              z2 )
```

Value:

```
        { \
        register float __x __asm__("fr8") = (x); \
        register float __y __asm__("fr9") = (y); \
        register float __z __asm__("fr10") = (z); \
        __asm__ __volatile__( \
                              "fldi0 fr11\n" \
                              "ftrv  xmtrx, fv8\n" \
                              : "=f" (__x), "=f" (__y), "=f" (__z) \
                              : "0" (__x), "1" (__y), "2" (__z) \
                              : "fr11" ); \
        x2 = __x; y2 = __y; z2 = __z; \
    }
```

Macro to transform a single vertex normal by the internal matrix.

This macro is an inline assembly operation to transform a 3 float vertex normal. It works most efficiently if the x value is in fr8, y is in fr9, and z is in fr10 before using the macro. This macro is similar to mat_trans_normal3(), but this one does not modify the input operands, instead storing the transformed vector to the output operands.

Parameters

x	The X normal to input transform.
y	The Y normal to input transform.
z	The Z normal to input transform.
x2	The X normal to output transform.
y2	The Y normal to output transform.
z2	The Z normal to output transform.

mat_trans_single

```
#define mat_trans_single(
              x,
              y,
              z )
```

Value:

```
        { \
        register float __x __asm__("fr0") = (x); \
        register float __y __asm__("fr1") = (y); \
        register float __z __asm__("fr2") = (z); \
        __asm__ __volatile__( \
                              "fldi1    fr3\n" \
                              "ftrv     xmtrx,fv0\n" \
                              "fldi1    fr2\n" \
                              "fdiv     fr3,fr2\n" \
                              "fmul     fr2,fr0\n" \
                              "fmul     fr2,fr1\n" \
                              : "=f" (__x), "=f" (__y), "=f" (__z) \
                              : "0" (__x), "1" (__y), "2" (__z) \
                              : "fr3" ); \
        x = __x; y = __y; z = __z; \
    }
```

Macro to transform a single vertex by the internal matrix.

This macro is an inline assembly operation to transform a single vertex. It works most efficiently if the x value is in fr0, y is in fr1, and z is in fr2 before using the macro.

Parameters

x	The X coordinate to transform.
y	The Y coordinate to transform.
z	The Z coordinate to transform.

mat_trans_single3

```
#define mat_trans_single3(
              x,
```

```
                y,
                z )
```

Value:

```
        { \
        register float __x __asm__("fr0") = (x); \
        register float __y __asm__("fr1") = (y); \
        register float __z __asm__("fr2") = (z); \
        __asm__ __volatile__( \
                              "fldi1    fr3\n" \
                              "ftrv     xmtrx,fv0\n" \
                              "fdiv     fr3,fr0\n" \
                              "fdiv     fr3,fr1\n" \
                              "fdiv     fr3,fr2\n" \
                              : "=f" (__x), "=f" (__y), "=f" (__z) \
                              : "0" (__x), "1" (__y), "2" (__z) \
                              : "fr3" ); \
        x = __x; y = __y; z = __z; \
    }
```

Macro to transform a single vertex by the internal matrix.

This macro is an inline assembly operation to transform a single vertex. It works most efficiently if the x value is in fr0, y is in fr1, and z is in fr2 before using the macro. This macro is similar to mat_trans_single(), but this one leaves z/w instead of 1/w for the z component.

Parameters

x	The X coordinate to transform.
y	The Y coordinate to transform.
z	The Z coordinate to transform.

mat_trans_single3_nodiv

```
#define mat_trans_single3_nodiv(
                x,
                y,
                z )
```

Value:

```
        { \
        register float __x __asm__("fr12") = (x); \
        register float __y __asm__("fr13") = (y); \
        register float __z __asm__("fr14") = (z); \
        __asm__ __volatile__( \
                              "fldi1 fr15\n" \
                              "ftrv  xmtrx, fv12\n" \
                              : "=f" (__x), "=f" (__y), "=f" (__z) \
                              : "0" (__x), "1" (__y), "2" (__z) \
                              : "fr15" ); \
        x = __x; y = __y; z = __z; \
    }
```

Macro to transform a single 3d vertex coordinate by the internal matrix with no perspective division.

This macro is an inline assembly operation to transform a 3 float vertex coordinate. It works most efficiently if the x value is in fr12, y is in fr13, and z is in fr14 before using the macro. This macro is similar to mat_trans_nodiv(), but this one sets the W component to 1 for use with a 3d vector.

Parameters

x	The X coordinate to transform.
y	The Y coordinate to transform.
z	The Z coordinate to transform.

mat_trans_single3_nodiv_div

```
#define mat_trans_single3_nodiv_div(
              x,
              y,
              z,
              xd,
              yd,
              zd )
```

Value:

```
        { \
        register float __x __asm__("fr0") = (x); \
        register float __y __asm__("fr1") = (y); \
        register float __z __asm__("fr2") = (z); \
        register float __xd __asm__("fr4"); \
        register float __yd __asm__("fr5"); \
        register float __zd __asm__("fr6"); \
        __asm__ __volatile__( \
                              "fldi1 fr3\n" \
                              "ftrv  xmtrx, fv0\n" \
                              "fmov  fr0, fr4\n" \
                              "fmov  fr1, fr5\n" \
                              "fmov  fr3, fr7\n" \
                              "fldi1 fr6\n" \
                              "fdiv    fr7, fr6\n" \
                              "fmul    fr6, fr4\n" \
                              "fmul    fr6, fr5\n" \
                              : "=f" (__x), "=f" (__y), "=f" (__z), \
                                "=f" (__xd), "=f" (__yd), "=f" (__zd) \
                              : "0" (__x), "1" (__y), "2" (__z) \
                              : "fr3" ); \
        x = __x; y = __y; z = __z; xd = __xd; yd = __yd; zd = __zd; \
    }
```

Macro to transform a single 3d vertex coordinate by the internal matrix both with and without perspective division.

This macro is an inline assembly operation to transform a 3 float vertex coordinate. It works most efficiently if the x value is in fr0, y is in fr1, and z is in fr2 before using the macro. This macro is similar to mat_trans_single(), but this one is used for transforming input vertex with and without perspective division.

Parameters

x	The X coordinate to transform without perspective divide.
y	The Y coordinate to transform without perspective divide.
z	The Z coordinate to transform without perspective divide.
xd	The X coordinate to output transform with perspective divide.
yd	The Y coordinate to output transform with perspective divide.
zd	The Z coordinate to output transform with perspective divide.

mat_trans_single3_nodiv_nomod

```
#define mat_trans_single3_nodiv_nomod(
                x,
                y,
                z,
                x2,
                y2,
                z2 )
```

Value:

```
        { \
        register float __x __asm__("fr12") = (x); \
        register float __y __asm__("fr13") = (y); \
        register float __z __asm__("fr14") = (z); \
        __asm__ __volatile__( \
                              "fldi1 fr15\n" \
                              "ftrv  xmtrx, fv12\n" \
                              : "=f" (__x), "=f" (__y), "=f" (__z) \
                              : "0" (__x), "1" (__y), "2" (__z) \
                              : "fr15" ); \
        x2 = __x; y2 = __y; z2 = __z; \
    }
```

Macro to transform a single 3d vertex coordinate by the internal matrix.

This macro is an inline assembly operation to transform a 3 float vertex coordinate. It works most efficiently if the x value is in fr12, y is in fr13, and z is in fr14 before using the macro. This macro is similar to mat_trans_single3_nodiv(), but this one does not modify the input operands, instead storing the transformed vector to the output operands.

Parameters

x	The X coordinate to input transform.
y	The Y coordinate to input transform.
z	The Z coordinate to input transform.
x2	The X coordinate to output transform.
y2	The Y coordinate to output transform.
z2	The Z coordinate to output transform.

mat_trans_single3_nodivw

```
#define mat_trans_single3_nodivw(
                x,
                y,
                z,
                w )
```

Value:

```
        { \
        register float __x __asm__("fr12") = (x); \
        register float __y __asm__("fr13") = (y); \
        register float __z __asm__("fr14") = (z); \
        register float __w __asm__("fr15") = 1.0f; \
        __asm__ __volatile__( \
                              "ftrv  xmtrx, fv12\n" \
                              : "=f" (__x), "=f" (__y), "=f" (__z), "=f" (__w) \
```

```
                        : "0" (__x), "1" (__y), "2" (__z), "3" (__w) ); \
    x = __x; y = __y; z = __z; w = __w; \
}
```

Macro to transform a single 3d vertex coordinate by the internal matrix.

This macro is an inline assembly operation to transform a 3 float vertex coordinate. It works most efficiently if the x value is in fr12, y is in fr13, and z is in fr14 before using the macro. This macro is similar to mat_trans_single3_nodiv(), but this one stores the W component of transform for later perspective divide.

Parameters

x	The X coordinate to transform.
y	The Y coordinate to transform.
z	The Z coordinate to transform.
w	The W coordinate output of transform.

mat_trans_single3_nomod

```
#define mat_trans_single3_nomod(
              x,
              y,
              z,
              x2,
              y2,
              z2 )
```

Value:

```
{ \
    register float __x __asm__("fr12") = (x); \
    register float __y __asm__("fr13") = (y); \
    register float __z __asm__("fr14") = (z); \
    __asm__ __volatile__( \
                        "fldi1 fr15\n" \
                        "ftrv    xmtrx, fv12\n" \
                        "fldi1 fr14\n" \
                        "fdiv    fr15, fr14\n" \
                        "fmul    fr14, fr12\n" \
                        "fmul    fr14, fr13\n" \
                        : "=f" (__x), "=f" (__y), "=f" (__z) \
                        : "0" (__x), "1" (__y), "2" (__z) \
                        : "fr15" ); \
    x2 = __x; y2 = __y; z2 = __z; \
}
```

Macro to transform a single 3d vertex coordinate by the internal matrix with perspective division.

This macro is an inline assembly operation to transform a 3 float vertex coordinate. It works most efficiently if the x value is in fr12, y is in fr13, and z is in fr14 before using the macro. This macro is similar to mat_trans_single(), but this one does not modify the input operands, instead storing the transformed vector to the output operands.

Parameters

x	The X coordinate to input transform.
y	The Y coordinate to input transform.
z	The Z coordinate to input transform.
x2	The X coordinate to output transform.
y2	The Y coordinate to output transform.
z2	The Z coordinate to output transform.

mat_trans_single4

```
#define mat_trans_single4(
                x,
                y,
                z,
                w )
```

Value:

```
        { \
        register float __x __asm__("fr0") = (x); \
        register float __y __asm__("fr1") = (y); \
        register float __z __asm__("fr2") = (z); \
        register float __w __asm__("fr3") = (w); \
        __asm__ __volatile__( \
                              "ftrv     xmtrx,fv0\n" \
                              "fdiv     fr3,fr0\n" \
                              "fdiv     fr3,fr1\n" \
                              "fdiv     fr3,fr2\n" \
                              "fldi1    fr4\n" \
                              "fdiv     fr3,fr4\n" \
                              "fmov     fr4,fr3\n" \
                              : "=f" (__x), "=f" (__y), "=f" (__z), "=f" (__w) \
                              : "0" (__x), "1" (__y), "2" (__z), "3" (__w) \
                              : "fr4" ); \
        x = __x; y = __y; z = __z; w = __w; \
    }
```

Macro to transform a single vertex by the internal matrix.

This macro is an inline assembly operation to transform a single vertex. It works most efficiently if the x value is in fr0, y is in fr1, z is in fr2, and w is in fr3 before using the macro. This macro is similar to mat_trans_single(), but this one allows an input to and preserves the Z/W value.

Parameters

x	The X coordinate to transform.
y	The Y coordinate to transform.
z	The Z coordinate to transform.
w	The W coordinate to transform.

9.212.3 Function Documentation

mat_apply()

```
void mat_apply (
             matrix_t * src )
```

Apply a matrix.

This function multiplies a matrix in memory onto the internal matrix.

Warning

`src` MUST be at least 8-byte aligned!

Note

For best performance, 32-byte alignment of `src` is recommended.

Parameters

src	A poitner to the matrix to multiply.

mat_identity()

```
void mat_identity (
            void  )
```

Clear the internal matrix to identity.

This function clears the internal matrix to a standard identity matrix.

mat_load()

```
void mat_load (
            matrix_t * out )
```

Copy a memory matrix into the internal one.

This function loads the internal matrix with the values of one in memory.

Warning

`out` MUST be at least 8-byte aligned!

Note

For best performance, 32-byte alignment of `out` is recommended.

Parameters

out	A pointer to where to load the matrix from (must be at least 8-byte aligned, should be 32-byte aligned).

mat_store()

```
void mat_store (
            matrix_t * out )
```

Copy the internal matrix to a memory one.

This function stores the current internal matrix to one in memory.

Warning

`out` MUST be at least 8-byte aligned!

Note

For best performance, 32-byte alignment of `out` is recommended.

Parameters

out	A pointer to where to store the matrix (must be at least 8-byte aligned, should be 32-byte aligned).

mat_transform()

```
void mat_transform (
             vector_t * invecs,
             vector_t * outvecs,
             int veccnt,
             int vecskip )
```

Transform vectors by the internal matrix.

This function transforms zero or more sets of vectors by the current internal matrix. Each vector is 3 single-precision floats long.

Parameters

invecs	The list of input vectors.
outvecs	The list of output vectors.
veccnt	How many vectors are in the list.
vecskip	Number of empty bytes between vectors.

mat_transform_sq()

```
void mat_transform_sq (
             void * input,
             void * output,
             int veccnt )
```

Transform vectors by the internal matrix into the store queues.

This function transforms one or more sets of vertices using the current internal matrix directly into the store queues. Each vertex is exactly 32-bytes long, and the non-xyz data that is with it will be copied over with the transformed coordinates. This is perfect, for instance, for transforming pvr_vertex_t vertices.

Parameters

input	The list of input vertices.
output	The output pointer.
veccnt	The number of vertices to transform.

Note

The QACR0 and QACR1 registers must be set appropriately BEFORE calling this function.

Author

Jim Ursetto

9.213 matrix.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/matrix.h
00004    Copyright (C) 2000 Megan Potter
00005    Copyright (C) 2013, 2014 Josh "PH3NOM" Pearson
00006    Copyright (C) 2018 Lawrence Sebald
00007
00008 */
00009
00010 /** \file   dc/matrix.h
00011     \brief  Basic matrix operations.
00012
00013     This file contains various basic matrix math functionality for using the
00014     SH4's matrix transformation unit. Higher level functionality, like the 3D
00015     functionality is built off of these operations.
00016
00017     \author Megan Potter
00018     \author Josh "PH3NOM" Pearson
00019     \see    dc/matrix3d.h
00020 */
00021
00022 #ifndef __DC_MATRIX_H
00023 #define __DC_MATRIX_H
00024
00025 #include <sys/cdefs.h>
00026 __BEGIN_DECLS
00027
00028 #include <dc/vector.h>
00029
00030 /** \brief  Copy the internal matrix to a memory one.
00031
00032     This function stores the current internal matrix to one in memory.
00033
00034     \warning
00035     \p out MUST be at least 8-byte aligned!
00036
00037     \note
00038     For best performance, 32-byte alignment of \p out is recommended.
00039
00040     \param  out             A pointer to where to store the matrix (must be at
00041                             least 8-byte aligned, should be 32-byte aligned).
00042 */
00043 void mat_store(matrix_t *out);
00044
00045 /** \brief  Copy a memory matrix into the internal one.
00046
00047     This function loads the internal matrix with the values of one in memory.
00048
00049     \warning
00050     \p out MUST be at least 8-byte aligned!
```

```
00051
00052     \note
00053     For best performance, 32-byte alignment of \p out is recommended.
00054
00055     \param  out             A pointer to where to load the matrix from (must be
00056                             at least 8-byte aligned, should be 32-byte aligned).
00057 */
00058 void mat_load(matrix_t *out);
00059
00060 /** \brief  Clear the internal matrix to identity.
00061
00062     This function clears the internal matrix to a standard identity matrix.
00063 */
00064 void mat_identity(void);
00065
00066 /** \brief  Apply a matrix.
00067
00068     This function multiplies a matrix in memory onto the internal matrix.
00069
00070     \warning
00071     \p src MUST be at least 8-byte aligned!
00072
00073     \note
00074     For best performance, 32-byte alignment of \p src is recommended.
00075
00076     \param  src             A poitner to the matrix to multiply.
00077 */
00078 void mat_apply(matrix_t *src);
00079
00080 /** \brief  Transform vectors by the internal matrix.
00081
00082     This function transforms zero or more sets of vectors by the current
00083     internal matrix. Each vector is 3 single-precision floats long.
00084
00085     \param  invecs          The list of input vectors.
00086     \param  outvecs         The list of output vectors.
00087     \param  veccnt          How many vectors are in the list.
00088     \param  vecskip         Number of empty bytes between vectors.
00089 */
00090 void mat_transform(vector_t *invecs, vector_t *outvecs, int veccnt, int vecskip);
00091
00092 /** \brief  Transform vectors by the internal matrix into the store queues.
00093
00094     This function transforms one or more sets of vertices using the current
00095     internal matrix directly into the store queues. Each vertex is exactly
00096     32-bytes long, and the non-xyz data that is with it will be copied over with
00097     the transformed coordinates. This is perfect, for instance, for transforming
00098     pvr_vertex_t vertices.
00099
00100     \param  input           The list of input vertices.
00101     \param  output          The output pointer.
00102     \param  veccnt          The number of vertices to transform.
00103     \note                   The \ref QACR0 and \ref QACR1 registers must be set
00104                             appropriately BEFORE calling this function.
00105     \author Jim Ursetto
00106 */
00107 void mat_transform_sq(void *input, void *output, int veccnt);
00108
00109 /** \brief  Macro to transform a single vertex by the internal matrix.
00110
00111     This macro is an inline assembly operation to transform a single vertex. It
00112     works most efficiently if the x value is in fr0, y is in fr1, and z is in
00113     fr2 before using the macro.
00114
00115     \param  x               The X coordinate to transform.
00116     \param  y               The Y coordinate to transform.
00117     \param  z               The Z coordinate to transform.
00118 */
00119 #define mat_trans_single(x, y, z) { \
00120         register float __x __asm__("fr0") = (x); \
00121         register float __y __asm__("fr1") = (y); \
00122         register float __z __asm__("fr2") = (z); \
00123         __asm__ __volatile__( \
00124                               "fldi1    fr3\n" \
00125                               "ftrv     xmtrx,fv0\n" \
00126                               "fldi1    fr2\n" \
00127                               "fdiv     fr3,fr2\n" \
00128                               "fmul     fr2,fr0\n" \
00129                               "fmul     fr2,fr1\n" \
00130                               : "=f" (__x), "=f" (__y), "=f" (__z) \
00131                               : "0" (__x), "1" (__y), "2" (__z) \
```

```
00132                               : "fr3" ); \
00133         x = __x; y = __y; z = __z; \
00134     }
00135
00136 /** \brief  Macro to transform a single vertex by the internal matrix.
00137
00138     This macro is an inline assembly operation to transform a single vertex. It
00139     works most efficiently if the x value is in fr0, y is in fr1, z is in
00140     fr2, and w is in fr3 before using the macro. This macro is similar to
00141     \ref mat_trans_single(), but this one allows an input to and preserves the
00142     Z/W value.
00143
00144     \param  x               The X coordinate to transform.
00145     \param  y               The Y coordinate to transform.
00146     \param  z               The Z coordinate to transform.
00147     \param  w               The W coordinate to transform.
00148 */
00149 #define mat_trans_single4(x, y, z, w) { \
00150         register float __x __asm__("fr0") = (x); \
00151         register float __y __asm__("fr1") = (y); \
00152         register float __z __asm__("fr2") = (z); \
00153         register float __w __asm__("fr3") = (w); \
00154         __asm__ __volatile__( \
00155                               "ftrv     xmtrx,fv0\n" \
00156                               "fdiv     fr3,fr0\n" \
00157                               "fdiv     fr3,fr1\n" \
00158                               "fdiv     fr3,fr2\n" \
00159                               "fldi1    fr4\n" \
00160                               "fdiv     fr3,fr4\n" \
00161                               "fmov     fr4,fr3\n" \
00162                               : "=f" (__x), "=f" (__y), "=f" (__z), "=f" (__w) \
00163                               : "0" (__x), "1" (__y), "2" (__z), "3" (__w) \
00164                               : "fr4" ); \
00165         x = __x; y = __y; z = __z; w = __w; \
00166     }
00167
00168 /** \brief  Macro to transform a single vertex by the internal matrix.
00169
00170     This macro is an inline assembly operation to transform a single vertex. It
00171     works most efficiently if the x value is in fr0, y is in fr1, and z is in
00172     fr2 before using the macro. This macro is similar to
00173     \ref mat_trans_single(), but this one leaves z/w instead of 1/w for the z
00174     component.
00175
00176     \param  x               The X coordinate to transform.
00177     \param  y               The Y coordinate to transform.
00178     \param  z               The Z coordinate to transform.
00179 */
00180 #define mat_trans_single3(x, y, z) { \
00181         register float __x __asm__("fr0") = (x); \
00182         register float __y __asm__("fr1") = (y); \
00183         register float __z __asm__("fr2") = (z); \
00184         __asm__ __volatile__( \
00185                               "fldi1    fr3\n" \
00186                               "ftrv     xmtrx,fv0\n" \
00187                               "fdiv     fr3,fr0\n" \
00188                               "fdiv     fr3,fr1\n" \
00189                               "fdiv     fr3,fr2\n" \
00190                               : "=f" (__x), "=f" (__y), "=f" (__z) \
00191                               : "0" (__x), "1" (__y), "2" (__z) \
00192                               : "fr3" ); \
00193         x = __x; y = __y; z = __z; \
00194     }
00195
00196 /** \brief  Macro to transform a single vertex by the internal matrix with no
00197             perspective division.
00198
00199     This macro is an inline assembly operation to transform a single vertex. It
00200     works most efficiently if the x value is in fr0, y is in fr1, z is in
00201     fr2, and w is in fr3 before using the macro. This macro is similar to
00202     \ref mat_trans_single(), but this one does not do any perspective division.
00203
00204     \param  x               The X coordinate to transform.
00205     \param  y               The Y coordinate to transform.
00206     \param  z               The Z coordinate to transform.
00207     \param  w               The W coordinate to transform.
00208 */
00209 #define mat_trans_nodiv(x, y, z, w) { \
00210         register float __x __asm__("fr0") = (x); \
00211         register float __y __asm__("fr1") = (y); \
00212         register float __z __asm__("fr2") = (z); \
```

```
00213         register float __w __asm__("fr3") = (w); \
00214         __asm__ __volatile__( \
00215                               "ftrv   xmtrx,fv0\n" \
00216                               : "=f" (__x), "=f" (__y), "=f" (__z), "=f" (__w) \
00217                               : "0" (__x), "1" (__y), "2" (__z), "3" (__w) ); \
00218         x = __x; y = __y; z = __z; w = __w; \
00219     }
00220
00221 /** \brief  Macro to transform a single 3d vertex coordinate by the internal
00222             matrix with no perspective division.
00223
00224     This macro is an inline assembly operation to transform a 3 float vertex
00225     coordinate. It works most efficiently if the x value is in fr12, y is in
00226     fr13, and z is in fr14 before using the macro. This macro is similar to
00227     \ref mat_trans_nodiv(), but this one sets the W component to 1 for use with
00228     a 3d vector.
00229
00230     \param  x               The X coordinate to transform.
00231     \param  y               The Y coordinate to transform.
00232     \param  z               The Z coordinate to transform.
00233 */
00234 #define mat_trans_single3_nodiv(x, y, z) { \
00235         register float __x __asm__("fr12") = (x); \
00236         register float __y __asm__("fr13") = (y); \
00237         register float __z __asm__("fr14") = (z); \
00238         __asm__ __volatile__( \
00239                               "fldi1 fr15\n" \
00240                               "ftrv  xmtrx, fv12\n" \
00241                               : "=f" (__x), "=f" (__y), "=f" (__z) \
00242                               : "0" (__x), "1" (__y), "2" (__z) \
00243                               : "fr15" ); \
00244         x = __x; y = __y; z = __z; \
00245     }
00246
00247 /** \brief  Macro to transform a single 3d vertex coordinate by the internal
00248             matrix with perspective division.
00249
00250     This macro is an inline assembly operation to transform a 3 float vertex
00251     coordinate. It works most efficiently if the x value is in fr12, y is in
00252     fr13, and z is in fr14 before using the macro. This macro is similar to
00253     \ref mat_trans_single(), but this one does not modify the input operands,
00254     instead storing the transformed vector to the output operands.
00255
00256     \param  x               The X coordinate to input transform.
00257     \param  y               The Y coordinate to input transform.
00258     \param  z               The Z coordinate to input transform.
00259     \param  x2              The X coordinate to output transform.
00260     \param  y2              The Y coordinate to output transform.
00261     \param  z2              The Z coordinate to output transform.
00262 */
00263 #define mat_trans_single3_nomod(x, y, z, x2, y2, z2) { \
00264         register float __x __asm__("fr12") = (x); \
00265         register float __y __asm__("fr13") = (y); \
00266         register float __z __asm__("fr14") = (z); \
00267         __asm__ __volatile__( \
00268                               "fldi1 fr15\n" \
00269                               "ftrv    xmtrx, fv12\n" \
00270                               "fldi1 fr14\n" \
00271                               "fdiv    fr15, fr14\n" \
00272                               "fmul    fr14, fr12\n" \
00273                               "fmul    fr14, fr13\n" \
00274                               : "=f" (__x), "=f" (__y), "=f" (__z) \
00275                               : "0" (__x), "1" (__y), "2" (__z) \
00276                               : "fr15" ); \
00277         x2 = __x; y2 = __y; z2 = __z; \
00278     }
00279
00280 /** \brief  Macro to transform a single 3d vertex coordinate by the internal
00281             matrix.
00282
00283     This macro is an inline assembly operation to transform a 3 float vertex
00284     coordinate. It works most efficiently if the x value is in fr12, y is in
00285     fr13, and z is in fr14 before using the macro. This macro is similar to
00286     \ref mat_trans_single3_nodiv(), but this one does not modify the input
00287     operands, instead storing the transformed vector to the output operands.
00288
00289     \param  x               The X coordinate to input transform.
00290     \param  y               The Y coordinate to input transform.
00291     \param  z               The Z coordinate to input transform.
00292     \param  x2              The X coordinate to output transform.
00293     \param  y2              The Y coordinate to output transform.
```

```
00294     \param  z2              The Z coordinate to output transform.
00295 */
00296 #define mat_trans_single3_nodiv_nomod(x, y, z, x2, y2, z2) { \
00297         register float __x __asm__("fr12") = (x); \
00298         register float __y __asm__("fr13") = (y); \
00299         register float __z __asm__("fr14") = (z); \
00300         __asm__ __volatile__( \
00301                                 "fldi1 fr15\n" \
00302                                 "ftrv  xmtrx, fv12\n" \
00303                                 : "=f" (__x), "=f" (__y), "=f" (__z) \
00304                                 : "0" (__x), "1" (__y), "2" (__z) \
00305                                 : "fr15" ); \
00306         x2 = __x; y2 = __y; z2 = __z; \
00307     }
00308
00309 /** \brief  Macro to transform a single 3d vertex coordinate by the internal
00310             matrix.
00311
00312     This macro is an inline assembly operation to transform a 3 float vertex
00313     coordinate. It works most efficiently if the x value is in fr12, y is in
00314     fr13, and z is in fr14 before using the macro. This macro is similar to
00315     \ref mat_trans_single3_nodiv(), but this one stores the W component of
00316     transform for later perspective divide.
00317
00318     \param  x               The X coordinate to transform.
00319     \param  y               The Y coordinate to transform.
00320     \param  z               The Z coordinate to transform.
00321     \param  w               The W coordinate output of transform.
00322 */
00323 #define mat_trans_single3_nodivw(x, y, z, w) { \
00324         register float __x __asm__("fr12") = (x); \
00325         register float __y __asm__("fr13") = (y); \
00326         register float __z __asm__("fr14") = (z); \
00327         register float __w __asm__("fr15") = 1.0f; \
00328         __asm__ __volatile__( \
00329                                 "ftrv  xmtrx, fv12\n" \
00330                                 : "=f" (__x), "=f" (__y), "=f" (__z), "=f" (__w) \
00331                                 : "0" (__x), "1" (__y), "2" (__z), "3" (__w) ); \
00332         x = __x; y = __y; z = __z; w = __w; \
00333     }
00334
00335 /** \brief  Macro to transform a single 3d vertex coordinate by the internal
00336             matrix both with and without perspective division.
00337
00338     This macro is an inline assembly operation to transform a 3 float vertex
00339     coordinate. It works most efficiently if the x value is in fr0, y is in fr1,
00340     and z is in fr2 before using the macro. This macro is similar to
00341     \ref mat_trans_single(), but this one is used for transforming input vertex
00342     with and without perspective division.
00343
00344     \param  x               The X coordinate to transform without perspective
00345                             divide.
00346     \param  y               The Y coordinate to transform without perspective
00347                             divide.
00348     \param  z               The Z coordinate to transform without perspective
00349                             divide.
00350     \param  xd              The X coordinate to output transform with
00351                             perspective divide.
00352     \param  yd              The Y coordinate to output transform with
00353                             perspective divide.
00354     \param  zd              The Z coordinate to output transform with
00355                             perspective divide.
00356 */
00357 #define mat_trans_single3_nodiv_div(x, y, z, xd, yd, zd) { \
00358         register float __x __asm__("fr0") = (x); \
00359         register float __y __asm__("fr1") = (y); \
00360         register float __z __asm__("fr2") = (z); \
00361         register float __xd __asm__("fr4"); \
00362         register float __yd __asm__("fr5"); \
00363         register float __zd __asm__("fr6"); \
00364         __asm__ __volatile__( \
00365                                 "fldi1 fr3\n" \
00366                                 "ftrv  xmtrx, fv0\n" \
00367                                 "fmov  fr0, fr4\n" \
00368                                 "fmov  fr1, fr5\n" \
00369                                 "fmov  fr3, fr7\n" \
00370                                 "fldi1 fr6\n" \
00371                                 "fdiv    fr7, fr6\n" \
00372                                 "fmul    fr6, fr4\n" \
00373                                 "fmul    fr6, fr5\n" \
00374                                 : "=f" (__x), "=f" (__y), "=f" (__z), \
```

```
00375                                 "=f" (__xd), "=f" (__yd), "=f" (__zd) \
00376                               : "0" (__x), "1" (__y), "2" (__z) \
00377                               : "fr3" ); \
00378         x = __x; y = __y; z = __z; xd = __xd; yd = __yd; zd = __zd; \
00379     }
00380
00381 /** \brief  Macro to transform a single vertex normal by the internal matrix.
00382
00383     This macro is an inline assembly operation to transform a 3 float vertex
00384     normal. It works most efficiently if the x value is in fr8, y is in fr9,
00385     and z is in fr10 before using the macro. This macro is similar to
00386     \ref mat_trans_nodiv(), but this one sets the W component to 0 in order to
00387     transform a vertex normal, rather than 1 for a vertex position.
00388
00389     \param  x               The X normal to transform.
00390     \param  y               The Y normal to transform.
00391     \param  z               The Z normal to transform.
00392 */
00393 #define mat_trans_normal3(x, y, z) { \
00394         register float __x __asm__("fr8") = (x); \
00395         register float __y __asm__("fr9") = (y); \
00396         register float __z __asm__("fr10") = (z); \
00397         __asm__ __volatile__( \
00398                               "fldi0 fr11\n" \
00399                               "ftrv  xmtrx, fv8\n" \
00400                               : "=f" (__x), "=f" (__y), "=f" (__z) \
00401                               : "0" (__x), "1" (__y), "2" (__z) \
00402                               : "fr11" ); \
00403         x = __x; y = __y; z = __z; \
00404     }
00405
00406 /** \brief  Macro to transform a single vertex normal by the internal matrix.
00407
00408     This macro is an inline assembly operation to transform a 3 float vertex
00409     normal. It works most efficiently if the x value is in fr8, y is in fr9,
00410     and z is in fr10 before using the macro. This macro is similar to
00411     \ref mat_trans_normal3(), but this one does not modify the input operands,
00412     instead storing the transformed vector to the output operands.
00413
00414     \param  x               The X normal to input transform.
00415     \param  y               The Y normal to input transform.
00416     \param  z               The Z normal to input transform.
00417     \param  x2              The X normal to output transform.
00418     \param  y2              The Y normal to output transform.
00419     \param  z2              The Z normal to output transform.
00420 */
00421 #define mat_trans_normal3_nomod(x, y, z, x2, y2, z2) { \
00422         register float __x __asm__("fr8") = (x); \
00423         register float __y __asm__("fr9") = (y); \
00424         register float __z __asm__("fr10") = (z); \
00425         __asm__ __volatile__( \
00426                               "fldi0 fr11\n" \
00427                               "ftrv  xmtrx, fv8\n" \
00428                               : "=f" (__x), "=f" (__y), "=f" (__z) \
00429                               : "0" (__x), "1" (__y), "2" (__z) \
00430                               : "fr11" ); \
00431         x2 = __x; y2 = __y; z2 = __z; \
00432     }
00433
00434 __END_DECLS
00435
00436 #endif  /* !__DC_MATRIX_H */
```

9.214 kernel/arch/dreamcast/include/dc/matrix3d.h File Reference

3D matrix operations.

```
#include <sys/cdefs.h>
#include <dc/matrix.h>
```

Functions

- void mat_rotate_x (float r)

  Rotate around the X-axis.
- void mat_rotate_y (float r)

  Rotate around the Y-axis.
- void mat_rotate_z (float r)

  Rotate around the Z-axis.
- void mat_rotate (float xr, float yr, float zr)

  Rotate around all axes.
- void mat_translate (float x, float y, float z)

  Perform a 3D translation.
- void mat_scale (float x, float y, float z)

  Perform a 3D scale operation.
- void mat_perspective (float xcenter, float ycenter, float cot_fovy_2, float znear, float zfar)

  Set up a perspective view frustum.
- void mat_lookat (const point_t *eye, const point_t *center, const vector_t *up)

  Set up a "camera".

9.214.1 Detailed Description

3D matrix operations.

This file contains various 3D matrix math functionality for using the SH4's matrix transformation unit.

Author

Megan Potter

Jordan DeLong

9.214.2 Function Documentation

mat_lookat()

```
void mat_lookat (
            const point_t * eye,
            const point_t * center,
            const vector_t * up )
```

Set up a "camera".

This function acts as the similarly named GL function to set up a "camera" by doing rotations/translations.

Parameters

eye	The eye coordinate.
center	The center coordinate.
up	The up vector.

mat_perspective()

```
void mat_perspective (
            float xcenter,
            float ycenter,
            float cot_fovy_2,
            float znear,
            float zfar )
```

Set up a perspective view frustum.

This function sets up a perspective view frustum for basic 3D usage.

Parameters

xcenter	Center of the X direction.
ycenter	Center of the Y direction.
cot_fovy_2	1.0 / tan(view_angle / 2).
znear	Near Z-plane.
zfar	Far Z-plane.

mat_rotate()

```
void mat_rotate (
            float xr,
            float yr,
            float zr )
```

Rotate around all axes.

This function sets up a rotation matrix around the X-axis, then around the Y, then around the Z.

Parameters

xr	The angle to rotate around the X-axis, in radians.
yr	The angle to rotate around the Y-axis, in radians.
zr	The angle to rotate around the Z-axis, in radians.

mat_rotate_x()

```
void mat_rotate_x (
            float r )
```

Rotate around the X-axis.

This function sets up a rotation matrix around the X-axis.

Parameters

r	The angle to rotate, in radians.

mat_rotate_y()

```
void mat_rotate_y (
            float r )
```

Rotate around the Y-axis.

This function sets up a rotation matrix around the Y-axis.

Parameters

r	The angle to rotate, in radians.

mat_rotate_z()

```
void mat_rotate_z (
            float r )
```

Rotate around the Z-axis.

This function sets up a rotation matrix around the Z-axis.

Parameters

r	The angle to rotate, in radians.

mat_scale()

```
void mat_scale (
            float x,
            float y,
            float z )
```

Perform a 3D scale operation.

This function sets up a scaling matrix with the specified parameters.

Parameters

x	The ratio to scale in X.
y	The ratio to scale in Y.
z	The ratio to scale in Z.

mat_translate()

```
void mat_translate (
             float x,
             float y,
             float z )
```

Perform a 3D translation.

This function sets up a translation matrix with the specified parameters.

Parameters

x	The amount to translate in X.
y	The amount to translate in Y.
z	The amount to translate in Z.

9.215 matrix3d.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    matrix3d.h
00004    (c)2000 Megan Potter and Jordan DeLong
00005
00006 */
00007
00008 /** \file   dc/matrix3d.h
00009     \brief  3D matrix operations.
00010
00011     This file contains various 3D matrix math functionality for using the SH4's
00012     matrix transformation unit.
00013
00014     \author Megan Potter
00015     \author Jordan DeLong
00016 */
00017
00018 #ifndef __KOS_MATRIX3D_H
00019 #define __KOS_MATRIX3D_H
00020
00021 #include <sys/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 #include <dc/matrix.h>
00025
00026 /** \brief  Rotate around the X-axis.
00027
00028     This function sets up a rotation matrix around the X-axis.
00029
00030     \param  r               The angle to rotate, in radians.
00031 */
00032 void mat_rotate_x(float r);
00033
00034 /** \brief  Rotate around the Y-axis.
00035
00036     This function sets up a rotation matrix around the Y-axis.
00037
00038     \param  r               The angle to rotate, in radians.
00039 */
00040 void mat_rotate_y(float r);
00041
00042 /** \brief  Rotate around the Z-axis.
00043
00044     This function sets up a rotation matrix around the Z-axis.
00045
```

```
00046     \param  r               The angle to rotate, in radians.
00047 */
00048 void mat_rotate_z(float r);
00049
00050 /** \brief  Rotate around all axes.
00051
00052     This function sets up a rotation matrix around the X-axis, then around the
00053     Y, then around the Z.
00054
00055     \param  xr              The angle to rotate around the X-axis, in radians.
00056     \param  yr              The angle to rotate around the Y-axis, in radians.
00057     \param  zr              The angle to rotate around the Z-axis, in radians.
00058 */
00059 void mat_rotate(float xr, float yr, float zr);
00060
00061 /** \brief  Perform a 3D translation.
00062
00063     This function sets up a translation matrix with the specified parameters.
00064
00065     \param  x               The amount to translate in X.
00066     \param  y               The amount to translate in Y.
00067     \param  z               The amount to translate in Z.
00068 */
00069 void mat_translate(float x, float y, float z);
00070
00071 /** \brief  Perform a 3D scale operation.
00072
00073     This function sets up a scaling matrix with the specified parameters.
00074
00075     \param  x               The ratio to scale in X.
00076     \param  y               The ratio to scale in Y.
00077     \param  z               The ratio to scale in Z.
00078 */
00079 void mat_scale(float x, float y, float z);
00080
00081 /** \brief  Set up a perspective view frustum.
00082
00083     This function sets up a perspective view frustum for basic 3D usage.
00084
00085     \param  xcenter         Center of the X direction.
00086     \param  ycenter         Center of the Y direction.
00087     \param  cot_fovy_2      1.0 / tan(view_angle / 2).
00088     \param  znear           Near Z-plane.
00089     \param  zfar            Far Z-plane.
00090 */
00091 void mat_perspective(float xcenter, float ycenter, float cot_fovy_2,
00092                      float znear, float zfar);
00093
00094 /** \brief  Set up a "camera".
00095
00096     This function acts as the similarly named GL function to set up a "camera"
00097     by doing rotations/translations.
00098
00099     \param  eye             The eye coordinate.
00100     \param  center          The center coordinate.
00101     \param  up              The up vector.
00102 */
00103 void mat_lookat(const point_t * eye, const point_t * center, const vector_t * up);
00104
00105 __END_DECLS
00106
00107 #endif  /* __KOS_MATRIX3D_H */
00108
00109
```

9.216 kernel/arch/dreamcast/include/dc/minifont.h File Reference

Simple font drawing functions.

```
#include <kos/cdefs.h>
#include <arch/types.h>
```

Functions

- int minifont_draw (uint16 *buffer, uint32 bufwidth, uint32 c)

  Draw a single character to a buffer.
- int minifont_draw_str (uint16 *b, uint32 bufwidth, const char *str)

  Draw a full string to any sort of buffer.

9.216.1 Detailed Description

Simple font drawing functions.

This file provides support for utilizing the "Naomi" font that is included in the KOS source code (in the utils/minifont.h file). This was designed for use when you really just want a *very* simple font to draw with.

Only ASCII characters are usable here. No other fancy encodings are supported, nor are any extended ASCII characters beyond the 7-bit range. Also, only 16-bit buffers (like what you would normally have for the framebuffer) are currently supported.

Author

Lawrence Sebald

9.216.2 Function Documentation

minifont_draw()

```
int minifont_draw (
            uint16 * buffer,
            uint32 bufwidth,
            uint32 c )
```

Draw a single character to a buffer.

This function draws a single character to the given buffer.

Parameters

buffer	The buffer to draw to (at least 8 x 16 pixels)
bufwidth	The width of the buffer in pixels
c	The character to draw

Returns

Amount of width covered in 16-bit increments.

minifont_draw_str()

```
int minifont_draw_str (
            uint16 * b,
            uint32 bufwidth,
            const char * str )
```

Draw a full string to any sort of buffer.

This function draws a NUL-terminated string to the given buffer. Only standard ASCII encoded strings are supported (no extended ASCII, ANSI, Unicode, JIS, EUC, etc).

Parameters

b	The buffer to draw to.
bufwidth	The width of the buffer in pixels.
str	The string to draw.

Returns

Amount of width covered in 16-bit increments.

9.217 minifont.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003     dc/minifont.h
00004     Copyright (C) 2020 Lawrence Sebald
00005
00006 */
00007
00008 /** \file   dc/minifont.h
00009     \brief  Simple font drawing functions.
00010
00011     This file provides support for utilizing the "Naomi" font that is included
00012     in the KOS source code (in the utils/minifont.h file). This was designed for
00013     use when you really just want a *very* simple font to draw with.
00014
00015     Only ASCII characters are usable here. No other fancy encodings are
00016     supported, nor are any extended ASCII characters beyond the 7-bit range.
00017     Also, only 16-bit buffers (like what you would normally have for the
00018     framebuffer) are currently supported.
00019
00020     \author Lawrence Sebald
00021 */
00022
00023 #ifndef __DC_MINIFONT_H
00024 #define __DC_MINIFONT_H
00025
00026 #include <kos/cdefs.h>
00027 __BEGIN_DECLS
00028
00029 #include <arch/types.h>
00030
00031 /** \brief  Draw a single character to a buffer.
00032
00033     This function draws a single character to the given buffer.
00034
00035     \param  buffer          The buffer to draw to (at least 8 x 16 pixels)
00036     \param  bufwidth        The width of the buffer in pixels
00037     \param  c               The character to draw
00038     \return                 Amount of width covered in 16-bit increments.
```

```
00039 */
00040 int minifont_draw(uint16 *buffer, uint32 bufwidth, uint32 c);
00041
00042 /** \brief  Draw a full string to any sort of buffer.
00043
00044     This function draws a NUL-terminated string to the given buffer. Only
00045     standard ASCII encoded strings are supported (no extended ASCII, ANSI,
00046     Unicode, JIS, EUC, etc).
00047
00048     \param  b               The buffer to draw to.
00049     \param  bufwidth           The width of the buffer in pixels.
00050     \param  str             The string to draw.
00051     \return                 Amount of width covered in 16-bit increments.
00052 */
00053 int minifont_draw_str(uint16 *b, uint32 bufwidth, const char *str);
00054
00055 __END_DECLS
00056
00057 #endif  /* __DC_MINIFONT_H */
```

9.218 kernel/arch/dreamcast/include/dc/modem/mconst.h File Reference

Constants used in the modem driver.

Macros

- #define MODEM_SPEED_AUTO 0x0
- #define MODEM_SPEED_1200 0x0
- #define MODEM_SPEED_2400 0x1
- #define MODEM_SPEED_4800 0x2
- #define MODEM_SPEED_7200 0x3
- #define MODEM_SPEED_9600 0x4
- #define MODEM_SPEED_12000 0x5
- #define MODEM_SPEED_14400 0x6
- #define MODEM_SPEED_16800 0x7
- #define MODEM_SPEED_19200 0x8
- #define MODEM_SPEED_21600 0x9
- #define MODEM_SPEED_24000 0xA
- #define MODEM_SPEED_26400 0xB
- #define MODEM_SPEED_28000 0xC
- #define MODEM_SPEED_31200 0xD
- #define MODEM_SPEED_33600 0xE
- #define MODEM_PROTOCOL_V17 0x0
- #define MODEM_PROTOCOL_V22 0x1
- #define MODEM_PROTOCOL_V22BIS 0x2
- #define MODEM_PROTOCOL_V32 0x3
- #define MODEM_PROTOCOL_V32BIS 0x4
- #define MODEM_PROTOCOL_V34 0x5
- #define MODEM_PROTOCOL_V8 0x6
- #define MODEM_SPEED_GET_PROTOCOL(x) ((modem_speed_t)(x) >> 4)

  Extract the protocol from a full speed/protocol value.
- #define MODEM_SPEED_GET_SPEED(x) ((modem_speed_t)(x) & 0xF)

  Extract the speed from a full speed/protocol value.
- #define MODEM_MAKE_SPEED(p, s) ((modem_speed_t)((((p) & 0xF) << 4) | ((s) & 0xF)))

  Combine a protocol and speed into a single value.

Typedefs

- typedef unsigned char modem_speed_t

  Modem speed/protocol value type.

9.218.1 Detailed Description

Constants used in the modem driver.

This file contains constants that are used for the modem driver. You should not ever need to include this file directly, as the main modem driver header file includes it automatically.

Generally, you will not need to use the stuff in this file yourself at all, as the main modem header file defines many useful combinations for you.

Author

Nick Kochakian

9.218.2 Macro Definition Documentation

MODEM_MAKE_SPEED

```
#define MODEM_MAKE_SPEED(
            p,
            s ) ((modem_speed_t)((((p) & 0xF) << 4) | ((s) & 0xF)))
```

Combine a protocol and speed into a single value.

Parameters

p	The protocol to use.
s	The speed to use.

Returns

The full speed/protocol value.

See also

Modem protocol values

Modem speed values

MODEM_SPEED_GET_PROTOCOL

```
#define MODEM_SPEED_GET_PROTOCOL(
              x ) ((modem_speed_t)(x) >> 4)
```

Extract the protocol from a full speed/protocol value.

Parameters

x	The speed/protocol value to look at.

Returns

The protocol in use.

See also

Modem protocol values

MODEM_SPEED_GET_SPEED

```
#define MODEM_SPEED_GET_SPEED(
              x ) ((modem_speed_t)(x) & 0xF)
```

Extract the speed from a full speed/protocol value.

Parameters

x	The speed/protocol value to look at.

Returns

The speed in use.

See also

Modem speed values

9.218.3 Typedef Documentation

modem_speed_t

```
typedef unsigned char modem_speed_t
```

Modem speed/protocol value type.

9.219 mconst.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    mconst.h
00004    Copyright (C)2002, 2004 Nick Kochakian
00005
00006    Distributed under the terms of the KOS license.
00007
00008 */
00009
00010 /** \file   dc/modem/mconst.h
00011     \brief  Constants used in the modem driver.
00012
00013     This file contains constants that are used for the modem driver. You should
00014     not ever need to include this file directly, as the main modem driver header
00015     file includes it automatically.
00016
00017     Generally, you will not need to use the stuff in this file yourself at all,
00018     as the main modem header file defines many useful combinations for you.
00019
00020     \author Nick Kochakian
00021 */
00022
00023 /* Modem constants are defined here. Automatically included by modem.h */
00024 #ifndef __MODEM_MCONST_H
00025 #define __MODEM_MCONST_H
00026
00027 /* Each speed constant contains information about the data rate in bps and the
00028    protocol that's being used. The first 4 bits identify the the speed that's
00029    being used, and the last 4 bits identify the protocol.
00030
00031    And don't try to create your own custom speeds from these, you'll cause
00032    something very bad to happen. Only use the MODEM_SPEED_* constants defined
00033    in modem.h! */
00034
00035 /** \defgroup   modem_speeds    Modem speed values
00036
00037     This group defines the available speed values that are able to be used with
00038     the Dreamcast's modem. The actual speed value consists of one of these in
00039     the lower 4 bits and one of the protocols in the upper 4 bits. Don't try to
00040     use any speeds not defined here, as bad things may happen.
00041
00042     It should be fairly obvious from the names what the speeds are (they're all
00043     expressed in bits per second).
00044
00045     @{
00046 */
00047 #define MODEM_SPEED_AUTO  0x0
00048 #define MODEM_SPEED_1200  0x0
00049 #define MODEM_SPEED_2400  0x1
00050 #define MODEM_SPEED_4800  0x2
00051 #define MODEM_SPEED_7200  0x3
00052 #define MODEM_SPEED_9600  0x4
00053 #define MODEM_SPEED_12000 0x5
00054 #define MODEM_SPEED_14400 0x6
00055 #define MODEM_SPEED_16800 0x7
00056 #define MODEM_SPEED_19200 0x8
00057 #define MODEM_SPEED_21600 0x9
00058 #define MODEM_SPEED_24000 0xA
00059 #define MODEM_SPEED_26400 0xB
00060 #define MODEM_SPEED_28000 0xC
00061 #define MODEM_SPEED_31200 0xD
00062 #define MODEM_SPEED_33600 0xE
00063 /** @} */
00064
00065 /** \defgroup   modem_protocols Modem protocol values
00066
00067     This group defines the available protocol values that are able to be used
00068     with the Dreamcast's modem. The actual speed value consists of one of these
00069     in the upper 4 bits and one of the speeds in the lower 4 bits. Don't try to
00070     use any protocols not defined here, as bad things may happen.
00071
00072     It should be fairly obvious from the names what the protocols that will be
00073     used are.
00074
00075     @{
00076 */
```

```
00077 #define MODEM_PROTOCOL_V17    0x0
00078 #define MODEM_PROTOCOL_V22    0x1
00079 #define MODEM_PROTOCOL_V22BIS 0x2
00080 #define MODEM_PROTOCOL_V32    0x3
00081 #define MODEM_PROTOCOL_V32BIS 0x4
00082 #define MODEM_PROTOCOL_V34    0x5
00083 #define MODEM_PROTOCOL_V8     0x6
00084 /** @} */
00085
00086 /** \brief  Extract the protocol from a full speed/protocol value.
00087     \param  x           The speed/protocol value to look at.
00088     \return             The protocol in use.
00089     \see                modem_protocols
00090 */
00091 #define MODEM_SPEED_GET_PROTOCOL(x) ((modem_speed_t)(x) » 4)
00092
00093 /** \brief  Extract the speed from a full speed/protocol value.
00094     \param  x           The speed/protocol value to look at.
00095     \return             The speed in use.
00096     \see                modem_speeds
00097 */
00098 #define MODEM_SPEED_GET_SPEED(x)    ((modem_speed_t)(x) & 0xF)
00099
00100 /** \brief  Combine a protocol and speed into a single value.
00101     \param  p           The protocol to use.
00102     \param  s           The speed to use.
00103     \return             The full speed/protocol value.
00104     \see                modem_protocols
00105     \see                modem_speeds
00106 */
00107 #define MODEM_MAKE_SPEED(p, s)      ((modem_speed_t)((((p) & 0xF) « 4) | ((s) & 0xF)))
00108
00109 /** \brief  Modem speed/protocol value type. */
00110 typedef unsigned char modem_speed_t;
00111
00112 #endif
```

9.220 kernel/arch/dreamcast/include/dc/modem/modem.h File Reference

Definitions to use the Dreamcast modem.

```
#include "mconst.h"
```

Macros

- #define MODEM_MODE_REMOTE 0

  Connect to a remote modem.
- #define MODEM_MODE_ANSWER 1

  Answer a call when a ring is detected.
- #define MODEM_MODE_NULL 255

  Modem not in use. Do not attempt to set this mode yourself!
- #define MODEM_SPEED_V22BIS_1200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V22BIS, MODEM_SPEED_1200)

  1200bps, V.22bis
- #define MODEM_SPEED_V22BIS_2400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V22BIS, MODEM_SPEED_2400)

  2400bps, V.22bis
- #define MODEM_SPEED_V22_1200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V22, MODEM_SPEED_1200)

  1200bps, V.22
- #define MODEM_SPEED_V32_4800 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32, MODEM_SPEED_4800)

  4800bps, V.32
- #define MODEM_SPEED_V32_9600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32, MODEM_SPEED_9600)

9600bps, V.32

- #define MODEM_SPEED_V32BIS_7200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32BIS, MODEM_SPEED_7200)

  7200bps, V.32bis
- #define MODEM_SPEED_V32BIS_12000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32BIS, MODEM_SPEED_12000)

  12000bps, V.32bis
- #define MODEM_SPEED_V32BIS_14400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32BIS, MODEM_SPEED_14400)

  14400bps, V.32bis
- #define MODEM_SPEED_V8_2400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_2400)

  2400bps, V.8
- #define MODEM_SPEED_V8_4800 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_4800)

  4800bps, V.8
- #define MODEM_SPEED_V8_7200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_7200)

  7200bps, V.8
- #define MODEM_SPEED_V8_9600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_9600)

  9600bps, V.8
- #define MODEM_SPEED_V8_12000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_12000)

  12000bps, V.8
- #define MODEM_SPEED_V8_14400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_14400)

  14400bps, V.8
- #define MODEM_SPEED_V8_16800 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_16800)

  16800bps, V.8
- #define MODEM_SPEED_V8_19200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_19200)

  19200bps, V.8
- #define MODEM_SPEED_V8_21600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_21600)

  21600bps, V.8
- #define MODEM_SPEED_V8_24000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_24000)

  24000bps, V.8
- #define MODEM_SPEED_V8_26400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_26400)

  26400bps, V.8
- #define MODEM_SPEED_V8_28000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_28000)

  28000bps, V.8
- #define MODEM_SPEED_V8_31200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_31200)

  31200bps, V.8
- #define MODEM_SPEED_V8_33600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_33600)

  33600bps, V.8
- #define MODEM_SPEED_V8_AUTO MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_1200)

  Automatically set speed, V.8.

Typedefs

- typedef void(∗ MODEMEVENTHANDLERPROC) (modemEvent_t event)

  Type of a modem event handling function.

Enumerations

- enum modemEvent_t {
  MODEM_EVENT_CONNECTION_FAILED = 0 , MODEM_EVENT_CONNECTED , MODEM_EVENT_DISCONNECTED , MODEM_EVENT_RX_NOT_EMPTY ,
  MODEM_EVENT_OVERFLOW , MODEM_EVENT_TX_EMPTY }

  Modem event types.

Functions

- int modem_init (void)

  Initialize the modem.
- void modem_shutdown (void)

  Shut down the modem.
- int modem_set_mode (int mode, modem_speed_t speed)

  Set the modem up for the specified mode.
- int modem_wait_dialtone (int ms_timeout)

  Wait for the modem to detect a dialtone.
- int modem_dial (const char *digits)

  Dial the specified number on the modem.
- void modem_set_event_handler (MODEMEVENTHANDLERPROC eventHandler)

  Set the event handler for the modem.
- void modem_disconnect (void)

  Disconnect the modem.
- int modem_is_connecting (void)

  Check if the modem is connecting.
- int modem_is_connected (void)

  Check if the modem is connected.
- unsigned long modem_get_connection_rate (void)

  Get the connection rate that the modem is connected at.
- int modem_read_data (unsigned char *data, int size)

  Read data from the modem buffers.
- int modem_write_data (unsigned char *data, int size)

  Write data to the modem buffers.
- int modem_has_data (void)

  Check if the modem has data waiting to be read.

9.220.1 Detailed Description

Definitions to use the Dreamcast modem.

This file contains functions and constants to be used with the Dreamcast modem driver.

Author

Nick Kochakian

9.220.2 Typedef Documentation

MODEMEVENTHANDLERPROC

```
typedef void(* MODEMEVENTHANDLERPROC) (modemEvent_t event)
```

Type of a modem event handling function.

9.220.3 Enumeration Type Documentation

modemEvent_t

```
enum modemEvent_t
```

Modem event types.

These are teh events that a modem event handler should be expected to receive at any given point in time.

Enumerator

MODEM_EVENT_CONNECTION_FAILED	The modem tried to establish a connection, but failed.
MODEM_EVENT_CONNECTED	A connection has been established.
MODEM_EVENT_DISCONNECTED	The remote modem dropped the connection.
MODEM_EVENT_RX_NOT_EMPTY	New data has entered the previously empty receive buffer.
MODEM_EVENT_OVERFLOW	The receive buffer overflowed and was cleared.
MODEM_EVENT_TX_EMPTY	The transmit buffer has been emptied.

9.220.4 Function Documentation

modem_dial()

```
int modem_dial (
            const char * digits )
```

Dial the specified number on the modem.

Parameters

digits	The number to dial, as a string.

Return values

0	On failure.
1	On success.

modem_disconnect()

```
void modem_disconnect (
             void  )
```

Disconnect the modem.

This function instructs the modem to disconnect from the remote modem.

modem_get_connection_rate()

```
unsigned long modem_get_connection_rate (
             void  )
```

Get the connection rate that the modem is connected at.

Returns

The connection rate in bits per second.

modem_has_data()

```
int modem_has_data (
             void  )
```

Check if the modem has data waiting to be read.

Returns

0 if no data available, non-zero otherwise.

modem_init()

```
int modem_init (
             void  )
```

Initialize the modem.

This function initializes the modem for use.

Return values

0	On failure.
1	On success.

modem_is_connected()

```
int modem_is_connected (
            void  )
```

Check if the modem is connected.

Returns

0 if the modem is not currently connected, non-zero otherwise.

modem_is_connecting()

```
int modem_is_connecting (
            void  )
```

Check if the modem is connecting.

Returns

0 if the modem is not currently connecting, non-zero otherwise.

modem_read_data()

```
int modem_read_data (
            unsigned char * data,
            int size )
```

Read data from the modem buffers.

Parameters

data	The buffer to read into.
size	The maximum number of bytes to read.

Returns

The actual number of bytes read.

modem_set_event_handler()

```
void modem_set_event_handler (
            MODEMEVENTHANDLERPROC eventHandler )
```

Set the event handler for the modem.

This function sets up an event handler for when things happen on the modem.

Parameters

eventHandler	The function to call when an event occurs.

modem_set_mode()

```
int modem_set_mode (
            int mode,
            modem_speed_t speed )
```

Set the modem up for the specified mode.

This function sets up the modem's registers for the specified mode and speed combination.

Parameters

mode	The mode to use.
speed	The speed to use.

See also

Modes of operation of the Dreamcast modem.
Modem V.22bis modes
Modem V.22 modes
Modem V.32 modes
Modem V.32 bis modes
Modem V.8 modes

modem_shutdown()

```
void modem_shutdown (
            void  )
```

Shut down the modem.

This function shuts down the modem after it has been initialized, resetting all of the registers to their defaults.

modem_wait_dialtone()

```
int modem_wait_dialtone (
            int ms_timeout )
```

Wait for the modem to detect a dialtone.

This function waits for a dialtone to be detected on the modem.

Parameters

ms_timeout	The number of milliseconds to wait, in multiples of 100.

Return values

0	If a dialtone is detected before timeout.
-1	If no dialtone is detected.

modem_write_data()

```
int modem_write_data (
            unsigned char * data,
            int size )
```

Write data to the modem buffers.

Parameters

data	The buffer to write from.
size	The maximum number of bytes to write.

Returns

The actual number of bytes written.

9.221 modem.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    modem.h
00004    Copyright (C)2002, 2004 Nick Kochakian
00005
00006    Distributed under the terms of the KOS license.
00007
00008 */
00009
00010 /** \file   dc/modem/modem.h
00011     \brief  Definitions to use the Dreamcast modem.
00012
00013     This file contains functions and constants to be used with the Dreamcast
00014     modem driver.
00015
00016     \author Nick Kochakian
00017 */
00018
00019 #ifndef __DC_MODEM_MODEM_H
00020 #define __DC_MODEM_MODEM_H
00021
00022 #include "mconst.h"
00023
00024 /** \defgroup   modem_modes     Modes of operation of the Dreamcast modem.
00025
00026     This group defines the modes that the Dreamcast modem can be in at any given
```

```
00027     point in time.
00028
00029     @{
00030 */
00031 /** \brief  Connect to a remote modem. */
00032 #define MODEM_MODE_REMOTE 0
00033
00034 /** \brief  Answer a call when a ring is detected. */
00035 #define MODEM_MODE_ANSWER 1
00036
00037 /** \brief  Modem not in use. Do not attempt to set this mode yourself! */
00038 #define MODEM_MODE_NULL   255
00039 /** @} */
00040
00041 /** \defgroup   modem_v22bis    Modem V.22bis modes
00042     @{
00043 */
00044 /** \brief  1200bps, V.22bis */
00045 #define MODEM_SPEED_V22BIS_1200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V22BIS, MODEM_SPEED_1200)
00046
00047 /** \brief  2400bps, V.22bis */
00048 #define MODEM_SPEED_V22BIS_2400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V22BIS, MODEM_SPEED_2400)
00049 /** @} */
00050
00051 /** \defgroup   modem_v22       Modem V.22 modes
00052     @{
00053 */
00054 /** \brief  1200bps, V.22 */
00055 #define MODEM_SPEED_V22_1200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V22, MODEM_SPEED_1200)
00056 /** @} */
00057
00058 /** \defgroup   modem_v32       Modem V.32 modes
00059     @{
00060 */
00061 /** \brief  4800bps, V.32 */
00062 #define MODEM_SPEED_V32_4800 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32, MODEM_SPEED_4800)
00063
00064 /** \brief  9600bps, V.32 */
00065 #define MODEM_SPEED_V32_9600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32, MODEM_SPEED_9600)
00066 /** @} */
00067
00068 /** \defgroup   modem_v32bis    Modem V.32 bis modes
00069     @{
00070 */
00071 /** \brief  7200bps, V.32bis */
00072 #define MODEM_SPEED_V32BIS_7200  MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32BIS, MODEM_SPEED_7200)
00073
00074 /** \brief  12000bps, V.32bis */
00075 #define MODEM_SPEED_V32BIS_12000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32BIS, MODEM_SPEED_12000)
00076
00077 /** \brief  14400bps, V.32bis */
00078 #define MODEM_SPEED_V32BIS_14400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V32BIS, MODEM_SPEED_14400)
00079 /** @} */
00080
00081 /** \defgroup   modem_v8        Modem V.8 modes
00082     @{
00083 */
00084 /** \brief  2400bps, V.8 */
00085 #define MODEM_SPEED_V8_2400  MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_2400)
00086
00087 /** \brief  4800bps, V.8 */
00088 #define MODEM_SPEED_V8_4800  MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_4800)
00089
00090 /** \brief  7200bps, V.8 */
00091 #define MODEM_SPEED_V8_7200  MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_7200)
00092
00093 /** \brief  9600bps, V.8 */
00094 #define MODEM_SPEED_V8_9600  MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_9600)
00095
00096 /** \brief  12000bps, V.8 */
00097 #define MODEM_SPEED_V8_12000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_12000)
00098
00099 /** \brief  14400bps, V.8 */
00100 #define MODEM_SPEED_V8_14400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_14400)
00101
00102 /** \brief  16800bps, V.8 */
00103 #define MODEM_SPEED_V8_16800 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_16800)
00104
00105 /** \brief  19200bps, V.8 */
00106 #define MODEM_SPEED_V8_19200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_19200)
00107
```

```
00108 /** \brief  21600bps, V.8 */
00109 #define MODEM_SPEED_V8_21600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_21600)
00110
00111 /** \brief  24000bps, V.8 */
00112 #define MODEM_SPEED_V8_24000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_24000)
00113
00114 /** \brief  26400bps, V.8 */
00115 #define MODEM_SPEED_V8_26400 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_26400)
00116
00117 /** \brief  28000bps, V.8 */
00118 #define MODEM_SPEED_V8_28000 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_28000)
00119
00120 /** \brief  31200bps, V.8 */
00121 #define MODEM_SPEED_V8_31200 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_31200)
00122
00123 /** \brief  33600bps, V.8 */
00124 #define MODEM_SPEED_V8_33600 MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_33600)
00125
00126 /** \brief  Automatically set speed, V.8 */
00127 #define MODEM_SPEED_V8_AUTO  MODEM_MAKE_SPEED(MODEM_PROTOCOL_V8, MODEM_SPEED_1200)
00128 /** @} */
00129
00130 /** \brief  Modem event types.
00131
00132     These are teh events that a modem event handler should be expected to
00133     receive at any given point in time.
00134 */
00135 typedef enum {
00136     /** \brief  The modem tried to establish a connection, but failed.*/
00137     MODEM_EVENT_CONNECTION_FAILED = 0,
00138
00139     /** \brief  A connection has been established. */
00140     MODEM_EVENT_CONNECTED,
00141
00142     /** \brief  The remote modem dropped the connection. */
00143     MODEM_EVENT_DISCONNECTED,
00144
00145     /** \brief  New data has entered the previously empty receive buffer. */
00146     MODEM_EVENT_RX_NOT_EMPTY,
00147
00148     /** \brief  The receive buffer overflowed and was cleared. */
00149     MODEM_EVENT_OVERFLOW,
00150
00151     /** \brief  The transmit buffer has been emptied. */
00152     MODEM_EVENT_TX_EMPTY
00153 } modemEvent_t;
00154
00155 /** \brief  Type of a modem event handling function. */
00156 typedef void (*MODEMEVENTHANDLERPROC)(modemEvent_t event);
00157
00158 /* From modem.c */
00159 /** \brief  Initialize the modem.
00160
00161     This function initializes the modem for use.
00162
00163     \retval 0               On failure.
00164     \retval 1               On success.
00165 */
00166 int           modem_init(void);
00167
00168 /** \brief  Shut down the modem.
00169
00170     This function shuts down the modem after it has been initialized, resetting
00171     all of the registers to their defaults.
00172 */
00173 void          modem_shutdown(void);
00174
00175 /** \brief  Set the modem up for the specified mode.
00176
00177     This function sets up the modem's registers for the specified mode and speed
00178     combination.
00179
00180     \param  mode            The mode to use.
00181     \param  speed           The speed to use.
00182     \see    modem_modes
00183     \see    modem_v22bis
00184     \see    modem_v22
00185     \see    modem_v32
00186     \see    modem_v32bis
00187     \see    modem_v8
00188 */
```

```
00189 int           modem_set_mode(int mode, modem_speed_t speed);
00190
00191 /** \brief  Wait for the modem to detect a dialtone.
00192
00193     This function waits for a dialtone to be detected on the modem.
00194
00195     \param  ms_timeout      The number of milliseconds to wait, in multiples of 100.
00196     \retval 0               If a dialtone is detected before timeout.
00197     \retval -1              If no dialtone is detected.
00198 */
00199 int           modem_wait_dialtone(int ms_timeout);
00200
00201 /** \brief  Dial the specified number on the modem.
00202     \param  digits          The number to dial, as a string.
00203     \retval 0               On failure.
00204     \retval 1               On success.
00205 */
00206 int           modem_dial(const char *digits);
00207
00208 /** \brief  Set the event handler for the modem.
00209
00210     This function sets up an event handler for when things happen on the modem.
00211
00212     \param  eventHandler    The function to call when an event occurs.
00213 */
00214 void          modem_set_event_handler(MODEMEVENTHANDLERPROC eventHandler);
00215
00216 /** \brief  Disconnect the modem.
00217
00218     This function instructs the modem to disconnect from the remote modem.
00219 */
00220 void          modem_disconnect(void);
00221
00222 /** \brief  Check if the modem is connecting.
00223     \return                 0 if the modem is not currently connecting, non-zero
00224                             otherwise.
00225 */
00226 int           modem_is_connecting(void);
00227
00228 /** \brief  Check if the modem is connected.
00229     \return                 0 if the modem is not currently connected, non-zero
00230                             otherwise.
00231 */
00232 int           modem_is_connected(void);
00233
00234 /** \brief  Get the connection rate that the modem is connected at.
00235     \return                 The connection rate in bits per second.
00236 */
00237 unsigned long modem_get_connection_rate(void);
00238
00239 /* From mdata.c */
00240 /** \brief  Read data from the modem buffers.
00241     \param  data            The buffer to read into.
00242     \param  size            The maximum number of bytes to read.
00243     \return                 The actual number of bytes read.
00244 */
00245 int modem_read_data(unsigned char *data, int size);
00246
00247 /** \brief  Write data to the modem buffers.
00248     \param  data            The buffer to write from.
00249     \param  size            The maximum number of bytes to write.
00250     \return                 The actual number of bytes written.
00251 */
00252 int modem_write_data(unsigned char *data, int size);
00253
00254 /** \brief  Check if the modem has data waiting to be read.
00255     \return                 0 if no data available, non-zero otherwise.
00256 */
00257 int modem_has_data(void);
00258
00259 #endif
```

9.222 kernel/arch/dreamcast/include/dc/net/broadband_adapter.h File Reference

Broadband Adapter support.

```
#include <sys/cdefs.h>
```

Macros

- #define RT_IDR0 0x00

  MAC address 0 (RW 32bit, RO 16/8)
- #define RT_IDR1 0x01

  MAC address 1 (Read-only)
- #define RT_IDR2 0x02

  MAC address 2 (Read-only)
- #define RT_IDR3 0x03

  MAC address 3 (Read-only)
- #define RT_IDR4 0x04

  MAC address 4 (RW 32bit, RO 16/8)
- #define RT_IDR5 0x05

  MAC address 5 (Read-only)
- #define RT_RES06 0x06

  Reserved.
- #define RT_RES07 0x07

  Reserved.
- #define RT_MAR0 0x08

  Multicast filter 0 (RW 32bit, RO 16/8)
- #define RT_MAR1 0x09

  Multicast filter 1 (Read-only)
- #define RT_MAR2 0x0A

  Multicast filter 2 (Read-only)
- #define RT_MAR3 0x0B

  Multicast filter 3 (Read-only)
- #define RT_MAR4 0x0C

  Multicast filter 4 (RW 32bit, RO 16/8)
- #define RT_MAR5 0x0D

  Multicast filter 5 (Read-only)
- #define RT_MAR6 0x0E

  Multicast filter 6 (Read-only)
- #define RT_MAR7 0x0F

  Multicast filter 7 (Read-only)
- #define RT_TXSTATUS0 0x10

  Transmit status 0 (32bit only)
- #define RT_TXSTATUS1 0x14

  Transmit status 1 (32bit only)
- #define RT_TXSTATUS2 0x18

  Transmit status 2 (32bit only)
- #define RT_TXSTATUS3 0x1C

  Transmit status 3 (32bit only)
- #define RT_TXADDR0 0x20

  Tx descriptor 0 (32bit only)

- #define RT_TXADDR1 0x24

  Tx descriptor 1 (32bit only)
- #define RT_TXADDR2 0x28

  Tx descriptor 2 (32bit only)
- #define RT_TXADDR3 0x2C

  Tx descriptor 3 (32bit only)
- #define RT_RXBUF 0x30

  Receive buffer start address (32bit only)
- #define RT_RXEARLYCNT 0x34

  Early Rx byte count (RO 16bit)
- #define RT_RXEARLYSTATUS 0x36

  Early Rx status (RO)
- #define RT_CHIPCMD 0x37

  Command register.
- #define RT_RXBUFTAIL 0x38

  Current address of packet read (queue tail) (16bit only)
- #define RT_RXBUFHEAD 0x3A

  Current buffer address (queue head) (RO 16bit)
- #define RT_INTRMASK 0x3C

  Interrupt mask (16bit only)
- #define RT_INTRSTATUS 0x3E

  Interrupt status (16bit only)
- #define RT_TXCONFIG 0x40

  Tx config (32bit only)
- #define RT_RXCONFIG 0x44

  Rx config (32bit only)
- #define RT_TIMER 0x48

  A general purpose counter, any write clears (32bit only)
- #define RT_RXMISSED 0x4C

  24 bits valid, write clears (32bit only)
- #define RT_CFG9346 0x50

  93C46 command register
- #define RT_CONFIG0 0x51

  Configuration reg 0.
- #define RT_CONFIG1 0x52

  Configuration reg 1.
- #define RT_RES53 0x53

  Reserved.
- #define RT_TIMERINT 0x54

  Timer interrupt register (32bit only)
- #define RT_MEDIASTATUS 0x58

  Media status register.
- #define RT_CONFIG3 0x59

  Config register 3.
- #define RT_CONFIG4 0x5A

  Config register 4.
- #define RT_RES5B 0x5B

Reserved.

- #define RT_MULTIINTR 0x5C

  Multiple interrupt select (32bit only)
- #define RT_RERID 0x5E

  PCI Revision ID (10h) (Read-only)
- #define RT_RES5F 0x5F

  Reserved.
- #define RT_MII_TSAD 0x60

  Transmit status of all descriptors (RO 16bit)
- #define RT_MII_BMCR 0x62

  Basic Mode Control Register (16bit only)
- #define RT_MII_BMSR 0x64

  Basic Mode Status Register (RO 16bit)
- #define RT_AS_ADVERT 0x66

  Auto-negotiation advertisement reg (16bit only)
- #define RT_AS_LPAR 0x68

  Auto-negotiation link partner reg (RO 16bit)
- #define RT_AS_EXPANSION 0x6A

  Auto-negotiation expansion reg (RO 16bit)
- #define RT_CONFIG5 0xD8

  Config register 5.
- #define RT_MII_RESET 0x8000

  Reset the MII chip.
- #define RT_MII_RES4000 0x4000

  Reserved.
- #define RT_MII_SPD_SET 0x2000

  1 for 100 0 for 10. Ignored if AN enabled.
- #define RT_MII_AN_ENABLE 0x1000

  Enable auto-negotiation.
- #define RT_MII_RES0800 0x0800

  Reserved.
- #define RT_MII_RES0400 0x0400

  Reserved.
- #define RT_MII_AN_START 0x0200

  Start auto-negotiation.
- #define RT_MII_DUPLEX 0x0100

  1 for full 0 for half. Ignored if AN enabled.
- #define RT_MII_LINK 0x0004

  Link is present.
- #define RT_MII_AN_CAPABLE 0x0008

  Can do auto negotiation.
- #define RT_MII_AN_COMPLETE 0x0020

  Auto-negotiation complete.
- #define RT_MII_10_HALF 0x0800

  Can do 10Mbit half duplex.
- #define RT_MII_10_FULL 0x1000

  Can do 10Mbit full.

- #define RT_MII_100_HALF 0x2000

  Can do 100Mbit half.
- #define RT_MII_100_FULL 0x4000

  Can do 100Mbit full.
- #define RT_CMD_RESET 0x10

  Reset the RTL8139C.
- #define RT_CMD_RX_ENABLE 0x08

  Enable Rx.
- #define RT_CMD_TX_ENABLE 0x04

  Enable Tx.
- #define RT_CMD_RX_BUF_EMPTY 0x01

  Empty the Rx buffer.
- #define RT_INT_PCIERR 0x8000

  PCI Bus error.
- #define RT_INT_TIMEOUT 0x4000

  Set when TCTR reaches TimerInt value.
- #define RT_INT_RXFIFO_OVERFLOW 0x0040

  Rx FIFO overflow.
- #define RT_INT_RXFIFO_UNDERRUN 0x0020

  Packet underrun / link change.
- #define RT_INT_LINK_CHANGE 0x0020

  Packet underrun / link change.
- #define RT_INT_RXBUF_OVERFLOW 0x0010

  Rx BUFFER overflow.
- #define RT_INT_TX_ERR 0x0008

  Tx error.
- #define RT_INT_TX_OK 0x0004

  Tx OK.
- #define RT_INT_RX_ERR 0x0002

  Rx error.
- #define RT_INT_RX_OK 0x0001

  Rx OK.
- #define RT_INT_RX_ACK (RT_INT_RXFIFO_OVERFLOW | RT_INT_RXBUF_OVERFLOW | RT_INT_RX_OK)

  Composite RX bits we check for while doing an RX interrupt.
- #define RT_TX_CARRIER_LOST 0x80000000

  Carrier sense lost.
- #define RT_TX_ABORTED 0x40000000

  Transmission aborted.
- #define RT_TX_OUT_OF_WINDOW 0x20000000

  Out of window collision.
- #define RT_TX_STATUS_OK 0x00008000

  Status ok: a good packet was transmitted.
- #define RT_TX_UNDERRUN 0x00004000

  Transmit FIFO underrun.
- #define RT_TX_HOST_OWNS 0x00002000

  Set to 1 when DMA operation is completed.
- #define RT_TX_SIZE_MASK 0x00001fff

Descriptor size mask.

- #define RT_RX_MULTICAST 0x00008000

  Multicast packet.
- #define RT_RX_PAM 0x00004000

  Physical address matched.
- #define RT_RX_BROADCAST 0x00002000

  Broadcast address matched.
- #define RT_RX_BAD_SYMBOL 0x00000020

  Invalid symbol in 100TX packet.
- #define RT_RX_RUNT 0x00000010

  Packet size is <64 bytes.
- #define RT_RX_TOO_LONG 0x00000008

  Packet size is >4K bytes.
- #define RT_RX_CRC_ERR 0x00000004

  CRC error.
- #define RT_RX_FRAME_ALIGN 0x00000002

  Frame alignment error.
- #define RT_RX_STATUS_OK 0x00000001

  Status ok: a good packet was received.
- #define RT_CONFIG1_LED1 0x80

  XXX DC bba has no LED, maybe repurposed.
- #define RT_CONFIG1_LED0 0x40

  XXX DC bba has no LED, maybe repurposed.
- #define RT_CONFIG1_DVRLOAD 0x20

  Sets the Driver as loaded.
- #define RT_CONFIG1_LWACT 0x10

  LWAKE active mode. Default 0.
- #define RT_CONFIG1_MEMMAP 0x08

  Registers mapped to PCI mem space. Read Only.
- #define RT_CONFIG1_IOMAP 0x04

  Registers mapped to PCI I/O space. Read Only.
- #define RT_CONFIG1_VPD 0x02

  Enable Vital Product Data.
- #define RT_CONFIG1_PMEn 0x01

  Power Management Enable.
- #define RT_CONFIG4_RxFIFIOAC 0x80

  Auto-clear the Rx FIFO overflow.
- #define RT_CONFIG4_AnaOff 0x40

  Turn off analog power. Default 0.
- #define RT_CONFIG4_LongWF 0x20

  Long Wake-up Frames.
- #define RT_CONFIG4_LWPME 0x10

  LWake vs PMEB.
- #define RT_CONFIG4_RES08 0x08

  Reserved.
- #define RT_CONFIG4_LWPTN 0x04

  LWAKE Pattern.

- #define RT_CONFIG4_RES02 0x02

  Reserved.
- #define RT_CONFIG4_PBWake 0x01

  Disable pre-Boot Wakeup.
- #define RT_CONFIG5_RES80 0x80

  Reserved.
- #define RT_CONFIG5_BWF 0x40

  Enable Broadcast Wakeup Frame. Default 0.
- #define RT_CONFIG5_MWF 0x20

  Enable Multicast Wakeup Frame. Default 0.
- #define RT_CONFIG5_UWF 0x10

  Enable Unicast Wakeup Frame. Default 0.
- #define RT_CONFIG5_FIFOAddr 0x08

  Set FIFO address pointer. For testing only.
- #define RT_CONFIG5_LDPS 0x04

  Disable Link Down Power Saving mode.
- #define RT_CONFIG5_LANW 0x02

  Enable LANWake signal.
- #define RT_CONFIG5_PME_STS 0x01

  Allow PCI reset to set PME_Status bit.
- #define BBA_TX_OK 0

  Transmit success.
- #define BBA_TX_ERROR -1

  Transmit error.
- #define BBA_TX_AGAIN -2

  Retry transmit again.
- #define BBA_TX_NOWAIT 0

  Don't block waiting for the transfer.
- #define BBA_TX_WAIT 1

  Wait, if needed on transfer.

Typedefs

- typedef void(∗ eth_rx_callback_t) (uint8 ∗pkt, int len)

  Receive packet callback function type.

Functions

- void bba_get_mac (uint8 ∗arr)

  Retrieve the MAC Address of the attached BBA.
- void bba_set_rx_callback (eth_rx_callback_t cb)

  Set the ethernet packet receive callback.
- int bba_tx (const uint8 ∗pkt, int len, int wait)

  Transmit a single packet.

9.222.1 Detailed Description

Broadband Adapter support.

This file contains declarations related to support for the HIT-0400 "Broadband Adapter". There's not really anything that users will generally have to deal with in here.

Author

Megan Potter

9.222.2 Macro Definition Documentation

BBA_TX_NOWAIT

```
#define BBA_TX_NOWAIT 0
```

Don't block waiting for the transfer.

BBA_TX_WAIT

```
#define BBA_TX_WAIT 1
```

Wait, if needed on transfer.

9.222.3 Typedef Documentation

eth_rx_callback_t

```
typedef void(* eth_rx_callback_t) (uint8 *pkt, int len)
```

Receive packet callback function type.

When a packet is received by the BBA, the callback function will be called to handle it.

Parameters

pkt	A pointer to the packet in question.
len	The length, in bytes, of the packet.

9.222.4 Function Documentation

bba_get_mac()

```
void bba_get_mac (
            uint8 * arr )
```

Retrieve the MAC Address of the attached BBA.

This function reads the MAC Address of the BBA and places it in the buffer passed in. The resulting data is undefined if no BBA is connected.

Parameters

arr	The array to read the MAC into.

bba_set_rx_callback()

```
void bba_set_rx_callback (
            eth_rx_callback_t cb )
```

Set the ethernet packet receive callback.

This function sets the function called when a packet is received by the BBA. Generally, this inputs into the network layer.

Parameters

cb	A pointer to the new callback function.

bba_tx()

```
int bba_tx (
            const uint8 * pkt,
            int len,
            int wait )
```

Transmit a single packet.

This function transmits a single packet on the bba, waiting for the link to become stable, if requested.

Parameters

pkt	The packet to transmit.
len	The length of the packet, in bytes.
wait	BBA_TX_WAIT if you don't mind blocking for the all clear to transmit, BBA_TX_NOWAIT otherwise.

Return values

BBA_TX_OK	On success.
BBA_TX_ERROR	If there was an error transmitting the packet.
BBA_TX_AGAIN	If BBA_TX_NOWAIT was specified and it is not ok to transmit right now.

9.223 broadband_adapter.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/net/broadband_adapter.h
00004    Copyright (C) 2001-2002 Megan Potter
00005
00006 */
00007
00008 /** \file   dc/net/broadband_adapter.h
00009     \brief  Broadband Adapter support.
00010
00011     This file contains declarations related to support for the HIT-0400
00012     "Broadband Adapter". There's not really anything that users will generally
00013     have to deal with in here.
00014
00015     \author Megan Potter
00016 */
00017
00018 #ifndef __DC_NET_BROADBAND_ADAPTER_H
00019 #define __DC_NET_BROADBAND_ADAPTER_H
00020
00021 #include <sys/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 /** \defgroup bba_regs  RTL8139C Register Definitions
00025
00026     The default assumption is that these are all RW at any aligned size unless
00027     otherwise noted. ex (RW 32bit, RO 16/8) indicates read/write at 32bit and
00028     read-only at 16 or 8bits.
00029
00030     @{
00031 */
00032 #define RT_IDR0             0x00    /**< \brief MAC address 0 (RW 32bit, RO 16/8) */
00033 #define RT_IDR1             0x01    /**< \brief MAC address 1 (Read-only) */
00034 #define RT_IDR2             0x02    /**< \brief MAC address 2 (Read-only) */
00035 #define RT_IDR3             0x03    /**< \brief MAC address 3 (Read-only) */
00036 #define RT_IDR4             0x04    /**< \brief MAC address 4 (RW 32bit, RO 16/8) */
00037 #define RT_IDR5             0x05    /**< \brief MAC address 5 (Read-only) */
00038 #define RT_RES06            0x06    /**< \brief Reserved */
00039 #define RT_RES07            0x07    /**< \brief Reserved */
00040 #define RT_MAR0             0x08    /**< \brief Multicast filter 0 (RW 32bit, RO 16/8) */
00041 #define RT_MAR1             0x09    /**< \brief Multicast filter 1 (Read-only) */
00042 #define RT_MAR2             0x0A    /**< \brief Multicast filter 2 (Read-only) */
00043 #define RT_MAR3             0x0B    /**< \brief Multicast filter 3 (Read-only) */
00044 #define RT_MAR4             0x0C    /**< \brief Multicast filter 4 (RW 32bit, RO 16/8) */
00045 #define RT_MAR5             0x0D    /**< \brief Multicast filter 5 (Read-only) */
00046 #define RT_MAR6             0x0E    /**< \brief Multicast filter 6 (Read-only) */
00047 #define RT_MAR7             0x0F    /**< \brief Multicast filter 7 (Read-only) */
00048 #define RT_TXSTATUS0        0x10    /**< \brief Transmit status 0 (32bit only) */
00049 #define RT_TXSTATUS1        0x14    /**< \brief Transmit status 1 (32bit only) */
00050 #define RT_TXSTATUS2        0x18    /**< \brief Transmit status 2 (32bit only) */
00051 #define RT_TXSTATUS3        0x1C    /**< \brief Transmit status 3 (32bit only) */
00052 #define RT_TXADDR0          0x20    /**< \brief Tx descriptor 0 (32bit only) */
00053 #define RT_TXADDR1          0x24    /**< \brief Tx descriptor 1 (32bit only) */
00054 #define RT_TXADDR2          0x28    /**< \brief Tx descriptor 2 (32bit only) */
00055 #define RT_TXADDR3          0x2C    /**< \brief Tx descriptor 3 (32bit only) */
00056 #define RT_RXBUF            0x30    /**< \brief Receive buffer start address (32bit only) */
00057 #define RT_RXEARLYCNT       0x34    /**< \brief Early Rx byte count (RO 16bit) */
00058 #define RT_RXEARLYSTATUS    0x36    /**< \brief Early Rx status (RO) */
00059 #define RT_CHIPCMD          0x37    /**< \brief Command register */
00060 #define RT_RXBUFTAIL        0x38    /**< \brief Current address of packet read (queue tail) (16bit only)
      */
00061 #define RT_RXBUFHEAD        0x3A    /**< \brief Current buffer address (queue head) (RO 16bit) */
```

```
00062 #define RT_INTRMASK         0x3C    /**< \brief Interrupt mask (16bit only) */
00063 #define RT_INTRSTATUS       0x3E    /**< \brief Interrupt status (16bit only) */
00064 #define RT_TXCONFIG         0x40    /**< \brief Tx config (32bit only) */
00065 #define RT_RXCONFIG         0x44    /**< \brief Rx config (32bit only) */
00066 #define RT_TIMER            0x48    /**< \brief A general purpose counter, any write clears (32bit only)
      */
00067 #define RT_RXMISSED         0x4C    /**< \brief 24 bits valid, write clears (32bit only) */
00068 #define RT_CFG9346          0x50    /**< \brief 93C46 command register */
00069 #define RT_CONFIG0          0x51    /**< \brief Configuration reg 0 */
00070 #define RT_CONFIG1          0x52    /**< \brief Configuration reg 1 */
00071 #define RT_RES53            0x53    /**< \brief Reserved */
00072 #define RT_TIMERINT         0x54    /**< \brief Timer interrupt register (32bit only) */
00073 #define RT_MEDIASTATUS      0x58    /**< \brief Media status register */
00074 #define RT_CONFIG3          0x59    /**< \brief Config register 3 */
00075 #define RT_CONFIG4          0x5A    /**< \brief Config register 4 */
00076 #define RT_RES5B            0x5B    /**< \brief Reserved */
00077 #define RT_MULTIINTR        0x5C    /**< \brief Multiple interrupt select (32bit only) */
00078 #define RT_RERID            0x5E    /**< \brief PCI Revision ID (10h) (Read-only) */
00079 #define RT_RES5F            0x5F    /**< \brief Reserved */
00080 #define RT_MII_TSAD         0x60    /**< \brief Transmit status of all descriptors (RO 16bit) */
00081 #define RT_MII_BMCR         0x62    /**< \brief Basic Mode Control Register (16bit only) */
00082 #define RT_MII_BMSR         0x64    /**< \brief Basic Mode Status Register (RO 16bit) */
00083 #define RT_AS_ADVERT        0x66    /**< \brief Auto-negotiation advertisement reg (16bit only) */
00084 #define RT_AS_LPAR          0x68    /**< \brief Auto-negotiation link partner reg (RO 16bit) */
00085 #define RT_AS_EXPANSION     0x6A    /**< \brief Auto-negotiation expansion reg (RO 16bit) */
00086
00087 #define RT_CONFIG5          0xD8    /**< \brief Config register 5 */
00088 /** @} */
00089
00090 /** \defgroup bba_miicb RTL8139C MII (media independent interface) control bits
00091     @{
00092 */
00093 #define RT_MII_RESET        0x8000  /**< \brief Reset the MII chip */
00094 #define RT_MII_RES4000      0x4000  /**< \brief Reserved */
00095 #define RT_MII_SPD_SET      0x2000  /**< \brief 1 for 100 0 for 10. Ignored if AN enabled. */
00096 #define RT_MII_AN_ENABLE    0x1000  /**< \brief Enable auto-negotiation */
00097 #define RT_MII_RES0800      0x0800  /**< \brief Reserved */
00098 #define RT_MII_RES0400      0x0400  /**< \brief Reserved */
00099 #define RT_MII_AN_START     0x0200  /**< \brief Start auto-negotiation */
00100 #define RT_MII_DUPLEX       0x0100  /**< \brief 1 for full 0 for half. Ignored if AN enabled. */
00101
00102 /** @} */
00103
00104 /** \defgroup bba_miisb RTL8139C MII (media independent interface) status bits
00105     @{
00106 */
00107 #define RT_MII_LINK         0x0004  /**< \brief Link is present */
00108 #define RT_MII_AN_CAPABLE   0x0008  /**< \brief Can do auto negotiation */
00109 #define RT_MII_AN_COMPLETE  0x0020  /**< \brief Auto-negotiation complete */
00110 #define RT_MII_10_HALF      0x0800  /**< \brief Can do 10Mbit half duplex */
00111 #define RT_MII_10_FULL      0x1000  /**< \brief Can do 10Mbit full */
00112 #define RT_MII_100_HALF     0x2000  /**< \brief Can do 100Mbit half */
00113 #define RT_MII_100_FULL     0x4000  /**< \brief Can do 100Mbit full */
00114 /** @} */
00115
00116 /** \defgroup bba_cbits RTL8139C Command Bits
00117
00118     OR appropriate bit values together and write into the RT_CHIPCMD register to
00119     execute the command.
00120
00121     @{
00122 */
00123 #define RT_CMD_RESET        0x10 /**< \brief Reset the RTL8139C */
00124 #define RT_CMD_RX_ENABLE    0x08 /**< \brief Enable Rx */
00125 #define RT_CMD_TX_ENABLE    0x04 /**< \brief Enable Tx */
00126 #define RT_CMD_RX_BUF_EMPTY 0x01 /**< \brief Empty the Rx buffer */
00127 /** @} */
00128
00129 /** \defgroup bba_ibits RTL8139C Interrupt Status bits
00130     @{
00131 */
00132 #define RT_INT_PCIERR           0x8000  /**< \brief PCI Bus error */
00133 #define RT_INT_TIMEOUT          0x4000  /**< \brief Set when TCTR reaches TimerInt value */
00134 #define RT_INT_RXFIFO_OVERFLOW  0x0040  /**< \brief Rx FIFO overflow */
00135 #define RT_INT_RXFIFO_UNDERRUN  0x0020  /**< \brief Packet underrun / link change */
00136 #define RT_INT_LINK_CHANGE      0x0020  /**< \brief Packet underrun / link change */
00137 #define RT_INT_RXBUF_OVERFLOW   0x0010  /**< \brief Rx BUFFER overflow */
00138 #define RT_INT_TX_ERR           0x0008  /**< \brief Tx error */
00139 #define RT_INT_TX_OK            0x0004  /**< \brief Tx OK */
00140 #define RT_INT_RX_ERR           0x0002  /**< \brief Rx error */
00141 #define RT_INT_RX_OK            0x0001  /**< \brief Rx OK */
```

```
00142
00143 /** \brief  Composite RX bits we check for while doing an RX interrupt. */
00144 #define RT_INT_RX_ACK (RT_INT_RXFIFO_OVERFLOW | RT_INT_RXBUF_OVERFLOW | RT_INT_RX_OK)
00145 /** @} */
00146
00147 /** \defgroup bba_tbits RTL8139C transmit status bits
00148     @{
00149 */
00150 #define RT_TX_CARRIER_LOST  0x80000000  /**< \brief Carrier sense lost */
00151 #define RT_TX_ABORTED       0x40000000  /**< \brief Transmission aborted */
00152 #define RT_TX_OUT_OF_WINDOW 0x20000000  /**< \brief Out of window collision */
00153 #define RT_TX_STATUS_OK     0x00008000  /**< \brief Status ok: a good packet was transmitted */
00154 #define RT_TX_UNDERRUN      0x00004000  /**< \brief Transmit FIFO underrun */
00155 #define RT_TX_HOST_OWNS     0x00002000  /**< \brief Set to 1 when DMA operation is completed */
00156 #define RT_TX_SIZE_MASK     0x00001fff  /**< \brief Descriptor size mask */
00157 /** @} */
00158
00159 /** \defgroup bba_rbits RTL8139C receive status bits
00160     @{
00161 */
00162 #define RT_RX_MULTICAST     0x00008000  /**< \brief Multicast packet */
00163 #define RT_RX_PAM           0x00004000  /**< \brief Physical address matched */
00164 #define RT_RX_BROADCAST     0x00002000  /**< \brief Broadcast address matched */
00165 #define RT_RX_BAD_SYMBOL    0x00000020  /**< \brief Invalid symbol in 100TX packet */
00166 #define RT_RX_RUNT          0x00000010  /**< \brief Packet size is <64 bytes */
00167 #define RT_RX_TOO_LONG      0x00000008  /**< \brief Packet size is >4K bytes */
00168 #define RT_RX_CRC_ERR       0x00000004  /**< \brief CRC error */
00169 #define RT_RX_FRAME_ALIGN   0x00000002  /**< \brief Frame alignment error */
00170 #define RT_RX_STATUS_OK     0x00000001  /**< \brief Status ok: a good packet was received */
00171 /** @} */
00172
00173 /** \defgroup bba_config1bits RTL8139C Config Register 1 bits
00174
00175     From RTL8139C(L) datasheet v1.4
00176
00177     @{
00178 */
00179 #define RT_CONFIG1_LED1     0x80 /**< \brief XXX DC bba has no LED, maybe repurposed. */
00180 #define RT_CONFIG1_LED0     0x40 /**< \brief XXX DC bba has no LED, maybe repurposed. */
00181 #define RT_CONFIG1_DVRLOAD  0x20 /**< \brief Sets the Driver as loaded. */
00182 #define RT_CONFIG1_LWACT    0x10 /**< \brief LWAKE active mode. Default 0. */
00183 #define RT_CONFIG1_MEMMAP   0x08 /**< \brief Registers mapped to PCI mem space. Read Only */
00184 #define RT_CONFIG1_IOMAP    0x04 /**< \brief Registers mapped to PCI I/O space. Read Only */
00185 #define RT_CONFIG1_VPD      0x02 /**< \brief Enable Vital Product Data. */
00186 #define RT_CONFIG1_PMEn     0x01 /**< \brief Power Management Enable */
00187 /** @} */
00188
00189 /** \defgroup bba_config4bits RTL8139C Config Register 4 bits
00190
00191     From RTL8139C(L) datasheet v1.4. Only RT_CONFIG4_RxFIFIOAC is used.
00192
00193     @{
00194 */
00195 #define RT_CONFIG4_RxFIFIOAC 0x80 /**< \brief Auto-clear the Rx FIFO overflow. */
00196 #define RT_CONFIG4_AnaOff    0x40 /**< \brief Turn off analog power. Default 0. */
00197 #define RT_CONFIG4_LongWF    0x20 /**< \brief Long Wake-up Frames. */
00198 #define RT_CONFIG4_LWPME     0x10 /**< \brief LWake vs PMEB. */
00199 #define RT_CONFIG4_RES08     0x08 /**< \brief Reserved. */
00200 #define RT_CONFIG4_LWPTN     0x04 /**< \brief LWAKE Pattern. */
00201 #define RT_CONFIG4_RES02     0x02 /**< \brief Reserved. */
00202 #define RT_CONFIG4_PBWake    0x01 /**< \brief Disable pre-Boot Wakeup. */
00203 /** @} */
00204
00205 /** \defgroup bba_config5bits RTL8139C Config Register 5 bits
00206
00207     From RTL8139C(L) datasheet v1.4. Only RT_CONFIG5_LDPS is used.
00208
00209     @{
00210 */
00211 #define RT_CONFIG5_RES80    0x80 /**< \brief Reserved. */
00212 #define RT_CONFIG5_BWF      0x40 /**< \brief Enable Broadcast Wakeup Frame. Default 0. */
00213 #define RT_CONFIG5_MWF      0x20 /**< \brief Enable Multicast Wakeup Frame. Default 0. */
00214 #define RT_CONFIG5_UWF      0x10 /**< \brief Enable Unicast Wakeup Frame. Default 0. */
00215 #define RT_CONFIG5_FIFOAddr 0x08 /**< \brief Set FIFO address pointer. For testing only. */
00216 #define RT_CONFIG5_LDPS     0x04 /**< \brief Disable Link Down Power Saving mode. */
00217 #define RT_CONFIG5_LANW     0x02 /**< \brief Enable LANWake signal. */
00218 #define RT_CONFIG5_PME_STS  0x01 /**< \brief Allow PCI reset to set PME_Status bit. */
00219 /** @} */
00220
00221 /** \brief  Retrieve the MAC Address of the attached BBA.
00222
```

```
00223     This function reads the MAC Address of the BBA and places it in the buffer
00224     passed in. The resulting data is undefined if no BBA is connected.
00225
00226     \param  arr             The array to read the MAC into.
00227 */
00228 void bba_get_mac(uint8 *arr);
00229
00230 /** \brief  Receive packet callback function type.
00231
00232     When a packet is received by the BBA, the callback function will be called
00233     to handle it.
00234
00235     \param  pkt             A pointer to the packet in question.
00236     \param  len             The length, in bytes, of the packet.
00237 */
00238 typedef void (*eth_rx_callback_t)(uint8 *pkt, int len);
00239
00240 /** \brief  Set the ethernet packet receive callback.
00241
00242     This function sets the function called when a packet is received by the BBA.
00243     Generally, this inputs into the network layer.
00244
00245     \param  cb              A pointer to the new callback function.
00246 */
00247 void bba_set_rx_callback(eth_rx_callback_t cb);
00248
00249 /** \defgroup bba_txrv  Return values from bba_tx().
00250     @{
00251 */
00252 #define BBA_TX_OK       0   /**< \brief Transmit success */
00253 #define BBA_TX_ERROR    -1  /**< \brief Transmit error */
00254 #define BBA_TX_AGAIN    -2  /**< \brief Retry transmit again */
00255 /** @} */
00256
00257 #define BBA_TX_NOWAIT   0   /**< \brief Don't block waiting for the transfer. */
00258 #define BBA_TX_WAIT     1   /**< \brief Wait, if needed on transfer. */
00259
00260 /** \brief  Transmit a single packet.
00261
00262     This function transmits a single packet on the bba, waiting for the link to
00263     become stable, if requested.
00264
00265     \param  pkt             The packet to transmit.
00266     \param  len             The length of the packet, in bytes.
00267     \param  wait            BBA_TX_WAIT if you don't mind blocking for the
00268                             all clear to transmit, BBA_TX_NOWAIT otherwise.
00269
00270     \retval BBA_TX_OK       On success.
00271     \retval BBA_TX_ERROR    If there was an error transmitting the packet.
00272     \retval BBA_TX_AGAIN    If BBA_TX_NOWAIT was specified and it is not ok to
00273                             transmit right now.
00274 */
00275 int bba_tx(const uint8 *pkt, int len, int wait);
00276
00277 /* \cond */
00278 /* Initialize */
00279 int bba_init(void);
00280
00281 /* Shutdown */
00282 int bba_shutdown(void);
00283 /* \endcond */
00284
00285 __END_DECLS
00286
00287 #endif  /* __DC_NET_BROADBAND_ADAPTER_H */
00288
```

9.224 kernel/arch/dreamcast/include/dc/net/lan_adapter.h File Reference

LAN Adapter support.

```
#include <sys/cdefs.h>
#include <kos/net.h>
```

9.224.1 Detailed Description

LAN Adapter support.

This file contains declarations related to support for the HIT-0300 "LAN Adapter". There's not really anything that users will generally have to deal with in here.

Author

Megan Potter

9.225 lan_adapter.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/net/lan_adapter.h
00004    Copyright (C) 2002 Megan Potter
00005
00006 */
00007
00008 /** \file   dc/net/lan_adapter.h
00009     \brief  LAN Adapter support.
00010
00011     This file contains declarations related to support for the HIT-0300 "LAN
00012     Adapter". There's not really anything that users will generally have to deal
00013     with in here.
00014
00015     \author Megan Potter
00016 */
00017
00018 #ifndef __DC_NET_LAN_ADAPTER_H
00019 #define __DC_NET_LAN_ADAPTER_H
00020
00021 #include <sys/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 #include <kos/net.h>
00025
00026 /* \cond */
00027 /* Initialize */
00028 int la_init(void);
00029
00030 /* Shutdown */
00031 int la_shutdown(void);
00032 /* \endcond */
00033
00034 __END_DECLS
00035
00036 #endif  /* __DC_NET_LAN_ADAPTER_H */
00037
```

9.226 kernel/arch/dreamcast/include/dc/pvr.h File Reference

Low-level PVR (3D hardware) interface.

```
#include <sys/cdefs.h>
#include <arch/memory.h>
#include <arch/types.h>
#include <dc/sq.h>
#include <kos/img.h>
```

Data Structures

- struct pvr_poly_cxt_t

  PVR polygon context.
- struct pvr_sprite_cxt_t

  PVR sprite context.
- struct pvr_poly_hdr_t

  PVR polygon header.
- struct pvr_poly_ic_hdr_t

  PVR polygon header with intensity color.
- struct pvr_poly_mod_hdr_t

  PVR polygon header to be used with modifier volumes.
- struct pvr_sprite_hdr_t

  PVR polygon header specifically for sprites.
- struct pvr_mod_hdr_t

  Modifier volume header.
- struct pvr_vertex_t

  Generic PVR vertex type.
- struct pvr_vertex_pcm_t

  PVR vertex type: Non-textured, packed color, affected by modifier volume.
- struct pvr_vertex_tpcm_t

  PVR vertex type: Textured, packed color, affected by modifer volume.
- struct pvr_sprite_txr_t

  PVR vertex type: Textured sprite.
- struct pvr_sprite_col_t

  PVR vertex type: Untextured sprite.
- struct pvr_modifier_vol_t

  PVR vertex type: Modifier volume.
- struct pvr_init_params_t

  PVR initialization structure.
- struct pvr_stats_t

  PVR statistics structure.

Macros

- #define PVR_LIST_OP_POLY 0

  Opaque polygon list.
- #define PVR_LIST_OP_MOD 1

  Opaque modifier list.
- #define PVR_LIST_TR_POLY 2

  Translucent polygon list.
- #define PVR_LIST_TR_MOD 3

  Translucent modifier list.
- #define PVR_LIST_PT_POLY 4

  Punch-thru polygon list.
- #define PVR_SHADE_FLAT 0

  Use flat shading.

- #define PVR_SHADE_GOURAUD 1

  Use Gouraud shading.
- #define PVR_DEPTHCMP_NEVER 0

  Never pass.
- #define PVR_DEPTHCMP_LESS 1

  Less than.
- #define PVR_DEPTHCMP_EQUAL 2

  Equal to.
- #define PVR_DEPTHCMP_LEQUAL 3

  Less than or equal to.
- #define PVR_DEPTHCMP_GREATER 4

  Greater than.
- #define PVR_DEPTHCMP_NOTEQUAL 5

  Not equal to.
- #define PVR_DEPTHCMP_GEQUAL 6

  Greater than or equal to.
- #define PVR_DEPTHCMP_ALWAYS 7

  Always pass.
- #define PVR_CULLING_NONE 0

  Disable culling.
- #define PVR_CULLING_SMALL 1

  Cull if small.
- #define PVR_CULLING_CCW 2

  Cull if counterclockwise.
- #define PVR_CULLING_CW 3

  Cull if clockwise.
- #define PVR_DEPTHWRITE_ENABLE 0

  Update the Z value.
- #define PVR_DEPTHWRITE_DISABLE 1

  Do not update the Z value.
- #define PVR_TEXTURE_DISABLE 0

  Disable texturing.
- #define PVR_TEXTURE_ENABLE 1

  Enable texturing.
- #define PVR_BLEND_ZERO 0

  None of this color.
- #define PVR_BLEND_ONE 1

  All of this color.
- #define PVR_BLEND_DESTCOLOR 2

  Destination color.
- #define PVR_BLEND_INVDESTCOLOR 3

  Inverse of destination color.
- #define PVR_BLEND_SRCALPHA 4

  Blend with source alpha.
- #define PVR_BLEND_INVSRCALPHA 5

  Blend with inverse source alpha.
- #define PVR_BLEND_DESTALPHA 6

Blend with destination alpha.

- #define PVR_BLEND_INVDESTALPHA 7

  Blend with inverse destination alpha.

- #define PVR_BLEND_DISABLE 0

  Disable blending.

- #define PVR_BLEND_ENABLE 1

  Enable blending.

- #define PVR_FOG_TABLE 0

  Table fog.

- #define PVR_FOG_VERTEX 1

  Vertex fog.

- #define PVR_FOG_DISABLE 2

  Disable fog.

- #define PVR_FOG_TABLE2 3

  Table fog mode 2.

- #define PVR_USERCLIP_DISABLE 0

  Disable clipping.

- #define PVR_USERCLIP_INSIDE 2

  Enable clipping inside area.

- #define PVR_USERCLIP_OUTSIDE 3

  Enable clipping outside area.

- #define PVR_CLRCLAMP_DISABLE 0

  Disable color clamping.

- #define PVR_CLRCLAMP_ENABLE 1

  Enable color clamping.

- #define PVR_SPECULAR_DISABLE 0

  Disable offset colors.

- #define PVR_SPECULAR_ENABLE 1

  Enable offset colors.

- #define PVR_ALPHA_DISABLE 0

  Disable alpha blending.

- #define PVR_ALPHA_ENABLE 1

  Enable alpha blending.

- #define PVR_TXRALPHA_ENABLE 0

  Enable alpha blending.

- #define PVR_TXRALPHA_DISABLE 1

  Disable alpha blending.

- #define PVR_UVFLIP_NONE 0

  No flipped coordinates.

- #define PVR_UVFLIP_V 1

  Flip V only.

- #define PVR_UVFLIP_U 2

  Flip U only.

- #define PVR_UVFLIP_UV 3

  Flip U and V.

- #define PVR_UVCLAMP_NONE 0

  Disable clamping.

- #define PVR_UVCLAMP_V 1

  Clamp V only.
- #define PVR_UVCLAMP_U 2

  Clamp U only.
- #define PVR_UVCLAMP_UV 3

  Clamp U and V.
- #define PVR_FILTER_NONE 0

  No filtering (point sample)
- #define PVR_FILTER_NEAREST 0

  No filtering (point sample)
- #define PVR_FILTER_BILINEAR 2

  Bilinear interpolation.
- #define PVR_FILTER_TRILINEAR1 4

  Trilinear interpolation pass 1.
- #define PVR_FILTER_TRILINEAR2 6

  Trilinear interpolation pass 2.
- #define PVR_MIPBIAS_NORMAL PVR_MIPBIAS_1_00 /∗ txr_mipmap_bias ∗/
- #define PVR_MIPBIAS_0_25 1
- #define PVR_MIPBIAS_0_50 2
- #define PVR_MIPBIAS_0_75 3
- #define PVR_MIPBIAS_1_00 4
- #define PVR_MIPBIAS_1_25 5
- #define PVR_MIPBIAS_1_50 6
- #define PVR_MIPBIAS_1_75 7
- #define PVR_MIPBIAS_2_00 8
- #define PVR_MIPBIAS_2_25 9
- #define PVR_MIPBIAS_2_50 10
- #define PVR_MIPBIAS_2_75 11
- #define PVR_MIPBIAS_3_00 12
- #define PVR_MIPBIAS_3_25 13
- #define PVR_MIPBIAS_3_50 14
- #define PVR_MIPBIAS_3_75 15
- #define PVR_TXRENV_REPLACE 0

  C = Ct, A = At.
- #define PVR_TXRENV_MODULATE 1

  C = Cs ∗ Ct, A = At.
- #define PVR_TXRENV_DECAL 2

  C = (Cs ∗ At) + (Cs ∗ (1-At)), A = As.
- #define PVR_TXRENV_MODULATEALPHA 3

  C = Cs ∗ Ct, A = As ∗ At.
- #define PVR_MIPMAP_DISABLE 0

  Disable mipmap processing.
- #define PVR_MIPMAP_ENABLE 1

  Enable mipmap processing.
- #define PVR_TXRFMT_NONE 0

  No texture.
- #define PVR_TXRFMT_VQ_DISABLE (0 << 30)

  Not VQ encoded.

- #define PVR_TXRFMT_VQ_ENABLE (1 << 30)

  VQ encoded.
- #define PVR_TXRFMT_ARGB1555 (0 << 27)

  16-bit ARGB1555
- #define PVR_TXRFMT_RGB565 (1 << 27)

  16-bit RGB565
- #define PVR_TXRFMT_ARGB4444 (2 << 27)

  16-bit ARGB4444
- #define PVR_TXRFMT_YUV422 (3 << 27)

  YUV422 format.
- #define PVR_TXRFMT_BUMP (4 << 27)

  Bumpmap format.
- #define PVR_TXRFMT_PAL4BPP (5 << 27)

  4BPP paletted format
- #define PVR_TXRFMT_PAL8BPP (6 << 27)

  8BPP paletted format
- #define PVR_TXRFMT_TWIDDLED (0 << 26)

  Texture is twiddled.
- #define PVR_TXRFMT_NONTWIDDLED (1 << 26)

  Texture is not twiddled.
- #define PVR_TXRFMT_NOSTRIDE (0 << 21)

  Texture is not strided.
- #define PVR_TXRFMT_STRIDE (1 << 21)

  Texture is strided.
- #define PVR_TXRFMT_8BPP_PAL(x) ((x) << 25)

  8BPP palette selector
- #define PVR_TXRFMT_4BPP_PAL(x) ((x) << 21)

  4BPP palette selector
- #define PVR_CLRFMT_ARGBPACKED 0

  32-bit integer ARGB
- #define PVR_CLRFMT_4FLOATS 1

  4 floating point values
- #define PVR_CLRFMT_INTENSITY 2

  Intensity color.
- #define PVR_CLRFMT_INTENSITY_PREV 3

  Use last intensity.
- #define PVR_UVFMT_32BIT 0

  32-bit floating point U/V
- #define PVR_UVFMT_16BIT 1

  16-bit floating point U/V
- #define PVR_MODIFIER_DISABLE 0

  Disable modifier effects.
- #define PVR_MODIFIER_ENABLE 1

  Enable modifier effects.
- #define PVR_MODIFIER_CHEAP_SHADOW 0
- #define PVR_MODIFIER_NORMAL 1
- #define PVR_MODIFIER_OTHER_POLY 0

Not the last polygon in the volume.

- #define PVR_MODIFIER_INCLUDE_LAST_POLY 1

  Last polygon, inclusion volume.
- #define PVR_MODIFIER_EXCLUDE_LAST_POLY 2

  Last polygon, exclusion volume.
- #define PVR_PACK_COLOR(a, r, g, b)

  Pack four floating point color values into a 32-bit integer form.
- #define PVR_CMD_POLYHDR 0x80840000

  PVR polygon header. Striplength set to 2.
- #define PVR_CMD_VERTEX 0xe0000000

  PVR vertex data.
- #define PVR_CMD_VERTEX_EOL 0xf0000000

  PVR vertex, end of strip.
- #define PVR_CMD_USERCLIP 0x20000000

  PVR user clipping area.
- #define PVR_CMD_MODIFIER 0x80000000

  PVR modifier volume.
- #define PVR_CMD_SPRITE 0xA0000000

  PVR sprite header.
- #define PVR_TA_CMD_TYPE_SHIFT 24
- #define PVR_TA_CMD_TYPE_MASK (7 << PVR_TA_CMD_TYPE_SHIFT)
- #define PVR_TA_CMD_USERCLIP_SHIFT 16
- #define PVR_TA_CMD_USERCLIP_MASK (3 << PVR_TA_CMD_USERCLIP_SHIFT)
- #define PVR_TA_CMD_CLRFMT_SHIFT 4
- #define PVR_TA_CMD_CLRFMT_MASK (7 << PVR_TA_CMD_CLRFMT_SHIFT)
- #define PVR_TA_CMD_SPECULAR_SHIFT 2
- #define PVR_TA_CMD_SPECULAR_MASK (1 << PVR_TA_CMD_SPECULAR_SHIFT)
- #define PVR_TA_CMD_SHADE_SHIFT 1
- #define PVR_TA_CMD_SHADE_MASK (1 << PVR_TA_CMD_SHADE_SHIFT)
- #define PVR_TA_CMD_UVFMT_SHIFT 0
- #define PVR_TA_CMD_UVFMT_MASK (1 << PVR_TA_CMD_UVFMT_SHIFT)
- #define PVR_TA_CMD_MODIFIER_SHIFT 7
- #define PVR_TA_CMD_MODIFIER_MASK (1 << PVR_TA_CMD_MODIFIER_SHIFT)
- #define PVR_TA_CMD_MODIFIERMODE_SHIFT 6
- #define PVR_TA_CMD_MODIFIERMODE_MASK (1 << PVR_TA_CMD_MODIFIERMODE_SHIFT)
- #define PVR_TA_PM1_DEPTHCMP_SHIFT 29
- #define PVR_TA_PM1_DEPTHCMP_MASK (7 << PVR_TA_PM1_DEPTHCMP_SHIFT)
- #define PVR_TA_PM1_CULLING_SHIFT 27
- #define PVR_TA_PM1_CULLING_MASK (3 << PVR_TA_PM1_CULLING_SHIFT)
- #define PVR_TA_PM1_DEPTHWRITE_SHIFT 26
- #define PVR_TA_PM1_DEPTHWRITE_MASK (1 << PVR_TA_PM1_DEPTHWRITE_SHIFT)
- #define PVR_TA_PM1_TXRENABLE_SHIFT 25
- #define PVR_TA_PM1_TXRENABLE_MASK (1 << PVR_TA_PM1_TXRENABLE_SHIFT)
- #define PVR_TA_PM1_MODIFIERINST_SHIFT 29
- #define PVR_TA_PM1_MODIFIERINST_MASK (3 << PVR_TA_PM1_MODIFIERINST_SHIFT)
- #define PVR_TA_PM2_SRCBLEND_SHIFT 29
- #define PVR_TA_PM2_SRCBLEND_MASK (7 << PVR_TA_PM2_SRCBLEND_SHIFT)
- #define PVR_TA_PM2_DSTBLEND_SHIFT 26
- #define PVR_TA_PM2_DSTBLEND_MASK (7 << PVR_TA_PM2_DSTBLEND_SHIFT)

- #define PVR_TA_PM2_SRCENABLE_SHIFT 25
- #define PVR_TA_PM2_SRCENABLE_MASK (1 << PVR_TA_PM2_SRCENABLE_SHIFT)
- #define PVR_TA_PM2_DSTENABLE_SHIFT 24
- #define PVR_TA_PM2_DSTENABLE_MASK (1 << PVR_TA_PM2_DSTENABLE_SHIFT)
- #define PVR_TA_PM2_FOG_SHIFT 22
- #define PVR_TA_PM2_FOG_MASK (3 << PVR_TA_PM2_FOG_SHIFT)
- #define PVR_TA_PM2_CLAMP_SHIFT 21
- #define PVR_TA_PM2_CLAMP_MASK (1 << PVR_TA_PM2_CLAMP_SHIFT)
- #define PVR_TA_PM2_ALPHA_SHIFT 20
- #define PVR_TA_PM2_ALPHA_MASK (1 << PVR_TA_PM2_ALPHA_SHIFT)
- #define PVR_TA_PM2_TXRALPHA_SHIFT 19
- #define PVR_TA_PM2_TXRALPHA_MASK (1 << PVR_TA_PM2_TXRALPHA_SHIFT)
- #define PVR_TA_PM2_UVFLIP_SHIFT 17
- #define PVR_TA_PM2_UVFLIP_MASK (3 << PVR_TA_PM2_UVFLIP_SHIFT)
- #define PVR_TA_PM2_UVCLAMP_SHIFT 15
- #define PVR_TA_PM2_UVCLAMP_MASK (3 << PVR_TA_PM2_UVCLAMP_SHIFT)
- #define PVR_TA_PM2_FILTER_SHIFT 12
- #define PVR_TA_PM2_FILTER_MASK (7 << PVR_TA_PM2_FILTER_SHIFT)
- #define PVR_TA_PM2_MIPBIAS_SHIFT 8
- #define PVR_TA_PM2_MIPBIAS_MASK (15 << PVR_TA_PM2_MIPBIAS_SHIFT)
- #define PVR_TA_PM2_TXRENV_SHIFT 6
- #define PVR_TA_PM2_TXRENV_MASK (3 << PVR_TA_PM2_TXRENV_SHIFT)
- #define PVR_TA_PM2_USIZE_SHIFT 3
- #define PVR_TA_PM2_USIZE_MASK (7 << PVR_TA_PM2_USIZE_SHIFT)
- #define PVR_TA_PM2_VSIZE_SHIFT 0
- #define PVR_TA_PM2_VSIZE_MASK (7 << PVR_TA_PM2_VSIZE_SHIFT)
- #define PVR_TA_PM3_MIPMAP_SHIFT 31
- #define PVR_TA_PM3_MIPMAP_MASK (1 << PVR_TA_PM3_MIPMAP_SHIFT)
- #define PVR_TA_PM3_TXRFMT_SHIFT 0
- #define PVR_TA_PM3_TXRFMT_MASK 0xffffffff
- #define PVR_GET(REG) (* ((vuint32*)(0xa05f8000 + (REG))))

  Retrieve a PVR register value.
- #define PVR_SET(REG, VALUE) PVR_GET(REG) = (VALUE)

  Set a PVR register value.
- #define PVR_ID 0x0000

  Chip ID.
- #define PVR_REVISION 0x0004

  Chip revision.
- #define PVR_RESET 0x0008

  Reset pins.
- #define PVR_ISP_START 0x0014

  Start the ISP/TSP.
- #define PVR_UNK_0018 0x0018

  ??
- #define PVR_ISP_VERTBUF_ADDR 0x0020

  Vertex buffer address for scene rendering.
- #define PVR_ISP_TILEMAT_ADDR 0x002c

  Tile matrix address for scene rendering.
- #define PVR_SPANSORT_CFG 0x0030

?? – write 0x101 for now

- #define PVR_BORDER_COLOR 0x0040

  Border Color in RGB888.
- #define PVR_FB_CFG_1 0x0044

  Framebuffer config 1.
- #define PVR_FB_CFG_2 0x0048

  Framebuffer config 2.
- #define PVR_RENDER_MODULO 0x004c

  Render modulo.
- #define PVR_FB_ADDR 0x0050

  Framebuffer start address.
- #define PVR_FB_IL_ADDR 0x0054

  Framebuffer odd-field start address for interlace.
- #define PVR_FB_SIZE 0x005c

  Framebuffer display size.
- #define PVR_RENDER_ADDR 0x0060

  Render output address.
- #define PVR_RENDER_ADDR_2 0x0064

  Output for strip-buffering.
- #define PVR_PCLIP_X 0x0068

  Horizontal clipping area.
- #define PVR_PCLIP_Y 0x006c

  Vertical clipping area.
- #define PVR_CHEAP_SHADOW 0x0074

  Cheap shadow control.
- #define PVR_OBJECT_CLIP 0x0078

  Distance for polygon culling.
- #define PVR_UNK_007C 0x007c

  ?? – write 0x0027df77 for now
- #define PVR_UNK_0080 0x0080

  ?? – write 7 for now
- #define PVR_TEXTURE_CLIP 0x0084

  Distance for texture clipping.
- #define PVR_BGPLANE_Z 0x0088

  Distance for background plane.
- #define PVR_BGPLANE_CFG 0x008c

  Background plane config.
- #define PVR_UNK_0098 0x0098

  ?? – write 0x00800408 for now
- #define PVR_UNK_00A0 0x00a0

  ?? – write 0x20 for now
- #define PVR_UNK_00A8 0x00a8

  ?? – write 0x15d1c951 for now
- #define PVR_FOG_TABLE_COLOR 0x00b0

  Table fog color.
- #define PVR_FOG_VERTEX_COLOR 0x00b4

  Vertex fog color.

- #define PVR_FOG_DENSITY 0x00b8

  Fog density coefficient.
- #define PVR_COLOR_CLAMP_MAX 0x00bc

  RGB Color clamp max.
- #define PVR_COLOR_CLAMP_MIN 0x00c0

  RGB Color clamp min.
- #define PVR_GUN_POS 0x00c4

  Light gun position.
- #define PVR_HPOS_IRQ 0x00c8

  Horizontal position IRQ.
- #define PVR_VPOS_IRQ 0x00cc

  Vertical position IRQ.
- #define PVR_IL_CFG 0x00d0

  Interlacing config.
- #define PVR_BORDER_X 0x00d4

  Window border X position.
- #define PVR_SCAN_CLK 0x00d8

  Clock and scanline values.
- #define PVR_BORDER_Y 0x00dc

  Window border Y position.
- #define PVR_TEXTURE_MODULO 0x00e4

  Output texture width modulo.
- #define PVR_VIDEO_CFG 0x00e8

  Misc video config.
- #define PVR_BITMAP_X 0x00ec

  Bitmap window X position.
- #define PVR_BITMAP_Y 0x00f0

  Bitmap window Y position.
- #define PVR_SCALER_CFG 0x00f4

  Smoothing scaler.
- #define PVR_PALETTE_CFG 0x0108

  Palette format.
- #define PVR_SYNC_STATUS 0x010c

  V/H blank status.
- #define PVR_UNK_0110 0x0110

  ?? – write 0x93f39 for now
- #define PVR_UNK_0114 0x0114

  ?? – write 0x200000 for now
- #define PVR_UNK_0118 0x0118

  ?? – write 0x8040 for now
- #define PVR_TA_OPB_START 0x0124

  Object Pointer Buffer start for TA usage.
- #define PVR_TA_VERTBUF_START 0x0128

  Vertex buffer start for TA usage.
- #define PVR_TA_OPB_END 0x012c

  OPB end for TA usage.
- #define PVR_TA_VERTBUF_END 0x0130

Vertex buffer end for TA usage.

- #define PVR_TA_OPB_POS 0x0134

  Top used memory location in OPB for TA usage.
- #define PVR_TA_VERTBUF_POS 0x0138

  Top used memory location in vertbuf for TA usage.
- #define PVR_TILEMAT_CFG 0x013c

  Tile matrix size config.
- #define PVR_OPB_CFG 0x0140

  Active lists / list size.
- #define PVR_TA_INIT 0x0144

  Initialize vertex reg. params.
- #define PVR_YUV_ADDR 0x0148

  YUV conversion destination.
- #define PVR_YUV_CFG 0x014c

  YUV configuration.
- #define PVR_YUV_STAT 0x0150

  The number of YUV macroblocks converted.
- #define PVR_UNK_0160 0x0160

  ??
- #define PVR_TA_OPB_INIT 0x0164

  Object pointer buffer position init.
- #define PVR_FOG_TABLE_BASE 0x0200

  Base of the fog table.
- #define PVR_PALETTE_TABLE_BASE 0x1000

  Base of the palette table.
- #define PVR_TA_INPUT 0x10000000

  TA command input.
- #define PVR_TA_YUV_CONV 0x10800000

  YUV converter.
- #define PVR_TA_TEX_MEM 0x11000000

  Texture memory.
- #define PVR_RAM_BASE 0xa5000000

  PVR RAM (raw)
- #define PVR_RAM_INT_BASE 0xa4000000

  PVR RAM (interleaved)
- #define PVR_RAM_SIZE (8∗1024∗1024)

  RAM size in bytes.
- #define PVR_RAM_TOP (PVR_RAM_BASE + PVR_RAM_SIZE)

  Top of raw PVR RAM.
- #define PVR_RAM_INT_TOP (PVR_RAM_INT_BASE + PVR_RAM_SIZE)

  Top of int PVR RAM.
- #define PVR_RESET_ALL 0xffffffff

  Reset the whole PVR.
- #define PVR_RESET_NONE 0x00000000

  Cancel reset state.
- #define PVR_RESET_TA 0x00000001

  Reset only the TA.

- #define PVR_RESET_ISPTSP 0x00000002

  Reset only the ISP/TSP.
- #define PVR_ISP_START_GO 0xffffffff

  Write to the PVR_ISP_START register to start rendering.
- #define PVR_TA_INIT_GO 0x80000000

  Write to the PVR_TA_INIT register to confirm settings.
- #define PVR_BINSIZE_0 0

  0-length (disables the list)
- #define PVR_BINSIZE_8 8

  8-word (32-byte) length
- #define PVR_BINSIZE_16 16

  16-word (64-byte) length
- #define PVR_BINSIZE_32 32

  32-word (128-byte) length
- #define PVR_PAL_ARGB1555 0

  16-bit ARGB1555 palette format
- #define PVR_PAL_RGB565 1

  16-bit RGB565 palette format
- #define PVR_PAL_ARGB4444 2

  16-bit ARGB4444 palette format
- #define PVR_PAL_ARGB8888 3

  32-bit ARGB8888 palette format
- #define pvr_dr_init(vtx_buf_ptr)

  Initialize a state variable for Direct Rendering.
- #define pvr_dr_target(vtx_buf_ptr)

  Obtain the target address for Direct Rendering.
- #define pvr_dr_commit(addr) __asm__ __volatile__("pref @%0" : : "r" (addr))

  Commit a primitive written into the Direct Rendering target address.
- #define PVR_TXRLOAD_4BPP 0x01

  4BPP format
- #define PVR_TXRLOAD_8BPP 0x02

  8BPP format
- #define PVR_TXRLOAD_16BPP 0x03

  16BPP format
- #define PVR_TXRLOAD_FMT_MASK 0x0f

  Bits used for basic formats.
- #define PVR_TXRLOAD_VQ_LOAD 0x10

  Do VQ encoding (not supported yet, if ever)
- #define PVR_TXRLOAD_INVERT_Y 0x20

  Invert the Y axis while loading.
- #define PVR_TXRLOAD_FMT_VQ 0x40

  Texture is already VQ encoded.
- #define PVR_TXRLOAD_FMT_TWIDDLED 0x80

  Texture is already twiddled.
- #define PVR_TXRLOAD_FMT_NOTWIDDLE 0x80

  Don't twiddle the texture while loading.
- #define PVR_TXRLOAD_DMA 0x8000

Use DMA to load the texture.

- #define PVR_TXRLOAD_NONBLOCK 0x4000

  Use non-blocking loads (only for DMA)
- #define PVR_TXRLOAD_SQ 0x2000

  Use store queues to load.
- #define PVR_DMA_VRAM64 0

  Transfer to VRAM in interleaved mode.
- #define PVR_DMA_VRAM32 1

  Transfer to VRAM in linear mode.
- #define PVR_DMA_TA 2

  Transfer to the tile accelerator.
- #define PVR_DMA_YUV 3

  Transfer to the YUV converter.

Typedefs

- typedef void * pvr_ptr_t

  PVR texture memory pointer.
- typedef uint32_t pvr_list_t

  PVR list specification.
- typedef uint32_t pvr_dr_state_t

  Direct Rendering state variable type.
- typedef void(* pvr_dma_callback_t) (void *data)

  PVR DMA interrupt callback type.

Functions

- static uint32_t PVR_PACK_16BIT_UV (float u, float v)

  Pack two floating point coordinates into one 32-bit value, truncating them to 16-bits each.
- int pvr_init (pvr_init_params_t *params)

  Initialize the PVR chip to ready status.
- int pvr_init_defaults (void)

  Simple PVR initialization.
- int pvr_shutdown (void)

  Shut down the PVR chip from ready status.
- void pvr_set_bg_color (float r, float g, float b)

  Set the background plane color.
- void pvr_set_shadow_scale (int enable, float scale_value)

  Set cheap shadow parameters.
- void pvr_set_zclip (float zc)

  Set Z clipping depth.
- int pvr_get_vbl_count (void)

  Retrieve the current VBlank count.
- int pvr_get_stats (pvr_stats_t *stat)

  Get the current statistics from the PVR.
- void pvr_set_pal_format (int fmt)

Set the palette format.

- static void pvr_set_pal_entry (uint32_t idx, uint32_t value)

  Set a palette value.
- void pvr_fog_table_color (float a, float r, float g, float b)

  Set the table fog color.
- void pvr_fog_vertex_color (float a, float r, float g, float b)

  Set the vertex fog color.
- void pvr_fog_far_depth (float d)

  Set the fog far depth.
- void pvr_fog_table_exp2 (float density)

  Initialize the fog table using an exp2 algorithm (like GL_EXP2).
- void pvr_fog_table_exp (float density)

  Initialize the fog table using an exp algorithm (like GL_EXP).
- void pvr_fog_table_linear (float start, float end)

  Initialize the fog table using a linear algorithm (like GL_LINEAR).
- void pvr_fog_table_custom (float tbl1[])

  Set a custom fog table from float values.
- pvr_ptr_t pvr_mem_malloc (size_t size)

  Allocate a chunk of memory from texture space.
- void pvr_mem_free (pvr_ptr_t chunk)

  Free a block of allocated memory in the PVR RAM pool.
- uint32_t pvr_mem_available (void)

  Return the number of bytes available still in the PVR RAM pool.
- void pvr_mem_reset (void)

  Reset the PVR RAM pool.
- void pvr_mem_print_list (void)

  Print the list of allocated blocks in the PVR RAM pool.
- void pvr_mem_stats (void)

  Print statistics about the PVR RAM pool.
- int pvr_vertex_dma_enabled (void)

  Is vertex DMA enabled?
- void * pvr_set_vertbuf (pvr_list_t list, void *buffer, int len)

  Setup a vertex buffer for one of the list types.
- void * pvr_vertbuf_tail (pvr_list_t list)

  Retrieve a pointer to the current output location in the DMA buffer for the requested list.
- void pvr_vertbuf_written (pvr_list_t list, uint32_t amt)

  Notify the PVR system that data have been written into the output buffer for the given list.
- void pvr_set_presort_mode (int presort)

  Set the translucent polygon sort mode for the next frame.
- void pvr_scene_begin (void)

  Begin collecting data for a frame of 3D output to the off-screen frame buffer.
- void pvr_scene_begin_txr (pvr_ptr_t txr, uint32_t *rx, uint32_t *ry)

  Begin collecting data for a frame of 3D output to the specified texture.
- int pvr_list_begin (pvr_list_t list)

  Begin collecting data for the given list type.
- int pvr_list_finish (void)

  End collecting data for the current list type.

- int pvr_prim (void *data, int size)

  Submit a primitive of the current list type.
- int pvr_list_prim (pvr_list_t list, void *data, int size)

  Submit a primitive of the given list type.
- int pvr_list_flush (pvr_list_t list)

  Flush the buffered data of the given list type to the TA.
- int pvr_scene_finish (void)

  Call this after you have finished submitting all data for a frame.
- int pvr_wait_ready (void)

  Block the caller until the PVR system is ready for another frame to be submitted.
- int pvr_check_ready (void)

  Check if the PVR system is ready for another frame to be submitted.
- void pvr_poly_compile (pvr_poly_hdr_t *dst, pvr_poly_cxt_t *src)

  Compile a polygon context into a polygon header.
- void pvr_poly_cxt_col (pvr_poly_cxt_t *dst, pvr_list_t list)

  Fill in a polygon context for non-textured polygons.
- void pvr_poly_cxt_txr (pvr_poly_cxt_t *dst, pvr_list_t list, int textureformat, int tw, int th, pvr_ptr_t textureaddr, int filtering)

  Fill in a polygon context for a textured polygon.
- void pvr_sprite_compile (pvr_sprite_hdr_t *dst, pvr_sprite_cxt_t *src)

  Compile a sprite context into a sprite header.
- void pvr_sprite_cxt_col (pvr_sprite_cxt_t *dst, pvr_list_t list)

  Fill in a sprite context for non-textured sprites.
- void pvr_sprite_cxt_txr (pvr_sprite_cxt_t *dst, pvr_list_t list, int textureformat, int tw, int th, pvr_ptr_t textureaddr, int filtering)

  Fill in a sprite context for a textured sprite.
- void pvr_mod_compile (pvr_mod_hdr_t *dst, pvr_list_t list, uint32_t mode, uint32_t cull)

  Create a modifier volume header.
- void pvr_poly_mod_compile (pvr_poly_mod_hdr_t *dst, pvr_poly_cxt_t *src)

  Compile a polygon context into a polygon header that is affected by modifier volumes.
- void pvr_poly_cxt_col_mod (pvr_poly_cxt_t *dst, pvr_list_t list)

  Fill in a polygon context for non-textured polygons affected by a modifier volume.
- void pvr_poly_cxt_txr_mod (pvr_poly_cxt_t *dst, pvr_list_t list, int textureformat, int tw, int th, pvr_ptr_t textureaddr, int filtering, int textureformat2, int tw2, int th2, pvr_ptr_t textureaddr2, int filtering2)

  Fill in a polygon context for a textured polygon affected by modifier volumes.
- void pvr_txr_load (void *src, pvr_ptr_t dst, uint32_t count)

  Load raw texture data from an SH-4 buffer into PVR RAM.
- void pvr_txr_load_ex (void *src, pvr_ptr_t dst, uint32_t w, uint32_t h, uint32_t flags)

  Load texture data from an SH-4 buffer into PVR RAM, twiddling it in the process.
- void pvr_txr_load_kimg (kos_img_t *img, pvr_ptr_t dst, uint32_t flags)

  Load a KOS Platform Independent Image (subject to constraint checking).
- int pvr_dma_transfer (void *src, uintptr_t dest, size_t count, int type, int block, pvr_dma_callback_t callback, void *cbdata)

  Perform a DMA transfer to the PVR.
- int pvr_txr_load_dma (void *src, pvr_ptr_t dest, size_t count, int block, pvr_dma_callback_t callback, void *cbdata)

  Load a texture using PVR DMA.
- int pvr_dma_load_ta (void *src, size_t count, int block, pvr_dma_callback_t callback, void *cbdata)

Load vertex data to the TA using PVR DMA.

- int pvr_dma_yuv_conv (void *src, size_t count, int block, pvr_dma_callback_t callback, void *cbdata)

  Load yuv data to the YUV converter using PVR DMA.
- int pvr_dma_ready (void)

  Is PVR DMA is inactive?
- void pvr_dma_init (void)

  Initialize PVR DMA.
- void pvr_dma_shutdown (void)

  Shut down PVR DMA.

9.226.1 Detailed Description

Low-level PVR (3D hardware) interface.

This file provides support for using the PVR 3D hardware in the Dreamcast. Note that this does not handle any sort of perspective transformations or anything of the like. This is just a very thin wrapper around the actual hardware support.

This file is used for pretty much everything related to the PVR, from memory management to actual primitive rendering.

Author

Megan Potter

Roger Cattermole

Paul Boese

Brian Paul

Lawrence Sebald

Benoit Miller

9.226.2 Macro Definition Documentation

pvr_dr_commit

```
#define pvr_dr_commit(
             addr ) __asm__ __volatile__("pref @%0" :  :  "r" (addr))
```

Commit a primitive written into the Direct Rendering target address.

Parameters

addr	The address returned by pvr_dr_target(), after you have written the primitive to it.

pvr_dr_init

```
#define pvr_dr_init(
```

```
                vtx_buf_ptr )
```

Value:

```
        do { \
        (vtx_buf_ptr) = 0; \
        QACR0 = ((((uint32)PVR_TA_INPUT) >> 26) << 2) & 0x1c; \
        QACR1 = ((((uint32)PVR_TA_INPUT) >> 26) << 2) & 0x1c; \
    } while(0)
```

Initialize a state variable for Direct Rendering.

Parameters

vtx_buf_ptr	A variable of type pvr_dr_state_t to init.

pvr_dr_target

```
#define pvr_dr_target(
                vtx_buf_ptr )
```

Value:

```
    ({ (vtx_buf_ptr) ^= 32; \
        (pvr_vertex_t *)(MEM_AREA_P4_BASE | (vtx_buf_ptr)); \
    })
```

Obtain the target address for Direct Rendering.

Parameters

vtx_buf_ptr	State variable for Direct Rendering. Should be of type pvr_dr_state_t, and must have been initialized previously in the scene with pvr_dr_init().

Returns

A write-only destination address where a primitive should be written to get ready to submit it to the TA in DR mode.

PVR_GET

```
#define PVR_GET(
                REG ) (* ( (vuint32*)( 0xa05f8000 + (REG) ) ) )
```

Retrieve a PVR register value.

Parameters

REG	The register to fetch

Returns

The value of that register (32-bits)

PVR_ISP_START_GO

```
#define PVR_ISP_START_GO 0xffffffff
```

Write to the PVR_ISP_START register to start rendering.

PVR_MODIFIER_CHEAP_SHADOW

```
#define PVR_MODIFIER_CHEAP_SHADOW 0
```

PVR_MODIFIER_NORMAL

```
#define PVR_MODIFIER_NORMAL 1
```

PVR_PACK_COLOR

```
#define PVR_PACK_COLOR(
              a,
              r,
              g,
              b )
```

Value:

```
( \
( ((uint8)( a * 255 ) ) « 24 ) | \
( ((uint8)( r * 255 ) ) « 16 ) | \
( ((uint8)( g * 255 ) ) « 8 ) | \
( ((uint8)( b * 255 ) ) « 0 ) )
```

Pack four floating point color values into a 32-bit integer form.

All of the color values should be between 0 and 1.

Parameters

a	Alpha value
r	Red value
g	Green value
b	Blue value

Returns

The packed color value

PVR_RAM_BASE

```
#define PVR_RAM_BASE 0xa5000000
```

PVR RAM (raw)

PVR_RAM_INT_BASE

```
#define PVR_RAM_INT_BASE 0xa4000000
```

PVR RAM (interleaved)

PVR_RAM_INT_TOP

```
#define PVR_RAM_INT_TOP (PVR_RAM_INT_BASE + PVR_RAM_SIZE)
```

Top of int PVR RAM.

PVR_RAM_SIZE

```
#define PVR_RAM_SIZE (8*1024*1024)
```

RAM size in bytes.

PVR_RAM_TOP

```
#define PVR_RAM_TOP (PVR_RAM_BASE + PVR_RAM_SIZE)
```

Top of raw PVR RAM.

PVR_SET

```
#define PVR_SET(
              REG,
              VALUE ) PVR_GET(REG) = (VALUE)
```

Set a PVR register value.

Parameters

REG	The register to set
VALUE	The value to set in the register (32-bits)

PVR_TA_INIT_GO

```
#define PVR_TA_INIT_GO 0x80000000
```

Write to the PVR_TA_INIT register to confirm settings.

PVR_TA_INPUT

```
#define PVR_TA_INPUT 0x10000000
```

TA command input.

PVR_TA_TEX_MEM

```
#define PVR_TA_TEX_MEM 0x11000000
```

Texture memory.

PVR_TA_YUV_CONV

```
#define PVR_TA_YUV_CONV 0x10800000
```

YUV converter.

9.226.3 Typedef Documentation

pvr_dma_callback_t

```
typedef void(* pvr_dma_callback_t) (void *data)
```

PVR DMA interrupt callback type.

Functions that act as callbacks when DMA completes should be of this type. These functions will be called inside an interrupt context, so don't try to use anything that might stall.

Parameters

data	User data passed in to the pvr_dma_transfer() function.

pvr_dr_state_t

```
typedef uint32_t pvr_dr_state_t
```

Direct Rendering state variable type.

pvr_list_t

```
typedef uint32_t pvr_list_t
```

PVR list specification.

Each primitive in the PVR is submitted to one of the hardware primitive lists. This type is an identifier for a list.

See also

PVR primitive list types

pvr_ptr_t

```
typedef void* pvr_ptr_t
```

PVR texture memory pointer.

Unlike the old "TA" system, PVR pointers in the new system are actually SH-4 compatible pointers and can be used directly in place of ta_txr_map().

Not that anyone probably even remembers the old TA system anymore...

9.226.4 Function Documentation

pvr_check_ready()

```
int pvr_check_ready (
            void  )
```

Check if the PVR system is ready for another frame to be submitted.

Return values

0	If the PVR is ready for a new scene. You must call pvr_wait_ready() afterwards, before starting a new scene.
-1	If the PVR is not ready for a new scene yet.

pvr_dma_init()

```
void pvr_dma_init (
             void  )
```

Initialize PVR DMA.

pvr_dma_load_ta()

```
int pvr_dma_load_ta (
             void * src,
             size_t count,
             int block,
             pvr_dma_callback_t callback,
             void * cbdata )
```

Load vertex data to the TA using PVR DMA.

This is essentially a convenience wrapper for pvr_dma_transfer(), so all notes that apply to it also apply here.

Parameters

src	Where to copy from. Must be 32-byte aligned.
count	The number of bytes to copy. Must be a multiple of 32.
block	Non-zero if you want the function to block until the DMA completes.
callback	A function to call upon completion of the DMA.
cbdata	Data to pass to the callback function.

Return values

0	On success.
-1	On failure. Sets errno as appropriate.

Error Conditions:

EINPROGRESS - DMA already in progress
EFAULT - dest is not 32-byte aligned
EIO - I/O error

pvr_dma_ready()

```
int pvr_dma_ready (
            void  )
```

Is PVR DMA is inactive?

Returns

Non-zero if there is no PVR DMA active, thus a DMA can begin or 0 if there is an active DMA.

pvr_dma_shutdown()

```
void pvr_dma_shutdown (
            void  )
```

Shut down PVR DMA.

pvr_dma_transfer()

```
int pvr_dma_transfer (
            void * src,
            uintptr_t dest,
            size_t count,
            int type,
            int block,
            pvr_dma_callback_t callback,
            void * cbdata )
```

Perform a DMA transfer to the PVR.

This function copies a block of data to the PVR or its memory via DMA. There are all kinds of constraints that must be fulfilled to actually do this, so make sure to read all the fine print with the parameter list.

If a callback is specified, it will be called in an interrupt context, so keep that in mind in writing the callback.

Parameters

src	Where to copy from. Must be 32-byte aligned.
dest	Where to copy to. Must be 32-byte aligned.
count	The number of bytes to copy. Must be a multiple of 32.
type	The type of DMA transfer to do (see list of modes).
block	Non-zero if you want the function to block until the DMA completes.
callback	A function to call upon completion of the DMA.
cbdata	Data to pass to the callback function.

Return values

0	On success.
-1	On failure. Sets errno as appropriate.

Error Conditions:

EINPROGRESS - DMA already in progress
EFAULT - dest is not 32-byte aligned
EIO - I/O error

See also

Transfer modes with PVR DMA

pvr_dma_yuv_conv()

```
int pvr_dma_yuv_conv (
            void * src,
            size_t count,
            int block,
            pvr_dma_callback_t callback,
            void * cbdata )
```

Load yuv data to the YUV converter using PVR DMA.

This is essentially a convenience wrapper for pvr_dma_transfer(), so all notes that apply to it also apply here.

Parameters

src	Where to copy from. Must be 32-byte aligned.
count	The number of bytes to copy. Must be a multiple of 32.
block	Non-zero if you want the function to block until the DMA completes.
callback	A function to call upon completion of the DMA.
cbdata	Data to pass to the callback function.

Return values

0	On success.
-1	On failure. Sets errno as appropriate.

Error Conditions:

EINPROGRESS - DMA already in progress
EFAULT - dest is not 32-byte aligned
EIO - I/O error

pvr_fog_far_depth()

```
void pvr_fog_far_depth (
            float d )
```

Set the fog far depth.

This function sets the PVR_FOG_DENSITY register appropriately for the specified value.

Parameters

d	The depth to set

pvr_fog_table_color()

```
void pvr_fog_table_color (
            float a,
            float r,
            float g,
            float b )
```

Set the table fog color.

This function sets the color of fog for table fog. 0-1 range for all colors.

Parameters

a	Alpha value of the fog
r	Red value of the fog
g	Green value of the fog
b	Blue value of the fog

pvr_fog_table_custom()

```
void pvr_fog_table_custom (
            float tbl1[] )
```

Set a custom fog table from float values.

This function allows you to specify whatever values you need to for your fog parameters. All values should be clamped between 0 and 1, and its your responsibility to set up the PVR_FOG_DENSITY register by calling pvr_fog_far_depth() with an appropriate value. The table passed in should have 129 entries, where the 0th entry is farthest from the eye and the last entry is nearest. Higher values = heavier fog.

Parameters

tbl1	The table of fog values to set

pvr_fog_table_exp()

```
void pvr_fog_table_exp (
             float density )
```

Initialize the fog table using an exp algorithm (like GL_EXP).

This function will automatically set the PVR_FOG_DENSITY register to 259.999999 as a part of its processing, then set up the fog table.

Parameters

density	Fog density value

pvr_fog_table_exp2()

```
void pvr_fog_table_exp2 (
             float density )
```

Initialize the fog table using an exp2 algorithm (like GL_EXP2).

This function will automatically set the PVR_FOG_DENSITY register to 259.999999 as a part of its processing, then set up the fog table.

Parameters

density	Fog density value

pvr_fog_table_linear()

```
void pvr_fog_table_linear (
             float start,
             float end )
```

Initialize the fog table using a linear algorithm (like GL_LINEAR).

This function will set the PVR_FOG_DENSITY register to the as appropriate for the end value, and initialize the fog table for perspectively correct linear fog.

Parameters

start	Fog start point
end	Fog end point

pvr_fog_vertex_color()

```
void pvr_fog_vertex_color (
            float a,
            float r,
            float g,
            float b )
```

Set the vertex fog color.

This function sets the fog color for vertex fog. 0-1 range for all colors. This function is currently not implemented, as vertex fog is not supported by KOS. Calling this function will cause an assertion failure.

Parameters

a	Alpha value of the fog
r	Red value of the fog
g	Green value of the fog
b	Blue value of the fog

pvr_get_stats()

```
int pvr_get_stats (
            pvr_stats_t * stat )
```

Get the current statistics from the PVR.

This function fills in the pvr_stats_t structure passed in with the current statistics of the system.

Parameters

stat	The statistics structure to fill in. Must not be NULL

Return values

0	On success
-1	If the PVR is not initialized

pvr_get_vbl_count()

```
int pvr_get_vbl_count (
            void  )
```

Retrieve the current VBlank count.

This function retrieves the number of VBlank interrupts that have occurred since the PVR was initialized.

Returns

The number of VBlanks since init

pvr_init()

```
int pvr_init (
            pvr_init_params_t * params )
```

Initialize the PVR chip to ready status.

This function enables the specified lists and uses the specified parameters. Note that bins and vertex buffers come from the texture memory pool, so only allocate what you actually need. Expects that a 2D mode was initialized already using the vid_* API.

Parameters

params	The set of parameters to initialize with

Return values

0	On success
-1	If the PVR has already been initialized or the video mode active is not suitable for 3D

pvr_init_defaults()

```
int pvr_init_defaults (
            void  )
```

Simple PVR initialization.

This simpler function initializes the PVR using 16/16 for the opaque and translucent lists' bin sizes, and 0's for everything else. It sets 512KB of vertex buffer. This is equivalent to the old ta_init_defaults() for now.

Return values

0	On success
-1	If the PVR has already been initialized or the video mode active is not suitable for 3D

pvr_list_begin()

```
int pvr_list_begin (
            pvr_list_t list )
```

Begin collecting data for the given list type.

Lists do not have to be submitted in any particular order, but all types of a list must be submitted at once (unless vertex DMA mode is enabled).

Note that there is no need to call this function in DMA mode unless you want to make use of pvr_prim() for compatibility. This function will automatically call pvr_list_finish() if a list is already opened before opening the new list.

Parameters

list	The list to open.

Return values

0	On success.
-1	If the specified list has already been closed.

pvr_list_finish()

```
int pvr_list_finish (
            void  )
```

End collecting data for the current list type.

Lists can never be opened again within a single frame once they have been closed. Thus submitting a primitive that belongs in a closed list is considered an error. Closing a list that is already closed is also an error.

Note that if you open a list but do not submit any primitives, a blank one will be submitted to satisfy the hardware. If vertex DMA mode is enabled, then this simply sets the current list pointer to no list, and none of the above restrictions apply.

Return values

0	On success.
-1	On error.

pvr_list_flush()

```
int pvr_list_flush (
            pvr_list_t list )
```

Flush the buffered data of the given list type to the TA.

This function is currently not implemented, and calling it will result in an assertion failure. It is intended to be used later in a "hybrid" mode where both direct and DMA TA submission is possible.

Parameters

list	The list to flush.

Return values

-1	On error (it is not possible to succeed).

pvr_list_prim()

```
int pvr_list_prim (
            pvr_list_t list,
            void * data,
            int size )
```

Submit a primitive of the given list type.

Data will be queued in a vertex buffer, thus one must be available for the list specified (will be asserted by the code).

Parameters

list	The list to submit to.
data	The primitive to submit.
size	The size of the primitive in bytes. This must be a multiple of 32.

Return values

0	On success.
-1	On error.

pvr_mem_available()

```
uint32_t pvr_mem_available (
            void  )
```

Return the number of bytes available still in the PVR RAM pool.

Returns

The number of bytes available

pvr_mem_free()

```
void pvr_mem_free (
            pvr_ptr_t chunk )
```

Free a block of allocated memory in the PVR RAM pool.

This function frees memory previously allocated with pvr_mem_malloc().

Parameters

chunk	The location of the start of the block to free

pvr_mem_malloc()

```
pvr_ptr_t pvr_mem_malloc (
            size_t size )
```

Allocate a chunk of memory from texture space.

This function acts as the memory allocator for the PVR texture RAM pool. It acts exactly as one would expect a malloc() function to act, returning a normal pointer that can be directly written to if one desires to do so. All allocations will be aligned to a 32-byte boundary.

Parameters

size	The amount of memory to allocate

Returns

A pointer to the memory on success, NULL on error

pvr_mem_print_list()

```
void pvr_mem_print_list (
            void  )
```

Print the list of allocated blocks in the PVR RAM pool.

This function only works if you've enabled KM_DBG in pvr_mem.c.

pvr_mem_reset()

```
void pvr_mem_reset (
            void  )
```

Reset the PVR RAM pool.

This will essentially free any blocks allocated within the pool. There's generally not many good reasons for doing this.

pvr_mem_stats()

```
void pvr_mem_stats (
            void  )
```

Print statistics about the PVR RAM pool.

This prints out statistics like what malloc_stats() provides. Also, if KM_DBG is enabled in pvr_mem.c, it prints the list of allocated blocks.

pvr_mod_compile()

```
void pvr_mod_compile (
            pvr_mod_hdr_t * dst,
            pvr_list_t list,
            uint32_t mode,
            uint32_t cull )
```

Create a modifier volume header.

This function fills in a modifier volume header with the parameters specified. Note that unlike for polygons and sprites, there is no context step for modifiers.

Parameters

dst	Where to store the modifier header.
list	The primitive list to be used.
mode	The mode for this modifier.
cull	The culling mode to use.

See also

Modifier volume mode parameters

PVR culling modes

PVR_PACK_16BIT_UV()

```
static uint32_t PVR_PACK_16BIT_UV (
            float u,
            float v )  [inline], [static]
```

Pack two floating point coordinates into one 32-bit value, truncating them to 16-bits each.

Parameters

u	First coordinate to pack
v	Second coordinate to pack

Returns

The packed coordinates

pvr_poly_compile()

```
void pvr_poly_compile (
            pvr_poly_hdr_t * dst,
            pvr_poly_cxt_t * src )
```

Compile a polygon context into a polygon header.

This function compiles a pvr_poly_cxt_t into the form needed by the hardware for rendering. This is for use with normal polygon headers.

Parameters

dst	Where to store the compiled header.
src	The context to compile.

pvr_poly_cxt_col()

```
void pvr_poly_cxt_col (
            pvr_poly_cxt_t * dst,
            pvr_list_t list )
```

Fill in a polygon context for non-textured polygons.

This function fills in a pvr_poly_cxt_t with default parameters appropriate for rendering a non-textured polygon in the given list.

Parameters

dst	Where to store the polygon context.
list	The primitive list to be used.

pvr_poly_cxt_col_mod()

```
void pvr_poly_cxt_col_mod (
            pvr_poly_cxt_t * dst,
            pvr_list_t list )
```

Fill in a polygon context for non-textured polygons affected by a modifier volume.

This function fills in a pvr_poly_cxt_t with default parameters appropriate for rendering a non-textured polygon in the given list that will be affected by modifier volumes.

Parameters

dst	Where to store the polygon context.
list	The primitive list to be used.

pvr_poly_cxt_txr()

```
void pvr_poly_cxt_txr (
            pvr_poly_cxt_t * dst,
            pvr_list_t list,
            int textureformat,
            int tw,
            int th,
            pvr_ptr_t textureaddr,
            int filtering )
```

Fill in a polygon context for a textured polygon.

This function fills in a pvr_poly_cxt_t with default parameters appropriate for rendering a textured polygon in the given list.

Parameters

dst	Where to store the polygon context.
list	The primitive list to be used.
textureformat	The format of the texture used.
tw	The width of the texture, in pixels.
th	The height of the texture, in pixels.
textureaddr	A pointer to the texture.
filtering	The type of filtering to use.

See also

PVR texture formats

PVR texture sampling modes

pvr_poly_cxt_txr_mod()

```
void pvr_poly_cxt_txr_mod (
            pvr_poly_cxt_t * dst,
            pvr_list_t list,
            int textureformat,
            int tw,
            int th,
            pvr_ptr_t textureaddr,
```

```
            int filtering,
            int textureformat2,
            int tw2,
            int th2,
            pvr_ptr_t textureaddr2,
            int filtering2 )
```

Fill in a polygon context for a textured polygon affected by modifier volumes.

This function fills in a pvr_poly_cxt_t with default parameters appropriate for rendering a textured polygon in the given list and being affected by modifier volumes.

Parameters

dst	Where to store the polygon context.
list	The primitive list to be used.
textureformat	The format of the texture used (outside).
tw	The width of the texture, in pixels (outside).
th	The height of the texture, in pixels (outside).
textureaddr	A pointer to the texture (outside).
filtering	The type of filtering to use (outside).
textureformat2	The format of the texture used (inside).
tw2	The width of the texture, in pixels (inside).
th2	The height of the texture, in pixels (inside).
textureaddr2	A pointer to the texture (inside).
filtering2	The type of filtering to use (inside).

See also

PVR texture formats

PVR texture sampling modes

pvr_poly_mod_compile()

```
void pvr_poly_mod_compile (
            pvr_poly_mod_hdr_t * dst,
            pvr_poly_cxt_t * src )
```

Compile a polygon context into a polygon header that is affected by modifier volumes.

This function works pretty similarly to pvr_poly_compile(), but compiles into the header type that is affected by a modifier volume. The context should have been created with either pvr_poly_cxt_col_mod() or pvr_poly_cxt_txr_mod().

Parameters

dst	Where to store the compiled header.
src	The context to compile.

pvr_prim()

```
int pvr_prim (
            void * data,
            int size )
```

Submit a primitive of the current list type.

Note that any values submitted in this fashion will go directly to the hardware without any sort of buffering, and submitting a primitive of the wrong type will quite likely ruin your scene. Note that this also will not work if you haven't begun any list types (i.e., all data is queued). If DMA is enabled, the primitive will be appended to the end of the currently selected list's buffer.

Parameters

data	The primitive to submit.
size	The length of the primitive, in bytes. Must be a multiple of 32.

Return values

0	On success.
-1	On error.

pvr_scene_begin()

```
void pvr_scene_begin (
            void  )
```

Begin collecting data for a frame of 3D output to the off-screen frame buffer.

You must call this function (or pvr_scene_begin_txr()) for ever frame of output.

pvr_scene_begin_txr()

```
void pvr_scene_begin_txr (
            pvr_ptr_t txr,
            uint32_t * rx,
            uint32_t * ry )
```

Begin collecting data for a frame of 3D output to the specified texture.

This function currently only supports outputting at the same size as the actual screen. Thus, make sure rx and ry are at least large enough for that. For a 640x480 output, rx will generally be 1024 on input and ry 512, as these are the smallest values that are powers of two and will hold the full screen sized output.

Parameters

txr	The texture to render to.
rx	Width of the texture buffer (in pixels).
ry	Height of the texture buffer (in pixels).

pvr_scene_finish()

```
int pvr_scene_finish (
            void  )
```

Call this after you have finished submitting all data for a frame.

Once this has been called, you can not submit any more data until one of the pvr_scene_begin() or pvr_scene_begin_txr() functions is called again.

Return values

0	On success.
-1	On error (no scene started).

pvr_set_bg_color()

```
void pvr_set_bg_color (
            float r,
            float g,
            float b )
```

Set the background plane color.

This function sets the color of the area of the screen not covered by any other polygons.

Parameters

r	Red component of the color to set
g	Green component of the color to set
b	Blue component of the color to set

pvr_set_pal_entry()

```
static void pvr_set_pal_entry (
            uint32_t idx,
            uint32_t value )  [inline], [static]
```

Set a palette value.

Note that while the color format is variable, each entry is still 32-bits in length regardless (and you only get a total of 1024 of them). If using one of the 16-bit palette formats, only the low-order 16-bits of the entry are valid, and the high bits should be filled in with 0.

Parameters

idx	The index to set to (0-1023)
value	The color value to set in that palette entry

References PVR_PALETTE_TABLE_BASE, and PVR_SET.

pvr_set_pal_format()

```
void pvr_set_pal_format (
             int fmt )
```

Set the palette format.

This function sets the currently active palette format on the PVR. Each entry in the palette table is 32-bits in length, regardless of what color format is in use.

Be sure to use care when using the PVR_PAL_ARGB8888 format. Rendering speed is greatly affected (cut about in half) if you use any filtering with paletted textures with ARGB8888 entries in the palette.

Parameters

fmt	The format to use

See also

PVR palette formats

pvr_set_presort_mode()

```
void pvr_set_presort_mode (
             int presort )
```

Set the translucent polygon sort mode for the next frame.

This function sets the translucent polygon sort mode for the next frame of output, potentially switching between autosort and presort mode.

For most programs, you'll probably want to set this at initialization time (with the autosort_disabled field in the pvr_init_params_t structure) and not mess with it per-frame. It is recommended that if you do use this function to change the mode that you should set it each frame to ensure that the mode is set properly.

Parameters

presort	Set to 1 to set the presort mode for translucent polygons, set to 0 to use autosort mode.

pvr_set_shadow_scale()

```
void pvr_set_shadow_scale (
            int enable,
            float scale_value )
```

Set cheap shadow parameters.

This function sets up the PVR cheap shadow parameters for use. You can only specify one scale value per frame, so the effect that you can get from this is somewhat limited, but if you want simple shadows, this is the easiest way to do it.

Polygons affected by a shadow modifier volume will effectively multiply their final color by the scale value set here when shadows are enabled and the polygon is inside the modifier (or outside for exclusion volumes).

Parameters

enable	Set to non-zero to enable cheap shadow mode.
scale_value	Floating point value (between 0 and 1) representing how colors of polygons affected by and inside the volume will be modified by the shadow volume.

pvr_set_vertbuf()

```
void * pvr_set_vertbuf (
            pvr_list_t list,
            void * buffer,
            int len )
```

Setup a vertex buffer for one of the list types.

If the specified list type already has a vertex buffer, it will be replaced by the new one. Note that each buffer should actually be twice as long as what you will need to hold two frames worth of data).

You should generally not try to do this at any time besides before a frame is begun, or Bad Things May Happen.

Parameters

list	The primitive list to set the buffer for.
buffer	The location of the buffer in main RAM. This must be aligned to a 32-byte boundary.
len	The length of the buffer. This must be a multiple of 64, and must be at least 128 (even if you're not using the list).

Returns

The old buffer location (if any)

pvr_set_zclip()

```
void pvr_set_zclip (
             float zc )
```

Set Z clipping depth.

This function sets the Z clipping depth. The default value for this is 0.0001.

Parameters

zc	The new value to set the z clip parameter to.

pvr_shutdown()

```
int pvr_shutdown (
             void  )
```

Shut down the PVR chip from ready status.

This essentially leaves the video system in 2D mode as it was before the init.

Return values

0	On success
-1	If the PVR has not been initialized

pvr_sprite_compile()

```
void pvr_sprite_compile (
             pvr_sprite_hdr_t * dst,
             pvr_sprite_cxt_t * src )
```

Compile a sprite context into a sprite header.

This function compiles a pvr_sprite_cxt_t into the form needed by the hardware for rendering. This is for use with sprite headers.

Parameters

dst	Where to store the compiled header.
src	The context to compile.

pvr_sprite_cxt_col()

```
void pvr_sprite_cxt_col (
            pvr_sprite_cxt_t * dst,
            pvr_list_t list )
```

Fill in a sprite context for non-textured sprites.

This function fills in a pvr_sprite_cxt_t with default parameters appropriate for rendering a non-textured sprite in the given list.

Parameters

dst	Where to store the sprite context.
list	The primitive list to be used.

pvr_sprite_cxt_txr()

```
void pvr_sprite_cxt_txr (
            pvr_sprite_cxt_t * dst,
            pvr_list_t list,
            int textureformat,
            int tw,
            int th,
            pvr_ptr_t textureaddr,
            int filtering )
```

Fill in a sprite context for a textured sprite.

This function fills in a pvr_sprite_cxt_t with default parameters appropriate for rendering a textured sprite in the given list.

Parameters

dst	Where to store the sprite context.
list	The primitive list to be used.
textureformat	The format of the texture used.
tw	The width of the texture, in pixels.
th	The height of the texture, in pixels.
textureaddr	A pointer to the texture.
filtering	The type of filtering to use.

See also

PVR texture formats

PVR texture sampling modes

pvr_txr_load()

```
void pvr_txr_load (
            void * src,
            pvr_ptr_t dst,
            uint32_t count )
```

Load raw texture data from an SH-4 buffer into PVR RAM.

This essentially just acts as a memcpy() from main RAM to PVR RAM, using the store queues.

Parameters

src	The location in main RAM holding the texture.
dst	The location in PVR RAM to copy to.
count	The size of the texture in bytes (must be a multiple of 32).

pvr_txr_load_dma()

```
int pvr_txr_load_dma (
            void * src,
            pvr_ptr_t dest,
            size_t count,
            int block,
            pvr_dma_callback_t callback,
            void * cbdata )
```

Load a texture using PVR DMA.

This is essentially a convenience wrapper for pvr_dma_transfer(), so all notes that apply to it also apply here.

Parameters

src	Where to copy from. Must be 32-byte aligned.
dest	Where to copy to. Must be 32-byte aligned.
count	The number of bytes to copy. Must be a multiple of 32.
block	Non-zero if you want the function to block until the DMA completes.
callback	A function to call upon completion of the DMA.
cbdata	Data to pass to the callback function.

Return values

0	On success.
-1	On failure. Sets errno as appropriate.

Error Conditions:

EINPROGRESS - DMA already in progress
EFAULT - dest is not 32-byte aligned
EIO - I/O error

pvr_txr_load_ex()

```
void pvr_txr_load_ex (
            void * src,
            pvr_ptr_t dst,
            uint32_t w,
            uint32_t h,
            uint32_t flags )
```

Load texture data from an SH-4 buffer into PVR RAM, twiddling it in the process.

This function loads a texture to the PVR's RAM with the specified set of flags. It will currently always twiddle the data, whether you ask it to or not, and many of the parameters are just plain not supported at all... Pretty much the only supported flag, other than the format ones is the PVR_TXRLOAD_INVERT_Y one.

This will be slower than using pvr_txr_load() in pretty much all cases, so unless you need to twiddle your texture, just use that instead.

Parameters

src	The location to copy from.
dst	The location to copy to.
w	The width of the texture, in pixels.
h	The height of the texture, in pixels.
flags	Some set of flags, ORed together.

See also

Texture loading constants

pvr_txr_load_kimg()

```
void pvr_txr_load_kimg (
            kos_img_t * img,
            pvr_ptr_t dst,
            uint32_t flags )
```

Load a KOS Platform Independent Image (subject to constraint checking).

This function loads a KOS Platform Independent image to the PVR's RAM with the specified set of flags. This function, unlike pvr_txr_load_ex() supports everything in the flags available, other than what's explicitly marked as not supported.

Parameters

img	The image to load.
dst	The location to copy to.
flags	Some set of flags, ORed together.

See also

Texture loading constants

Note

Unless you explicitly tell this function to not twiddle the texture (by ORing PVR_TXRLOAD_FMT_NOTWIDDLE or it's equivalent PVR_TXRLOAD_FMT_TWIDDLED with flags), this function will twiddle the texture while loading. Keep that in mind when setting the texture format in polygon headers later.

You cannot specify both PVR_TXRLOAD_FMT_NOTWIDDLE (or equivalently PVR_TXRLOAD_FMT_TWIDDLED) and PVR_TXRLOAD_INVERT_Y in the flags.

DMA and Store Queue based loading is not available from this function if it twiddles the texture while loading.

pvr_vertbuf_tail()

```
void * pvr_vertbuf_tail (
            pvr_list_t list )
```

Retrieve a pointer to the current output location in the DMA buffer for the requested list.

Vertex DMA must globally be enabled for this to work. Data may be added to this buffer by the user program directly; however, make sure to call pvr_vertbuf_written() to notify the system of any such changes.

Parameters

list	The primitive list to get the buffer for.

Returns

The tail of that list's buffer.

pvr_vertbuf_written()

```
void pvr_vertbuf_written (
            pvr_list_t list,
            uint32_t amt )
```

Notify the PVR system that data have been written into the output buffer for the given list.

This should always be done after writing data directly to these buffers or it will get overwritten by other data.

Parameters

list	The primitive list that was modified.
amt	Number of bytes written. Must be a multiple of 32.

pvr_vertex_dma_enabled()

```
int pvr_vertex_dma_enabled (
            void  )
```

Is vertex DMA enabled?

Returns

Non-zero if vertex DMA was enabled at init time

pvr_wait_ready()

```
int pvr_wait_ready (
            void  )
```

Block the caller until the PVR system is ready for another frame to be submitted.

The PVR system allocates enough space for two frames: one in data collection mode, and another in rendering mode. If a frame is currently rendering, and another frame has already been closed, then the caller cannot do anything else until the rendering frame completes. Note also that the new frame cannot be activated except during a vertical blanking period, so this essentially waits until a rendered frame is complete and a vertical blank happens.

Return values

0	On success. A new scene can be started now.
-1	On error. Something is probably very wrong...

9.227 pvr.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/pvr.h
00004    Copyright (C) 2002 Megan Potter
00005    Copyright (C) 2014 Lawrence Sebald
00006
00007    Low-level PVR 3D interface for the DC
00008    Note: this API does _not_ handle any sort of transformations
00009    (including perspective!) so for that, you should look to KGL.
00010 */
00011
00012 /** \file   dc/pvr.h
```

```
00013     \brief  Low-level PVR (3D hardware) interface.
00014
00015     This file provides support for using the PVR 3D hardware in the Dreamcast.
00016     Note that this does not handle any sort of perspective transformations or
00017     anything of the like. This is just a very thin wrapper around the actual
00018     hardware support.
00019
00020     This file is used for pretty much everything related to the PVR, from memory
00021     management to actual primitive rendering.
00022
00023     \author Megan Potter
00024     \author Roger Cattermole
00025     \author Paul Boese
00026     \author Brian Paul
00027     \author Lawrence Sebald
00028     \author Benoit Miller
00029 */
00030
00031 #ifndef __DC_PVR_H
00032 #define __DC_PVR_H
00033
00034 #include <sys/cdefs.h>
00035 __BEGIN_DECLS
00036
00037 #include <arch/memory.h>
00038 #include <arch/types.h>
00039 #include <dc/sq.h>
00040 #include <kos/img.h>
00041
00042 /* Data types ************************************************/
00043
00044 /** \brief  PVR texture memory pointer.
00045
00046     Unlike the old "TA" system, PVR pointers in the new system are actually SH-4
00047     compatible pointers and can be used directly in place of ta_txr_map().
00048
00049     Not that anyone probably even remembers the old TA system anymore... */
00050 typedef void *pvr_ptr_t;
00051
00052 /** \brief  PVR list specification.
00053
00054     Each primitive in the PVR is submitted to one of the hardware primitive
00055     lists. This type is an identifier for a list.
00056
00057     \see    pvr_lists
00058 */
00059 typedef uint32_t pvr_list_t;
00060
00061 /** \brief  PVR polygon context.
00062
00063     You should use this more human readable format for specifying your polygon
00064     contexts, and then compile them into polygon headers when you are ready to
00065     start using them.
00066
00067     This has embedded structures in it for two reasons; the first reason is to
00068     make it easier for me to add new stuff later without breaking existing code.
00069     The second reason is to make it more readable and usable.
00070
00071     Unfortunately, it seems that Doxygen chokes up a little bit on this
00072     structure, and others like it. The documentation should still be mostly
00073     understandable though...
00074
00075     \headerfile dc/pvr.h
00076 */
00077 typedef struct {
00078     int     list_type;          /**< \brief Primitive list
00079                                      \see   pvr_lists */
00080     struct {
00081         int     alpha;          /**< \brief Enable or disable alpha outside modifier
00082                                      \see   pvr_alpha_switch */
00083         int     shading;        /**< \brief Shading type
00084                                      \see   pvr_shading_types */
00085         int     fog_type;       /**< \brief Fog type outside modifier
00086                                      \see   pvr_fog_types */
00087         int     culling;        /**< \brief Culling mode
00088                                      \see   pvr_cull_modes */
00089         int     color_clamp;    /**< \brief Color clamp enable/disable outside modifier
00090                                      \see   pvr_colclamp_switch */
00091         int     clip_mode;      /**< \brief Clipping mode
00092                                      \see   pvr_clip_modes */
00093         int     modifier_mode;  /**< \brief Modifier mode */
```

```
00094         int     specular;       /**< \brief Offset color enable/disable outside modifier
00095                                       \see   pvr_offset_switch */
00096         int     alpha2;         /**< \brief Enable/disable alpha inside modifier
00097                                       \see   pvr_alpha_switch */
00098         int     fog_type2;      /**< \brief Fog type inside modifier
00099                                       \see   pvr_fog_types */
00100         int     color_clamp2;   /**< \brief Color clamp enable/disable inside modifier
00101                                       \see   pvr_colclamp_switch */
00102     } gen;                      /**< \brief General parameters */
00103     struct {
00104         int     src;            /**< \brief Source blending mode outside modifier
00105                                       \see   pvr_blend_modes */
00106         int     dst;            /**< \brief Dest blending mode outside modifier
00107                                       \see   pvr_blend_modes */
00108         int     src_enable;     /**< \brief Source blending enable outside modifier
00109                                       \see   pvr_blend_switch */
00110         int     dst_enable;     /**< \brief Dest blending enable outside modifier
00111                                       \see   pvr_blend_switch */
00112         int     src2;           /**< \brief Source blending mode inside modifier
00113                                       \see   pvr_blend_modes */
00114         int     dst2;           /**< \brief Dest blending mode inside modifier
00115                                       \see   pvr_blend_modes */
00116         int     src_enable2;    /**< \brief Source blending mode inside modifier
00117                                       \see   pvr_blend_switch */
00118         int     dst_enable2;    /**< \brief Dest blending mode inside modifier
00119                                       \see   pvr_blend_switch */
00120     } blend;                    /**< \brief Blending parameters */
00121     struct {
00122         int     color;          /**< \brief Color format in vertex
00123                                       \see   pvr_color_fmts */
00124         int     uv;             /**< \brief U/V data format in vertex
00125                                       \see   pvr_uv_fmts */
00126         int     modifier;       /**< \brief Enable or disable modifier effect
00127                                       \see   pvr_mod_switch */
00128     } fmt;                      /**< \brief Format control */
00129     struct {
00130         int     comparison;     /**< \brief Depth comparison mode
00131                                       \see pvr_depth_modes */
00132         int     write;          /**< \brief Enable or disable depth writes
00133                                       \see pvr_depth_switch */
00134     } depth;                    /**< \brief Depth comparison/write modes */
00135     struct {
00136         int     enable;         /**< \brief Enable/disable texturing
00137                                       \see   pvr_txr_switch */
00138         int     filter;         /**< \brief Filtering mode
00139                                       \see   pvr_filter_modes */
00140         int     mipmap;         /**< \brief Enable/disable mipmaps
00141                                       \see   pvr_mip_switch */
00142         int     mipmap_bias;    /**< \brief Mipmap bias
00143                                       \see   pvr_mip_bias */
00144         int     uv_flip;        /**< \brief Enable/disable U/V flipping
00145                                       \see   pvr_uv_flip */
00146         int     uv_clamp;       /**< \brief Enable/disable U/V clamping
00147                                       \see   pvr_uv_clamp */
00148         int     alpha;          /**< \brief Enable/disable texture alpha
00149                                       \see   pvr_txralpha_switch */
00150         int     env;            /**< \brief Texture color contribution
00151                                       \see   pvr_txrenv_modes */
00152         int     width;          /**< \brief Texture width (requires a power of 2) */
00153         int     height;         /**< \brief Texture height (requires a power of 2) */
00154         int     format;         /**< \brief Texture format
00155                                       \see   pvr_txr_fmts */
00156         pvr_ptr_t base;         /**< \brief Texture pointer */
00157     } txr;                      /**< \brief Texturing params outside modifier */
00158     struct {
00159         int     enable;         /**< \brief Enable/disable texturing
00160                                       \see   pvr_txr_switch */
00161         int     filter;         /**< \brief Filtering mode
00162                                       \see   pvr_filter_modes */
00163         int     mipmap;         /**< \brief Enable/disable mipmaps
00164                                       \see   pvr_mip_switch */
00165         int     mipmap_bias;    /**< \brief Mipmap bias
00166                                       \see   pvr_mip_bias */
00167         int     uv_flip;        /**< \brief Enable/disable U/V flipping
00168                                       \see   pvr_uv_flip */
00169         int     uv_clamp;       /**< \brief Enable/disable U/V clamping
00170                                       \see   pvr_uv_clamp */
00171         int     alpha;          /**< \brief Enable/disable texture alpha
00172                                       \see   pvr_txralpha_switch */
00173         int     env;            /**< \brief Texture color contribution
00174                                       \see   pvr_txrenv_modes */
```

```
00175         int     width;          /**< \brief Texture width (requires a power of 2) */
00176         int     height;         /**< \brief Texture height (requires a power of 2) */
00177         int     format;         /**< \brief Texture format
00178                                      \see   pvr_txr_fmts */
00179         pvr_ptr_t base;         /**< \brief Texture pointer */
00180     } txr2;                     /**< \brief Texturing params inside modifier */
00181 } pvr_poly_cxt_t;
00182
00183 /** \brief  PVR sprite context.
00184
00185     You should use this more human readable format for specifying your sprite
00186     contexts, and then compile them into sprite headers when you are ready to
00187     start using them.
00188
00189     Unfortunately, it seems that Doxygen chokes up a little bit on this
00190     structure, and others like it. The documentation should still be mostly
00191     understandable though...
00192
00193     \headerfile dc/pvr.h
00194 */
00195 typedef struct {
00196     int     list_type;          /**< \brief Primitive list
00197                                      \see   pvr_lists */
00198     struct {
00199         int     alpha;          /**< \brief Enable or disable alpha
00200                                      \see   pvr_alpha_switch */
00201         int     fog_type;       /**< \brief Fog type
00202                                      \see   pvr_fog_types */
00203         int     culling;        /**< \brief Culling mode
00204                                      \see   pvr_cull_modes */
00205         int     color_clamp;    /**< \brief Color clamp enable/disable
00206                                      \see   pvr_colclamp_switch */
00207         int     clip_mode;      /**< \brief Clipping mode
00208                                      \see   pvr_clip_modes */
00209         int     specular;       /**< \brief Offset color enable/disable
00210                                      \see   pvr_offset_switch */
00211     } gen;                      /**< \brief General parameters */
00212     struct {
00213         int     src;            /**< \brief Source blending mode
00214                                      \see   pvr_blend_modes */
00215         int     dst;            /**< \brief Dest blending mode
00216                                      \see   pvr_blend_modes */
00217         int     src_enable;     /**< \brief Source blending enable
00218                                      \see   pvr_blend_switch */
00219         int     dst_enable;     /**< \brief Dest blending enable
00220                                      \see   pvr_blend_switch */
00221     } blend;
00222     struct {
00223         int     comparison;     /**< \brief Depth comparison mode
00224                                      \see pvr_depth_modes */
00225         int     write;          /**< \brief Enable or disable depth writes
00226                                      \see pvr_depth_switch */
00227     } depth;                    /**< \brief Depth comparison/write modes */
00228     struct {
00229         int     enable;         /**< \brief Enable/disable texturing
00230                                      \see   pvr_txr_switch */
00231         int     filter;         /**< \brief Filtering mode
00232                                      \see   pvr_filter_modes */
00233         int     mipmap;         /**< \brief Enable/disable mipmaps
00234                                      \see   pvr_mip_switch */
00235         int     mipmap_bias;    /**< \brief Mipmap bias
00236                                      \see   pvr_mip_bias */
00237         int     uv_flip;        /**< \brief Enable/disable U/V flipping
00238                                      \see   pvr_uv_flip */
00239         int     uv_clamp;       /**< \brief Enable/disable U/V clamping
00240                                      \see   pvr_uv_clamp */
00241         int     alpha;          /**< \brief Enable/disable texture alpha
00242                                      \see   pvr_txralpha_switch */
00243         int     env;            /**< \brief Texture color contribution
00244                                      \see   pvr_txrenv_modes */
00245         int     width;          /**< \brief Texture width (requires a power of 2) */
00246         int     height;         /**< \brief Texture height (requires a power of 2) */
00247         int     format;         /**< \brief Texture format
00248                                      \see   pvr_txr_fmts */
00249         pvr_ptr_t base;         /**< \brief Texture pointer */
00250     } txr;                      /**< \brief Texturing params */
00251 } pvr_sprite_cxt_t;
00252
00253 /* Constants for the above structure; thanks to Benoit Miller for these */
00254
00255 /** \defgroup pvr_lists PVR primitive list types
```

```
00256
00257     Each primitive submitted to the PVR must be placed in one of these lists,
00258     depending on its characteristics.
00259
00260     @{
00261 */
00262 #define PVR_LIST_OP_POLY        0   /**< \brief Opaque polygon list */
00263 #define PVR_LIST_OP_MOD         1   /**< \brief Opaque modifier list */
00264 #define PVR_LIST_TR_POLY        2   /**< \brief Translucent polygon list */
00265 #define PVR_LIST_TR_MOD         3   /**< \brief Translucent modifier list*/
00266 #define PVR_LIST_PT_POLY        4   /**< \brief Punch-thru polygon list */
00267 /** @} */
00268
00269 /** \defgroup pvr_shading_types     PVR shading modes
00270
00271     Each polygon can define how it wants to be shaded, be it with flat or
00272     Gouraud shading using these constants in the appropriate place in its
00273     pvr_poly_cxt_t.
00274
00275     @{
00276 */
00277 #define PVR_SHADE_FLAT          0   /**< \brief Use flat shading */
00278 #define PVR_SHADE_GOURAUD       1   /**< \brief Use Gouraud shading */
00279 /** @} */
00280
00281 /** \defgroup pvr_depth_modes       PVR depth comparison modes
00282
00283     These set the depth function used for comparisons.
00284
00285     @{
00286 */
00287 #define PVR_DEPTHCMP_NEVER      0   /**< \brief Never pass */
00288 #define PVR_DEPTHCMP_LESS       1   /**< \brief Less than */
00289 #define PVR_DEPTHCMP_EQUAL      2   /**< \brief Equal to */
00290 #define PVR_DEPTHCMP_LEQUAL     3   /**< \brief Less than or equal to */
00291 #define PVR_DEPTHCMP_GREATER    4   /**< \brief Greater than */
00292 #define PVR_DEPTHCMP_NOTEQUAL   5   /**< \brief Not equal to */
00293 #define PVR_DEPTHCMP_GEQUAL     6   /**< \brief Greater than or equal to */
00294 #define PVR_DEPTHCMP_ALWAYS     7   /**< \brief Always pass */
00295 /** @} */
00296
00297 /** \defgroup pvr_cull_modes        PVR culling modes
00298
00299     These culling modes can be set by polygons to determine when they are
00300     culled. They work pretty much as you'd expect them to if you've ever used
00301     any 3D hardware before.
00302
00303     @{
00304 */
00305 #define PVR_CULLING_NONE        0   /**< \brief Disable culling */
00306 #define PVR_CULLING_SMALL       1   /**< \brief Cull if small */
00307 #define PVR_CULLING_CCW         2   /**< \brief Cull if counterclockwise */
00308 #define PVR_CULLING_CW          3   /**< \brief Cull if clockwise */
00309 /** @} */
00310
00311 /** \defgroup pvr_depth_switch      Enable or disable PVR depth writes
00312     @{
00313 */
00314 #define PVR_DEPTHWRITE_ENABLE   0   /**< \brief Update the Z value */
00315 #define PVR_DEPTHWRITE_DISABLE  1   /**< \brief Do not update the Z value */
00316 /** @} */
00317
00318 /** \defgroup pvr_txr_switch        Enable or disable texturing on polygons
00319     @{
00320 */
00321 #define PVR_TEXTURE_DISABLE     0   /**< \brief Disable texturing */
00322 #define PVR_TEXTURE_ENABLE      1   /**< \brief Enable texturing */
00323 /** @} */
00324
00325 /** \defgroup pvr_blend_modes       PVR blending modes
00326
00327     These are all the blending modes that can be done with regard to alpha
00328     blending on the PVR.
00329
00330     @{
00331 */
00332 #define PVR_BLEND_ZERO          0   /**< \brief None of this color */
00333 #define PVR_BLEND_ONE           1   /**< \brief All of this color */
00334 #define PVR_BLEND_DESTCOLOR     2   /**< \brief Destination color */
00335 #define PVR_BLEND_INVDESTCOLOR  3   /**< \brief Inverse of destination color */
00336 #define PVR_BLEND_SRCALPHA      4   /**< \brief Blend with source alpha */
```

```
00337 #define PVR_BLEND_INVSRCALPHA   5   /**< \brief Blend with inverse source alpha */
00338 #define PVR_BLEND_DESTALPHA     6   /**< \brief Blend with destination alpha */
00339 #define PVR_BLEND_INVDESTALPHA  7   /**< \brief Blend with inverse destination alpha */
00340 /** @} */
00341
00342 /** \defgroup pvr_blend_switch      Enable or disable blending
00343     @{
00344 */
00345 #define PVR_BLEND_DISABLE       0   /**< \brief Disable blending */
00346 #define PVR_BLEND_ENABLE        1   /**< \brief Enable blending */
00347 /** @} */
00348
00349 /** \defgroup pvr_fog_types         PVR fog modes
00350
00351     Each polygon can decide what fog type is used with regard to it using these
00352     constants in its pvr_poly_cxt_t.
00353
00354     @{
00355 */
00356 #define PVR_FOG_TABLE           0   /**< \brief Table fog */
00357 #define PVR_FOG_VERTEX          1   /**< \brief Vertex fog */
00358 #define PVR_FOG_DISABLE         2   /**< \brief Disable fog */
00359 #define PVR_FOG_TABLE2          3   /**< \brief Table fog mode 2 */
00360 /** @} */
00361
00362 /** \defgroup pvr_clip_modes        PVR clipping modes
00363
00364     These control how primitives are clipped against the user clipping area.
00365
00366     @{
00367 */
00368 #define PVR_USERCLIP_DISABLE    0   /**< \brief Disable clipping */
00369 #define PVR_USERCLIP_INSIDE     2   /**< \brief Enable clipping inside area */
00370 #define PVR_USERCLIP_OUTSIDE    3   /**< \brief Enable clipping outside area */
00371 /** @} */
00372
00373 /** \defgroup pvr_colclamp_switch   Enable or disable color clamping
00374
00375     Enabling color clamping will clamp colors between the minimum and maximum
00376     values before any sort of fog processing.
00377
00378     @{
00379 */
00380 #define PVR_CLRCLAMP_DISABLE    0   /**< \brief Disable color clamping */
00381 #define PVR_CLRCLAMP_ENABLE     1   /**< \brief Enable color clamping */
00382 /** @} */
00383
00384 /** \defgroup pvr_offset_switch     Enable or disable offset color
00385
00386     Enabling offset color calculation allows for "specular" like effects on a
00387     per-vertex basis, by providing an additive color in the calculation of the
00388     final pixel colors. In vertex types with a "oargb" parameter, that's what it
00389     is for.
00390
00391     Note that this must be enabled for bumpmap polygons in order to allow you to
00392     specify the parameters in the oargb field of the vertices.
00393
00394     @{
00395 */
00396 #define PVR_SPECULAR_DISABLE    0   /**< \brief Disable offset colors */
00397 #define PVR_SPECULAR_ENABLE     1   /**< \brief Enable offset colors */
00398 /** @} */
00399
00400 /** \defgroup pvr_alpha_switch      Enable or disable alpha blending
00401
00402     This causes the alpha value in the vertex color to be paid attention to. It
00403     really only makes sense to enable this for translucent or punch-thru polys.
00404
00405     @{
00406 */
00407 #define PVR_ALPHA_DISABLE       0   /**< \brief Disable alpha blending */
00408 #define PVR_ALPHA_ENABLE        1   /**< \brief Enable alpha blending */
00409 /** @} */
00410
00411 /** \defgroup pvr_txralpha_switch   Enable or disable texture alpha blending
00412
00413     This causes the alpha value in the texel color to be paid attention to. It
00414     really only makes sense to enable this for translucent or punch-thru polys.
00415
00416     @{
00417 */
```

```
00418 #define PVR_TXRALPHA_ENABLE     0   /**< \brief Enable alpha blending */
00419 #define PVR_TXRALPHA_DISABLE    1   /**< \brief Disable alpha blending */
00420 /** @} */
00421
00422 /** \defgroup pvr_uv_flip           Enable or disable U/V flipping on the PVR
00423
00424     These flags determine what happens when U/V coordinate values exceed 1.0.
00425     In any of the flipped cases, the specified coordinate value will flip around
00426     after 1.0, essentially mirroring the image. So, if you displayed an image
00427     with a U coordinate of 0.0 on the left hand side and 2.0 on the right hand
00428     side with U flipping turned on, you'd have an image that was displayed twice
00429     as if mirrored across the middle. This mirroring behavior happens at every
00430     unit boundary (so at 2.0 it returns to normal, at 3.0 it flips, etc).
00431
00432     The default case is to disable mirroring. In addition, clamping of the U/V
00433     coordinates by PVR_UVCLAMP_U, PVR_UVCLAMP_V, or PVR_UVCLAMP_UV will disable
00434     the mirroring behavior.
00435     @{
00436 */
00437 #define PVR_UVFLIP_NONE         0   /**< \brief No flipped coordinates */
00438 #define PVR_UVFLIP_V            1   /**< \brief Flip V only */
00439 #define PVR_UVFLIP_U            2   /**< \brief Flip U only */
00440 #define PVR_UVFLIP_UV           3   /**< \brief Flip U and V */
00441 /** @} */
00442
00443 /** \defgroup pvr_uv_clamp          Enable or disable clamping of U/V on the PVR
00444
00445     These flags determine whether clamping will be applied to U/V coordinate
00446     values that exceed 1.0. If enabled, these modes will explicitly override the
00447     flip/mirroring modes (PVR_UVFLIP_U, PVR_UVFLIP_V, and PVR_UVFLIP_UV), and
00448     will instead ensure that the coordinate(s) in question never exceed 1.0.
00449     @{
00450 */
00451 #define PVR_UVCLAMP_NONE        0   /**< \brief Disable clamping */
00452 #define PVR_UVCLAMP_V           1   /**< \brief Clamp V only */
00453 #define PVR_UVCLAMP_U           2   /**< \brief Clamp U only */
00454 #define PVR_UVCLAMP_UV          3   /**< \brief Clamp U and V */
00455 /** @} */
00456
00457 /** \defgroup pvr_filter_modes      PVR texture sampling modes
00458     @{
00459 */
00460 #define PVR_FILTER_NONE         0   /**< \brief No filtering (point sample) */
00461 #define PVR_FILTER_NEAREST      0   /**< \brief No filtering (point sample) */
00462 #define PVR_FILTER_BILINEAR     2   /**< \brief Bilinear interpolation */
00463 #define PVR_FILTER_TRILINEAR1   4   /**< \brief Trilinear interpolation pass 1 */
00464 #define PVR_FILTER_TRILINEAR2   6   /**< \brief Trilinear interpolation pass 2 */
00465 /** @} */
00466
00467 /** \defgroup pvr_mip_bias          PVR mipmap bias modes
00468     @{
00469 */
00470 #define PVR_MIPBIAS_NORMAL      PVR_MIPBIAS_1_00    /* txr_mipmap_bias */
00471 #define PVR_MIPBIAS_0_25        1
00472 #define PVR_MIPBIAS_0_50        2
00473 #define PVR_MIPBIAS_0_75        3
00474 #define PVR_MIPBIAS_1_00        4
00475 #define PVR_MIPBIAS_1_25        5
00476 #define PVR_MIPBIAS_1_50        6
00477 #define PVR_MIPBIAS_1_75        7
00478 #define PVR_MIPBIAS_2_00        8
00479 #define PVR_MIPBIAS_2_25        9
00480 #define PVR_MIPBIAS_2_50        10
00481 #define PVR_MIPBIAS_2_75        11
00482 #define PVR_MIPBIAS_3_00        12
00483 #define PVR_MIPBIAS_3_25        13
00484 #define PVR_MIPBIAS_3_50        14
00485 #define PVR_MIPBIAS_3_75        15
00486 /** @} */
00487
00488 /** \defgroup pvr_txrenv_modes      Texture color calculation modes
00489     @{
00490 */
00491 #define PVR_TXRENV_REPLACE          0   /**< \brief C = Ct, A = At */
00492 #define PVR_TXRENV_MODULATE         1   /**< \brief  C = Cs * Ct, A = At */
00493 #define PVR_TXRENV_DECAL            2   /**< \brief C = (Cs * At) + (Cs * (1-At)), A = As */
00494 #define PVR_TXRENV_MODULATEALPHA    3   /**< \brief C = Cs * Ct, A = As * At */
00495 /** @} */
00496
00497 /** \defgroup pvr_mip_switch        Enable or disable PVR mipmap processing
00498     @{
```

```
00499 */
00500 #define PVR_MIPMAP_DISABLE      0   /**< \brief Disable mipmap processing */
00501 #define PVR_MIPMAP_ENABLE       1   /**< \brief Enable mipmap processing */
00502 /** @} */
00503
00504 /** \defgroup pvr_txr_fmts          PVR texture formats
00505
00506     These are the texture formats that the PVR supports. Note that some of
00507     these, you can OR together with other values.
00508
00509     @{
00510 */
00511 #define PVR_TXRFMT_NONE         0           /**< \brief No texture */
00512 #define PVR_TXRFMT_VQ_DISABLE   (0 « 30)    /**< \brief Not VQ encoded */
00513 #define PVR_TXRFMT_VQ_ENABLE    (1 « 30)    /**< \brief VQ encoded */
00514 #define PVR_TXRFMT_ARGB1555     (0 « 27)    /**< \brief 16-bit ARGB1555 */
00515 #define PVR_TXRFMT_RGB565       (1 « 27)    /**< \brief 16-bit RGB565 */
00516 #define PVR_TXRFMT_ARGB4444     (2 « 27)    /**< \brief 16-bit ARGB4444 */
00517 #define PVR_TXRFMT_YUV422       (3 « 27)    /**< \brief YUV422 format */
00518 #define PVR_TXRFMT_BUMP         (4 « 27)    /**< \brief Bumpmap format */
00519 #define PVR_TXRFMT_PAL4BPP      (5 « 27)    /**< \brief 4BPP paletted format */
00520 #define PVR_TXRFMT_PAL8BPP      (6 « 27)    /**< \brief 8BPP paletted format */
00521 #define PVR_TXRFMT_TWIDDLED     (0 « 26)    /**< \brief Texture is twiddled */
00522 #define PVR_TXRFMT_NONTWIDDLED  (1 « 26)    /**< \brief Texture is not twiddled */
00523 #define PVR_TXRFMT_NOSTRIDE     (0 « 21)    /**< \brief Texture is not strided */
00524 #define PVR_TXRFMT_STRIDE       (1 « 21)    /**< \brief Texture is strided */
00525
00526 /* OR one of these into your texture format if you need it. Note that
00527    these coincide with the twiddled/stride bits, so you can't have a
00528    non-twiddled/strided texture that's paletted! */
00529 /** \brief  8BPP palette selector
00530     \param  x               The palette index */
00531 #define PVR_TXRFMT_8BPP_PAL(x)  ((x) « 25)
00532
00533 /** \brief 4BPP palette selector
00534     \param  x               The palette index */
00535 #define PVR_TXRFMT_4BPP_PAL(x)  ((x) « 21)
00536 /** @} */
00537
00538 /** \defgroup pvr_color_fmts        PVR vertex color formats
00539
00540     These control how colors are represented in polygon data.
00541
00542     @{
00543 */
00544 #define PVR_CLRFMT_ARGBPACKED       0   /**< \brief 32-bit integer ARGB */
00545 #define PVR_CLRFMT_4FLOATS          1   /**< \brief 4 floating point values */
00546 #define PVR_CLRFMT_INTENSITY        2   /**< \brief Intensity color */
00547 #define PVR_CLRFMT_INTENSITY_PREV   3   /**< \brief Use last intensity */
00548 /** @} */
00549
00550 /** \defgroup pvr_uv_fmts           PVR U/V data format control
00551     @{
00552 */
00553 #define PVR_UVFMT_32BIT         0   /**< \brief 32-bit floating point U/V */
00554 #define PVR_UVFMT_16BIT         1   /**< \brief 16-bit floating point U/V */
00555 /** @} */
00556
00557 /** \defgroup pvr_mod_switch        Enable or disable modifier effects
00558     @{
00559 */
00560 #define PVR_MODIFIER_DISABLE    0   /**< \brief Disable modifier effects */
00561 #define PVR_MODIFIER_ENABLE     1   /**< \brief Enable modifier effects */
00562 /** @} */
00563
00564 #define PVR_MODIFIER_CHEAP_SHADOW   0
00565 #define PVR_MODIFIER_NORMAL         1
00566
00567 /** \defgroup pvr_mod_modes         Modifier volume mode parameters
00568
00569     All triangles in a single modifier volume should be of the other poly type,
00570     except for the last one. That should be either of the other two types,
00571     depending on whether you want an inclusion or exclusion volume.
00572
00573     @{
00574 */
00575 #define PVR_MODIFIER_OTHER_POLY         0   /**< \brief Not the last polygon in the volume */
00576 #define PVR_MODIFIER_INCLUDE_LAST_POLY  1   /**< \brief Last polygon, inclusion volume */
00577 #define PVR_MODIFIER_EXCLUDE_LAST_POLY  2   /**< \brief Last polygon, exclusion volume */
00578 /** @} */
00579
```

```
00580
00581 /** \brief  PVR polygon header.
00582
00583     This is the hardware equivalent of a rendering context; you'll create one of
00584     these from your pvr_poly_cxt_t and use it for submission to the hardware.
00585
00586     \headerfile dc/pvr.h
00587 */
00588 typedef struct {
00589     uint32_t cmd;               /**< \brief TA command */
00590     uint32_t mode1;             /**< \brief Parameter word 1 */
00591     uint32_t mode2;             /**< \brief Parameter word 2 */
00592     uint32_t mode3;             /**< \brief Parameter word 3 */
00593     uint32_t d1;                /**< \brief Dummy value */
00594     uint32_t d2;                /**< \brief Dummy value */
00595     uint32_t d3;                /**< \brief Dummy value */
00596     uint32_t d4;                /**< \brief Dummy value */
00597 } pvr_poly_hdr_t;
00598
00599 /** \brief  PVR polygon header with intensity color.
00600
00601     This is the equivalent of pvr_poly_hdr_t, but for use with intensity color.
00602
00603     \headerfile dc/pvr.h
00604 */
00605 typedef struct {
00606     uint32_t cmd;               /**< \brief TA command */
00607     uint32_t mode1;             /**< \brief Parameter word 1 */
00608     uint32_t mode2;             /**< \brief Parameter word 2 */
00609     uint32_t mode3;             /**< \brief Parameter word 3 */
00610     float    a;                 /**< \brief Face color alpha component */
00611     float    r;                 /**< \brief Face color red component */
00612     float    g;                 /**< \brief Face color green component */
00613     float    b;                 /**< \brief Face color blue component */
00614 } pvr_poly_ic_hdr_t;
00615
00616 /** \brief  PVR polygon header to be used with modifier volumes.
00617
00618     This is the equivalent of a pvr_poly_hdr_t for use when a polygon is to be
00619     used with modifier volumes.
00620
00621     \headerfile dc/pvr.h
00622 */
00623 typedef struct {
00624     uint32_t cmd;               /**< \brief TA command */
00625     uint32_t mode1;             /**< \brief Parameter word 1 */
00626     uint32_t mode2_0;           /**< \brief Parameter word 2 (outside volume) */
00627     uint32_t mode3_0;           /**< \brief Parameter word 3 (outside volume) */
00628     uint32_t mode2_1;           /**< \brief Parameter word 2 (inside volume) */
00629     uint32_t mode3_1;           /**< \brief Parameter word 3 (inside volume) */
00630     uint32_t d1;                /**< \brief Dummy value */
00631     uint32_t d2;                /**< \brief Dummy value */
00632 } pvr_poly_mod_hdr_t;
00633
00634 /** \brief  PVR polygon header specifically for sprites.
00635
00636     This is the equivalent of a pvr_poly_hdr_t for use when a quad/sprite is to
00637     be rendered. Note that the color data is here, not in the vertices.
00638
00639     \headerfile dc/pvr.h
00640 */
00641 typedef struct {
00642     uint32_t cmd;               /**< \brief TA command */
00643     uint32_t mode1;             /**< \brief Parameter word 1 */
00644     uint32_t mode2;             /**< \brief Parameter word 2 */
00645     uint32_t mode3;             /**< \brief Parameter word 3 */
00646     uint32_t argb;              /**< \brief Sprite face color */
00647     uint32_t oargb;             /**< \brief Sprite offset color */
00648     uint32_t d1;                /**< \brief Dummy value */
00649     uint32_t d2;                /**< \brief Dummy value */
00650 } pvr_sprite_hdr_t;
00651
00652 /** \brief  Modifier volume header.
00653
00654     This is the header that should be submitted when dealing with setting a
00655     modifier volume.
00656
00657     \headerfile dc/pvr.h
00658 */
00659 typedef struct {
00660     uint32_t cmd;               /**< \brief TA command */
```

```
00661     uint32_t mode1;             /**< \brief Parameter word 1 */
00662     uint32_t d1;                /**< \brief Dummy value */
00663     uint32_t d2;                /**< \brief Dummy value */
00664     uint32_t d3;                /**< \brief Dummy value */
00665     uint32_t d4;                /**< \brief Dummy value */
00666     uint32_t d5;                /**< \brief Dummy value */
00667     uint32_t d6;                /**< \brief Dummy value */
00668 } pvr_mod_hdr_t;
00669
00670 /** \brief  Generic PVR vertex type.
00671
00672     The PVR chip itself supports many more vertex types, but this is the main
00673     one that can be used with both textured and non-textured polygons, and is
00674     fairly fast.
00675
00676     \headerfile dc/pvr.h
00677 */
00678 typedef struct {
00679     uint32_t flags;             /**< \brief TA command (vertex flags) */
00680     float    x;                 /**< \brief X coordinate */
00681     float    y;                 /**< \brief Y coordinate */
00682     float    z;                 /**< \brief Z coordinate */
00683     float    u;                 /**< \brief Texture U coordinate */
00684     float    v;                 /**< \brief Texture V coordinate */
00685     uint32_t argb;              /**< \brief Vertex color */
00686     uint32_t oargb;             /**< \brief Vertex offset color */
00687 } pvr_vertex_t;
00688
00689 /** \brief  PVR vertex type: Non-textured, packed color, affected by modifier
00690             volume.
00691
00692     This vertex type has two copies of colors. The second color is used when
00693     enclosed within a modifier volume.
00694
00695     \headerfile dc/pvr.h
00696 */
00697 typedef struct {
00698     uint32_t flags;             /**< \brief TA command (vertex flags) */
00699     float    x;                 /**< \brief X coordinate */
00700     float    y;                 /**< \brief Y coordinate */
00701     float    z;                 /**< \brief Z coordinate */
00702     uint32_t argb0;             /**< \brief Vertex color (outside volume) */
00703     uint32_t argb1;             /**< \brief Vertex color (inside volume) */
00704     uint32_t d1;                /**< \brief Dummy value */
00705     uint32_t d2;                /**< \brief Dummy value */
00706 } pvr_vertex_pcm_t;
00707
00708 /** \brief  PVR vertex type: Textured, packed color, affected by modifer volume.
00709
00710     Note that this vertex type has two copies of colors, offset colors, and
00711     texture coords. The second set of texture coords, colors, and offset colors
00712     are used when enclosed within a modifer volume.
00713
00714     \headerfile dc/pvr.h
00715 */
00716 typedef struct {
00717     uint32_t flags;             /**< \brief TA command (vertex flags) */
00718     float    x;                /**< \brief X coordinate */
00719     float    y;                /**< \brief Y coordinate */
00720     float    z;                /**< \brief Z coordinate */
00721     float    u0;               /**< \brief Texture U coordinate (outside) */
00722     float    v0;               /**< \brief Texture V coordinate (outside) */
00723     uint32_t argb0;             /**< \brief Vertex color (outside) */
00724     uint32_t oargb0;            /**< \brief Vertex offset color (outside) */
00725     float    u1;               /**< \brief Texture U coordinate (inside) */
00726     float    v1;               /**< \brief Texture V coordinate (inside) */
00727     uint32_t argb1;             /**< \brief Vertex color (inside) */
00728     uint32_t oargb1;            /**< \brief Vertex offset color (inside) */
00729     uint32_t d1;                /**< \brief Dummy value */
00730     uint32_t d2;                /**< \brief Dummy value */
00731     uint32_t d3;                /**< \brief Dummy value */
00732     uint32_t d4;                /**< \brief Dummy value */
00733 } pvr_vertex_tpcm_t;
00734
00735 /** \brief  PVR vertex type: Textured sprite.
00736
00737     This vertex type is to be used with the sprite polygon header and the sprite
00738     related commands to draw textured sprites. Note that there is no fourth Z
00739     coordinate. I suppose it just gets interpolated?
00740
00741     The U/V coordinates in here are in the 16-bit per coordinate form. Also,
```

```
00742     like the fourth Z value, there is no fourth U or V, so it must get
00743     interpolated from the others.
00744
00745     \headerfile dc/pvr.h
00746 */
00747 typedef struct {
00748     uint32_t flags;             /**< \brief TA command (vertex flags) */
00749     float   ax;                 /**< \brief First X coordinate */
00750     float   ay;                 /**< \brief First Y coordinate */
00751     float   az;                 /**< \brief First Z coordinate */
00752     float   bx;                 /**< \brief Second X coordinate */
00753     float   by;                 /**< \brief Second Y coordinate */
00754     float   bz;                 /**< \brief Second Z coordinate */
00755     float   cx;                 /**< \brief Third X coordinate */
00756     float   cy;                 /**< \brief Third Y coordinate */
00757     float   cz;                 /**< \brief Third Z coordinate */
00758     float   dx;                 /**< \brief Fourth X coordinate */
00759     float   dy;                 /**< \brief Fourth Y coordinate */
00760     uint32_t dummy;             /**< \brief Dummy value */
00761     uint32_t auv;               /**< \brief First U/V texture coordinates */
00762     uint32_t buv;               /**< \brief Second U/V texture coordinates */
00763     uint32_t cuv;               /**< \brief Third U/V texture coordinates */
00764 } pvr_sprite_txr_t;
00765
00766 /** \brief  PVR vertex type: Untextured sprite.
00767
00768     This vertex type is to be used with the sprite polygon header and the sprite
00769     related commands to draw untextured sprites (aka, quads).
00770 */
00771 typedef struct {
00772     uint32_t flags;             /**< \brief TA command (vertex flags) */
00773     float   ax;                 /**< \brief First X coordinate */
00774     float   ay;                 /**< \brief First Y coordinate */
00775     float   az;                 /**< \brief First Z coordinate */
00776     float   bx;                 /**< \brief Second X coordinate */
00777     float   by;                 /**< \brief Second Y coordinate */
00778     float   bz;                 /**< \brief Second Z coordinate */
00779     float   cx;                 /**< \brief Third X coordinate */
00780     float   cy;                 /**< \brief Third Y coordinate */
00781     float   cz;                 /**< \brief Third Z coordinate */
00782     float   dx;                 /**< \brief Fourth X coordinate */
00783     float   dy;                 /**< \brief Fourth Y coordinate */
00784     uint32_t d1;                /**< \brief Dummy value */
00785     uint32_t d2;                /**< \brief Dummy value */
00786     uint32_t d3;                /**< \brief Dummy value */
00787     uint32_t d4;                /**< \brief Dummy value */
00788 } pvr_sprite_col_t;
00789
00790 /** \brief  PVR vertex type: Modifier volume.
00791
00792     This vertex type is to be used with the modifier volume header to specify
00793     triangular modifier areas.
00794 */
00795 typedef struct {
00796     uint32_t flags;             /**< \brief TA command (vertex flags) */
00797     float   ax;                 /**< \brief First X coordinate */
00798     float   ay;                 /**< \brief First Y coordinate */
00799     float   az;                 /**< \brief First Z coordinate */
00800     float   bx;                 /**< \brief Second X coordinate */
00801     float   by;                 /**< \brief Second Y coordinate */
00802     float   bz;                 /**< \brief Second Z coordinate */
00803     float   cx;                 /**< \brief Third X coordinate */
00804     float   cy;                 /**< \brief Third Y coordinate */
00805     float   cz;                 /**< \brief Third Z coordinate */
00806     uint32_t d1;                /**< \brief Dummy value */
00807     uint32_t d2;                /**< \brief Dummy value */
00808     uint32_t d3;                /**< \brief Dummy value */
00809     uint32_t d4;                /**< \brief Dummy value */
00810     uint32_t d5;                /**< \brief Dummy value */
00811     uint32_t d6;                /**< \brief Dummy value */
00812 } pvr_modifier_vol_t;
00813
00814 /** \brief  Pack four floating point color values into a 32-bit integer form.
00815
00816     All of the color values should be between 0 and 1.
00817
00818     \param  a               Alpha value
00819     \param  r               Red value
00820     \param  g               Green value
00821     \param  b               Blue value
00822     \return                 The packed color value
```

```
00823 */
00824 #define PVR_PACK_COLOR(a, r, g, b) ( \
00825                                     ( ((uint8)( a * 255 ) ) « 24 ) | \
00826                                     ( ((uint8)( r * 255 ) ) « 16 ) | \
00827                                     ( ((uint8)( g * 255 ) ) « 8 ) | \
00828                                     ( ((uint8)( b * 255 ) ) « 0 ) )
00829
00830 /** \brief  Pack two floating point coordinates into one 32-bit value,
00831             truncating them to 16-bits each.
00832
00833     \param  u               First coordinate to pack
00834     \param  v               Second coordinate to pack
00835     \return                 The packed coordinates
00836 */
00837 static inline uint32_t PVR_PACK_16BIT_UV(float u, float v) {
00838     union {
00839         float f;
00840         uint32_t i;
00841     } u2, v2;
00842
00843     u2.f = u;
00844     v2.f = v;
00845
00846     return (u2.i & 0xFFFF0000) | (v2.i » 16);
00847 }
00848
00849 /** \defgroup pvr_commands          TA command values
00850
00851     These are are appropriate values for TA commands. Use whatever goes with the
00852     primitive type you're using.
00853
00854     @{
00855 */
00856 #define PVR_CMD_POLYHDR     0x80840000  /**< \brief PVR polygon header.
00857 Striplength set to 2 */
00858 #define PVR_CMD_VERTEX      0xe0000000  /**< \brief PVR vertex data */
00859 #define PVR_CMD_VERTEX_EOL  0xf0000000  /**< \brief PVR vertex, end of strip */
00860 #define PVR_CMD_USERCLIP    0x20000000  /**< \brief PVR user clipping area */
00861 #define PVR_CMD_MODIFIER    0x80000000  /**< \brief PVR modifier volume */
00862 #define PVR_CMD_SPRITE      0xA0000000  /**< \brief PVR sprite header */
00863 /** @} */
00864
00865 /** \defgroup pvr_bitmasks          Constants and bitmasks for handling polygon
00866                                     headers.
00867
00868     Note that thanks to the arrangement of constants, this is mainly a matter of
00869     bit shifting to compile headers...
00870
00871     @{
00872 */
00873 #define PVR_TA_CMD_TYPE_SHIFT       24
00874 #define PVR_TA_CMD_TYPE_MASK        (7 « PVR_TA_CMD_TYPE_SHIFT)
00875
00876 #define PVR_TA_CMD_USERCLIP_SHIFT   16
00877 #define PVR_TA_CMD_USERCLIP_MASK    (3 « PVR_TA_CMD_USERCLIP_SHIFT)
00878
00879 #define PVR_TA_CMD_CLRFMT_SHIFT     4
00880 #define PVR_TA_CMD_CLRFMT_MASK      (7 « PVR_TA_CMD_CLRFMT_SHIFT)
00881
00882 #define PVR_TA_CMD_SPECULAR_SHIFT   2
00883 #define PVR_TA_CMD_SPECULAR_MASK    (1 « PVR_TA_CMD_SPECULAR_SHIFT)
00884
00885 #define PVR_TA_CMD_SHADE_SHIFT      1
00886 #define PVR_TA_CMD_SHADE_MASK       (1 « PVR_TA_CMD_SHADE_SHIFT)
00887
00888 #define PVR_TA_CMD_UVFMT_SHIFT      0
00889 #define PVR_TA_CMD_UVFMT_MASK       (1 « PVR_TA_CMD_UVFMT_SHIFT)
00890
00891 #define PVR_TA_CMD_MODIFIER_SHIFT   7
00892 #define PVR_TA_CMD_MODIFIER_MASK    (1 «  PVR_TA_CMD_MODIFIER_SHIFT)
00893
00894 #define PVR_TA_CMD_MODIFIERMODE_SHIFT   6
00895 #define PVR_TA_CMD_MODIFIERMODE_MASK    (1 «  PVR_TA_CMD_MODIFIERMODE_SHIFT)
00896
00897 #define PVR_TA_PM1_DEPTHCMP_SHIFT   29
00898 #define PVR_TA_PM1_DEPTHCMP_MASK    (7 « PVR_TA_PM1_DEPTHCMP_SHIFT)
00899
00900 #define PVR_TA_PM1_CULLING_SHIFT    27
00901 #define PVR_TA_PM1_CULLING_MASK     (3 « PVR_TA_PM1_CULLING_SHIFT)
00902
00903 #define PVR_TA_PM1_DEPTHWRITE_SHIFT 26
```

```
00904 #define PVR_TA_PM1_DEPTHWRITE_MASK  (1 « PVR_TA_PM1_DEPTHWRITE_SHIFT)
00905
00906 #define PVR_TA_PM1_TXRENABLE_SHIFT  25
00907 #define PVR_TA_PM1_TXRENABLE_MASK   (1 « PVR_TA_PM1_TXRENABLE_SHIFT)
00908
00909 #define PVR_TA_PM1_MODIFIERINST_SHIFT   29
00910 #define PVR_TA_PM1_MODIFIERINST_MASK    (3 «  PVR_TA_PM1_MODIFIERINST_SHIFT)
00911
00912 #define PVR_TA_PM2_SRCBLEND_SHIFT   29
00913 #define PVR_TA_PM2_SRCBLEND_MASK    (7 « PVR_TA_PM2_SRCBLEND_SHIFT)
00914
00915 #define PVR_TA_PM2_DSTBLEND_SHIFT   26
00916 #define PVR_TA_PM2_DSTBLEND_MASK    (7 « PVR_TA_PM2_DSTBLEND_SHIFT)
00917
00918 #define PVR_TA_PM2_SRCENABLE_SHIFT  25
00919 #define PVR_TA_PM2_SRCENABLE_MASK   (1 « PVR_TA_PM2_SRCENABLE_SHIFT)
00920
00921 #define PVR_TA_PM2_DSTENABLE_SHIFT  24
00922 #define PVR_TA_PM2_DSTENABLE_MASK   (1 « PVR_TA_PM2_DSTENABLE_SHIFT)
00923
00924 #define PVR_TA_PM2_FOG_SHIFT        22
00925 #define PVR_TA_PM2_FOG_MASK     (3 « PVR_TA_PM2_FOG_SHIFT)
00926
00927 #define PVR_TA_PM2_CLAMP_SHIFT      21
00928 #define PVR_TA_PM2_CLAMP_MASK       (1 « PVR_TA_PM2_CLAMP_SHIFT)
00929
00930 #define PVR_TA_PM2_ALPHA_SHIFT      20
00931 #define PVR_TA_PM2_ALPHA_MASK       (1 « PVR_TA_PM2_ALPHA_SHIFT)
00932
00933 #define PVR_TA_PM2_TXRALPHA_SHIFT   19
00934 #define PVR_TA_PM2_TXRALPHA_MASK    (1 « PVR_TA_PM2_TXRALPHA_SHIFT)
00935
00936 #define PVR_TA_PM2_UVFLIP_SHIFT     17
00937 #define PVR_TA_PM2_UVFLIP_MASK      (3 « PVR_TA_PM2_UVFLIP_SHIFT)
00938
00939 #define PVR_TA_PM2_UVCLAMP_SHIFT    15
00940 #define PVR_TA_PM2_UVCLAMP_MASK     (3 « PVR_TA_PM2_UVCLAMP_SHIFT)
00941
00942 #define PVR_TA_PM2_FILTER_SHIFT     12
00943 #define PVR_TA_PM2_FILTER_MASK      (7 « PVR_TA_PM2_FILTER_SHIFT)
00944
00945 #define PVR_TA_PM2_MIPBIAS_SHIFT    8
00946 #define PVR_TA_PM2_MIPBIAS_MASK     (15 « PVR_TA_PM2_MIPBIAS_SHIFT)
00947
00948 #define PVR_TA_PM2_TXRENV_SHIFT     6
00949 #define PVR_TA_PM2_TXRENV_MASK      (3 « PVR_TA_PM2_TXRENV_SHIFT)
00950
00951 #define PVR_TA_PM2_USIZE_SHIFT      3
00952 #define PVR_TA_PM2_USIZE_MASK       (7 « PVR_TA_PM2_USIZE_SHIFT)
00953
00954 #define PVR_TA_PM2_VSIZE_SHIFT      0
00955 #define PVR_TA_PM2_VSIZE_MASK       (7 « PVR_TA_PM2_VSIZE_SHIFT)
00956
00957 #define PVR_TA_PM3_MIPMAP_SHIFT     31
00958 #define PVR_TA_PM3_MIPMAP_MASK      (1 « PVR_TA_PM3_MIPMAP_SHIFT)
00959
00960 #define PVR_TA_PM3_TXRFMT_SHIFT     0
00961 #define PVR_TA_PM3_TXRFMT_MASK      0xffffffff
00962 /** @} */
00963
00964 /**** Register macros ***************************************************/
00965
00966 /* We use these macros to do all PVR register access, so that it's
00967    simple later on to hook them for debugging or whatnot. */
00968
00969 /** \brief  Retrieve a PVR register value
00970     \param  REG             The register to fetch
00971     \return                 The value of that register (32-bits)
00972 */
00973 #define PVR_GET(REG) (* ( (vuint32*)( 0xa05f8000 + (REG) ) ) )
00974
00975 /** \brief  Set a PVR register value
00976     \param  REG             The register to set
00977     \param  VALUE           The value to set in the register (32-bits)
00978 */
00979 #define PVR_SET(REG, VALUE) PVR_GET(REG) = (VALUE)
00980
00981 /* The registers themselves; these are from Maiwe's powervr-reg.txt */
00982 /* Note that 2D specific registers have been excluded for now (like
00983    vsync, hsync, v/h size, etc) */
00984
```

```
00985 /** \defgroup pvr_regs              Offsets to registers of the PVR
00986     @{
00987 */
00988 #define PVR_ID                  0x0000  /**< \brief Chip ID */
00989 #define PVR_REVISION            0x0004  /**< \brief Chip revision */
00990 #define PVR_RESET               0x0008  /**< \brief Reset pins */
00991
00992 #define PVR_ISP_START           0x0014  /**< \brief Start the ISP/TSP */
00993 #define PVR_UNK_0018            0x0018  /**< \brief ?? */
00994
00995 #define PVR_ISP_VERTBUF_ADDR    0x0020  /**< \brief Vertex buffer address for scene rendering */
00996
00997 #define PVR_ISP_TILEMAT_ADDR    0x002c  /**< \brief Tile matrix address for scene rendering */
00998 #define PVR_SPANSORT_CFG        0x0030  /**< \brief ?? -- write 0x101 for now */
00999
01000 #define PVR_BORDER_COLOR        0x0040  /**< \brief Border Color in RGB888 */
01001 #define PVR_FB_CFG_1            0x0044  /**< \brief Framebuffer config 1 */
01002 #define PVR_FB_CFG_2            0x0048  /**< \brief Framebuffer config 2 */
01003 #define PVR_RENDER_MODULO       0x004c  /**< \brief Render modulo */
01004 #define PVR_FB_ADDR             0x0050  /**< \brief Framebuffer start address */
01005 #define PVR_FB_IL_ADDR          0x0054  /**< \brief Framebuffer odd-field start address for interlace */
01006
01007 #define PVR_FB_SIZE             0x005c  /**< \brief Framebuffer display size */
01008 #define PVR_RENDER_ADDR         0x0060  /**< \brief Render output address */
01009 #define PVR_RENDER_ADDR_2       0x0064  /**< \brief Output for strip-buffering */
01010 #define PVR_PCLIP_X             0x0068  /**< \brief Horizontal clipping area */
01011 #define PVR_PCLIP_Y             0x006c  /**< \brief Vertical clipping area */
01012
01013 #define PVR_CHEAP_SHADOW        0x0074  /**< \brief Cheap shadow control */
01014 #define PVR_OBJECT_CLIP         0x0078  /**< \brief Distance for polygon culling */
01015 #define PVR_UNK_007C            0x007c  /**< \brief ?? -- write 0x0027df77 for now */
01016 #define PVR_UNK_0080            0x0080  /**< \brief ?? -- write 7 for now */
01017 #define PVR_TEXTURE_CLIP        0x0084  /**< \brief Distance for texture clipping */
01018 #define PVR_BGPLANE_Z           0x0088  /**< \brief Distance for background plane */
01019 #define PVR_BGPLANE_CFG         0x008c  /**< \brief Background plane config */
01020
01021 #define PVR_UNK_0098            0x0098  /**< \brief ?? -- write 0x00800408 for now */
01022
01023 #define PVR_UNK_00A0            0x00a0  /**< \brief ?? -- write 0x20 for now */
01024
01025 #define PVR_UNK_00A8            0x00a8  /**< \brief ?? -- write 0x15d1c951 for now */
01026
01027 #define PVR_FOG_TABLE_COLOR     0x00b0  /**< \brief Table fog color */
01028 #define PVR_FOG_VERTEX_COLOR    0x00b4  /**< \brief Vertex fog color */
01029 #define PVR_FOG_DENSITY         0x00b8  /**< \brief Fog density coefficient */
01030 #define PVR_COLOR_CLAMP_MAX     0x00bc  /**< \brief RGB Color clamp max */
01031 #define PVR_COLOR_CLAMP_MIN     0x00c0  /**< \brief RGB Color clamp min */
01032 #define PVR_GUN_POS             0x00c4  /**< \brief Light gun position */
01033 #define PVR_HPOS_IRQ            0x00c8  /**< \brief Horizontal position IRQ */
01034 #define PVR_VPOS_IRQ            0x00cc  /**< \brief Vertical position IRQ */
01035 #define PVR_IL_CFG              0x00d0  /**< \brief Interlacing config */
01036 #define PVR_BORDER_X            0x00d4  /**< \brief Window border X position */
01037 #define PVR_SCAN_CLK            0x00d8  /**< \brief Clock and scanline values */
01038 #define PVR_BORDER_Y            0x00dc  /**< \brief Window border Y position */
01039
01040 #define PVR_TEXTURE_MODULO      0x00e4  /**< \brief Output texture width modulo */
01041 #define PVR_VIDEO_CFG           0x00e8  /**< \brief Misc video config */
01042 #define PVR_BITMAP_X            0x00ec  /**< \brief Bitmap window X position */
01043 #define PVR_BITMAP_Y            0x00f0  /**< \brief Bitmap window Y position */
01044 #define PVR_SCALER_CFG          0x00f4  /**< \brief Smoothing scaler */
01045
01046 #define PVR_PALETTE_CFG         0x0108  /**< \brief Palette format */
01047 #define PVR_SYNC_STATUS         0x010c  /**< \brief V/H blank status */
01048 #define PVR_UNK_0110            0x0110  /**< \brief ?? -- write 0x93f39 for now */
01049 #define PVR_UNK_0114            0x0114  /**< \brief ?? -- write 0x200000 for now */
01050 #define PVR_UNK_0118            0x0118  /**< \brief ?? -- write 0x8040 for now */
01051
01052 #define PVR_TA_OPB_START        0x0124  /**< \brief Object Pointer Buffer start for TA usage */
01053 #define PVR_TA_VERTBUF_START    0x0128  /**< \brief Vertex buffer start for TA usage */
01054 #define PVR_TA_OPB_END          0x012c  /**< \brief OPB end for TA usage */
01055 #define PVR_TA_VERTBUF_END      0x0130  /**< \brief Vertex buffer end for TA usage */
01056 #define PVR_TA_OPB_POS          0x0134  /**< \brief Top used memory location in OPB for TA usage */
01057 #define PVR_TA_VERTBUF_POS      0x0138  /**< \brief Top used memory location in vertbuf for TA usage */
01058 #define PVR_TILEMAT_CFG         0x013c  /**< \brief Tile matrix size config */
01059 #define PVR_OPB_CFG             0x0140  /**< \brief Active lists / list size */
01060 #define PVR_TA_INIT             0x0144  /**< \brief Initialize vertex reg. params */
01061 #define PVR_YUV_ADDR            0x0148  /**< \brief YUV conversion destination */
01062 #define PVR_YUV_CFG             0x014c  /**< \brief YUV configuration */
01063 #define PVR_YUV_STAT            0x0150  /**< \brief The number of YUV macroblocks converted */
01064
01065 #define PVR_UNK_0160            0x0160  /**< \brief ?? */
```

```
01066 #define PVR_TA_OPB_INIT         0x0164  /**< \brief Object pointer buffer position init */
01067
01068 #define PVR_FOG_TABLE_BASE      0x0200  /**< \brief Base of the fog table */
01069
01070 #define PVR_PALETTE_TABLE_BASE  0x1000  /**< \brief Base of the palette table */
01071 /** @} */
01072
01073 /* Useful memory locations */
01074 #define PVR_TA_INPUT        0x10000000  /**< \brief TA command input */
01075 #define PVR_TA_YUV_CONV     0x10800000  /**< \brief YUV converter */
01076 #define PVR_TA_TEX_MEM      0x11000000  /**< \brief Texture memory */
01077 #define PVR_RAM_BASE        0xa5000000  /**< \brief PVR RAM (raw) */
01078 #define PVR_RAM_INT_BASE    0xa4000000  /**< \brief PVR RAM (interleaved) */
01079
01080 #define PVR_RAM_SIZE        (8*1024*1024)   /**< \brief RAM size in bytes */
01081
01082 #define PVR_RAM_TOP         (PVR_RAM_BASE + PVR_RAM_SIZE)       /**< \brief Top of raw PVR RAM */
01083 #define PVR_RAM_INT_TOP     (PVR_RAM_INT_BASE + PVR_RAM_SIZE)   /**< \brief Top of int PVR RAM */
01084
01085 /* Register content defines, as needed; these will be filled in over time
01086    as the implementation requires them. There's too many to do otherwise. */
01087
01088 /** \defgroup pvr_reset_vals        Values used to reset parts of the PVR
01089
01090     These values are written to the PVR_RESET register in order to reset the
01091     system or to take it out of reset.
01092
01093     @{
01094 */
01095 #define PVR_RESET_ALL       0xffffffff  /**< \brief Reset the whole PVR */
01096 #define PVR_RESET_NONE      0x00000000  /**< \brief Cancel reset state */
01097 #define PVR_RESET_TA        0x00000001  /**< \brief Reset only the TA */
01098 #define PVR_RESET_ISPTSP    0x00000002  /**< \brief Reset only the ISP/TSP */
01099 /** @} */
01100
01101 #define PVR_ISP_START_GO    0xffffffff  /**< \brief Write to the PVR_ISP_START register to start rendering
      */
01102
01103 #define PVR_TA_INIT_GO      0x80000000  /**< \brief Write to the PVR_TA_INIT register to confirm settings
      */
01104
01105
01106 /* Initialization ****************************************************/
01107
01108 /* Initialization and shutdown: stuff you should only ever have to do
01109    once in your program. */
01110
01111 /** \defgroup pvr_binsizes          Available sizes for primitive bins
01112     @{
01113 */
01114 #define PVR_BINSIZE_0   0   /**< \brief 0-length (disables the list) */
01115 #define PVR_BINSIZE_8   8   /**< \brief 8-word (32-byte) length */
01116 #define PVR_BINSIZE_16  16  /**< \brief 16-word (64-byte) length */
01117 #define PVR_BINSIZE_32  32  /**< \brief 32-word (128-byte) length */
01118 /** @} */
01119
01120 /** \brief  PVR initialization structure
01121
01122     This structure defines how the PVR initializes various parts of the system,
01123     including the primitive bin sizes, the vertex buffer size, and whether
01124     vertex DMA will be enabled.
01125
01126     You essentially fill one of these in, and pass it to pvr_init().
01127
01128     \headerfile dc/pvr.h
01129 */
01130 typedef struct {
01131     /** \brief  Bin sizes.
01132
01133         The bins go in the following order: opaque polygons, opaque modifiers,
01134         translucent polygons, translucent modifiers, punch-thrus
01135     */
01136     int     opb_sizes[5];
01137
01138     /** \brief  Vertex buffer size (should be a nice round number) */
01139     int     vertex_buf_size;
01140
01141     /** \brief  Enable vertex DMA?
01142
01143         Set to non-zero if we want to enable vertex DMA mode. Note that if this
01144         is set, then _all_ enabled lists need to have a vertex buffer assigned,
```

```
01145         even if you never use that list for anything.
01146     */
01147     int     dma_enabled;
01148
01149     /** \brief  Enable horizontal scaling?
01150
01151         Set to non-zero if horizontal scaling is to be enabled. By enabling this
01152         setting and stretching your image to double the native screen width, you
01153         can get horizontal full-screen anti-aliasing. */
01154     int     fsaa_enabled;
01155
01156     /** \brief  Disable translucent polygon autosort?
01157
01158         Set to non-zero to disable translucent polygon autosorting. By enabling
01159         this setting, the PVR acts more like a traditional Z-buffered system
01160         when rendering translucent polygons, meaning you must pre-sort them
01161         yourself if you want them to appear in the right order. */
01162     int     autosort_disabled;
01163
01164
01165     /** \brief  OPB Overflow Count.
01166
01167         Preallocates this many extra OPBs (sets of tile bins), allowing the PVR
01168         to use the extra space when there's too much geometry in the first OPB.
01169
01170         Increasing this value can eliminate artifacts where pieces of geometry
01171         flicker in and out of existence along the tile boundaries. */
01172
01173     int     opb_overflow_count;
01174
01175 } pvr_init_params_t;
01176
01177 /** \brief  Initialize the PVR chip to ready status.
01178
01179     This function enables the specified lists and uses the specified parameters.
01180     Note that bins and vertex buffers come from the texture memory pool, so only
01181     allocate what you actually need. Expects that a 2D mode was initialized
01182     already using the vid_* API.
01183
01184     \param  params          The set of parameters to initialize with
01185     \retval 0               On success
01186     \retval -1              If the PVR has already been initialized or the video
01187                             mode active is not suitable for 3D
01188 */
01189 int pvr_init(pvr_init_params_t *params);
01190
01191 /** \brief  Simple PVR initialization.
01192
01193     This simpler function initializes the PVR using 16/16 for the opaque
01194     and translucent lists' bin sizes, and 0's for everything else. It sets 512KB
01195     of vertex buffer. This is equivalent to the old ta_init_defaults() for now.
01196
01197     \retval 0               On success
01198     \retval -1              If the PVR has already been initialized or the video
01199                             mode active is not suitable for 3D
01200 */
01201 int pvr_init_defaults(void);
01202
01203 /** \brief  Shut down the PVR chip from ready status.
01204
01205     This essentially leaves the video system in 2D mode as it was before the
01206     init.
01207
01208     \retval 0               On success
01209     \retval -1              If the PVR has not been initialized
01210 */
01211 int pvr_shutdown(void);
01212
01213
01214 /* Misc parameters ***************************************************/
01215
01216 /* These are miscellaneous parameters you can set which affect the
01217    rendering process. */
01218
01219 /** \brief  Set the background plane color.
01220
01221     This function sets the color of the area of the screen not covered by any
01222     other polygons.
01223
01224     \param  r               Red component of the color to set
01225     \param  g               Green component of the color to set
```

```
01226     \param  b               Blue component of the color to set
01227 */
01228 void pvr_set_bg_color(float r, float g, float b);
01229
01230 /** \brief  Set cheap shadow parameters.
01231
01232     This function sets up the PVR cheap shadow parameters for use. You can only
01233     specify one scale value per frame, so the effect that you can get from this
01234     is somewhat limited, but if you want simple shadows, this is the easiest way
01235     to do it.
01236
01237     Polygons affected by a shadow modifier volume will effectively multiply
01238     their final color by the scale value set here when shadows are enabled and
01239     the polygon is inside the modifier (or outside for exclusion volumes).
01240
01241     \param  enable          Set to non-zero to enable cheap shadow mode.
01242     \param  scale_value     Floating point value (between 0 and 1) representing
01243                             how colors of polygons affected by and inside the
01244                             volume will be modified by the shadow volume.
01245 */
01246 void pvr_set_shadow_scale(int enable, float scale_value);
01247
01248 /** \brief  Set Z clipping depth.
01249
01250     This function sets the Z clipping depth. The default value for this is
01251     0.0001.
01252
01253     \param  zc              The new value to set the z clip parameter to.
01254 */
01255 void pvr_set_zclip(float zc);
01256
01257 /** \brief  Retrieve the current VBlank count.
01258
01259     This function retrieves the number of VBlank interrupts that have occurred
01260     since the PVR was initialized.
01261
01262     \return                 The number of VBlanks since init
01263 */
01264 int pvr_get_vbl_count(void);
01265
01266 /* Statistics structure */
01267 /** \brief  PVR statistics structure.
01268
01269     This structure is used to hold various statistics about the operation of the
01270     PVR since initialization.
01271
01272     \headerfile dc/pvr.h
01273 */
01274 typedef struct pvr_stats {
01275     uint32_t enabled_list_mask;  /**< \brief Which lists are enabled? */
01276     uint32_t vbl_count;          /**< \brief VBlank count */
01277     int      frame_last_time;    /**< \brief Ready-to-Ready length for the last frame in milliseconds */
01278     float    frame_rate;         /**< \brief Current frame rate (per second) */
01279     int      reg_last_time;      /**< \brief Registration time for the last frame in milliseconds */
01280     int      rnd_last_time;      /**< \brief Rendering time for the last frame in milliseconds */
01281     int      vtx_buffer_used;    /**< \brief Number of bytes used in the vertex buffer for the last frame
    */
01282     int      vtx_buffer_used_max; /**< \brief Number of bytes used in the vertex buffer for the largest
    frame */
01283     int      buf_last_time;      /**< \brief DMA buffer file time for the last frame in milliseconds */
01284     uint32_t frame_count;        /**< \brief Total number of rendered/viewed frames */
01285     /* ... more later as it's implemented ... */
01286 } pvr_stats_t;
01287
01288 /** \brief  Get the current statistics from the PVR.
01289
01290     This function fills in the pvr_stats_t structure passed in with the current
01291     statistics of the system.
01292
01293     \param  stat            The statistics structure to fill in. Must not be
01294                             NULL
01295     \retval 0               On success
01296     \retval -1              If the PVR is not initialized
01297 */
01298 int pvr_get_stats(pvr_stats_t *stat);
01299
01300
01301 /* Palette management ************************************************/
01302
01303 /* In addition to its 16-bit truecolor modes, the PVR also supports some
01304    nice paletted modes. These aren't useful for super high quality images
```

```
01305    most of the time, but they can be useful for doing some interesting
01306    special effects, like the old cheap "worm hole". */
01307
01308 /** \defgroup pvr_palfmts          PVR palette formats
01309
01310     Entries in the PVR's palettes can be of any of these formats. Note that you
01311     can only have one format active at a time.
01312
01313     @{
01314 */
01315 #define PVR_PAL_ARGB1555    0   /**< \brief 16-bit ARGB1555 palette format */
01316 #define PVR_PAL_RGB565      1   /**< \brief 16-bit RGB565 palette format */
01317 #define PVR_PAL_ARGB4444    2   /**< \brief 16-bit ARGB4444 palette format */
01318 #define PVR_PAL_ARGB8888    3   /**< \brief 32-bit ARGB8888 palette format */
01319 /** @} */
01320
01321 /** \brief  Set the palette format.
01322
01323     This function sets the currently active palette format on the PVR. Each
01324     entry in the palette table is 32-bits in length, regardless of what color
01325     format is in use.
01326
01327     Be sure to use care when using the PVR_PAL_ARGB8888 format. Rendering speed
01328     is greatly affected (cut about in half) if you use any filtering with
01329     paletted textures with ARGB8888 entries in the palette.
01330
01331     \param  fmt             The format to use
01332     \see    pvr_palfmts
01333 */
01334 void pvr_set_pal_format(int fmt);
01335
01336 /** \brief  Set a palette value.
01337
01338     Note that while the color format is variable, each entry is still 32-bits in
01339     length regardless (and you only get a total of 1024 of them). If using one
01340     of the 16-bit palette formats, only the low-order 16-bits of the entry are
01341     valid, and the high bits should be filled in with 0.
01342
01343     \param  idx             The index to set to (0-1023)
01344     \param  value           The color value to set in that palette entry
01345 */
01346 static inline void pvr_set_pal_entry(uint32_t idx, uint32_t value) {
01347     PVR_SET(PVR_PALETTE_TABLE_BASE + 4 * idx, value);
01348 }
01349
01350
01351 /* Hardware Fog parameters ****************************************/
01352
01353 /* Thanks to Paul Boese for figuring this stuff out */
01354
01355 /** \brief  Set the table fog color.
01356
01357     This function sets the color of fog for table fog. 0-1 range for all colors.
01358
01359     \param  a               Alpha value of the fog
01360     \param  r               Red value of the fog
01361     \param  g               Green value of the fog
01362     \param  b               Blue value of the fog
01363 */
01364 void pvr_fog_table_color(float a, float r, float g, float b);
01365
01366 /** \brief  Set the vertex fog color.
01367
01368     This function sets the fog color for vertex fog. 0-1 range for all colors.
01369     This function is currently not implemented, as vertex fog is not supported
01370     by KOS. Calling this function will cause an assertion failure.
01371
01372     \param  a               Alpha value of the fog
01373     \param  r               Red value of the fog
01374     \param  g               Green value of the fog
01375     \param  b               Blue value of the fog
01376 */
01377 void pvr_fog_vertex_color(float a, float r, float g, float b);
01378
01379 /** \brief  Set the fog far depth.
01380
01381     This function sets the PVR_FOG_DENSITY register appropriately for the
01382     specified value.
01383
01384     \param  d               The depth to set
01385 */
```

```
01386 void pvr_fog_far_depth(float d);
01387
01388 /** \brief  Initialize the fog table using an exp2 algorithm (like GL_EXP2).
01389
01390     This function will automatically set the PVR_FOG_DENSITY register to
01391     259.999999 as a part of its processing, then set up the fog table.
01392
01393     \param  density         Fog density value
01394 */
01395 void pvr_fog_table_exp2(float density);
01396
01397 /** \brief  Initialize the fog table using an exp algorithm (like GL_EXP).
01398
01399     This function will automatically set the PVR_FOG_DENSITY register to
01400     259.999999 as a part of its processing, then set up the fog table.
01401
01402     \param  density         Fog density value
01403 */
01404 void pvr_fog_table_exp(float density);
01405
01406 /** \brief  Initialize the fog table using a linear algorithm (like GL_LINEAR).
01407
01408     This function will set the PVR_FOG_DENSITY register to the as appropriate
01409     for the end value, and initialize the fog table for perspectively correct
01410     linear fog.
01411
01412     \param  start           Fog start point
01413     \param  end             Fog end point
01414 */
01415 void pvr_fog_table_linear(float start, float end);
01416
01417 /** \brief  Set a custom fog table from float values
01418
01419     This function allows you to specify whatever values you need to for your fog
01420     parameters. All values should be clamped between 0 and 1, and its your
01421     responsibility to set up the PVR_FOG_DENSITY register by calling
01422     pvr_fog_far_depth() with an appropriate value. The table passed in should
01423     have 129 entries, where the 0th entry is farthest from the eye and the last
01424     entry is nearest. Higher values = heavier fog.
01425
01426     \param  tbl1            The table of fog values to set
01427 */
01428 void pvr_fog_table_custom(float tbl1[]);
01429
01430
01431 /* Memory management ************************************************/
01432
01433 /* PVR memory management in KOS uses a modified dlmalloc; see the
01434    source file pvr_mem_core.c for more info. */
01435
01436 /** \brief  Allocate a chunk of memory from texture space.
01437
01438     This function acts as the memory allocator for the PVR texture RAM pool. It
01439     acts exactly as one would expect a malloc() function to act, returning a
01440     normal pointer that can be directly written to if one desires to do so. All
01441     allocations will be aligned to a 32-byte boundary.
01442
01443     \param  size            The amount of memory to allocate
01444     \return                 A pointer to the memory on success, NULL on error
01445 */
01446 pvr_ptr_t pvr_mem_malloc(size_t size);
01447
01448 /** \brief  Free a block of allocated memory in the PVR RAM pool.
01449
01450     This function frees memory previously allocated with pvr_mem_malloc().
01451
01452     \param  chunk           The location of the start of the block to free
01453 */
01454 void pvr_mem_free(pvr_ptr_t chunk);
01455
01456 /** \brief  Return the number of bytes available still in the PVR RAM pool.
01457     \return                 The number of bytes available
01458 */
01459 uint32_t pvr_mem_available(void);
01460
01461 /** \brief  Reset the PVR RAM pool.
01462
01463     This will essentially free any blocks allocated within the pool. There's
01464     generally not many good reasons for doing this.
01465 */
01466 void pvr_mem_reset(void);
```

```
01467
01468 /** \brief  Print the list of allocated blocks in the PVR RAM pool.
01469
01470     This function only works if you've enabled KM_DBG in pvr_mem.c.
01471 */
01472 void pvr_mem_print_list(void);
01473
01474 /** \brief  Print statistics about the PVR RAM pool.
01475
01476     This prints out statistics like what malloc_stats() provides. Also, if
01477     KM_DBG is enabled in pvr_mem.c, it prints the list of allocated blocks.
01478 */
01479 void pvr_mem_stats(void);
01480
01481 /* Scene rendering ***************************************************/
01482
01483 /* This API is used to submit triangle strips to the PVR via the TA
01484    interface in the chip.
01485
01486    An important side note about the PVR is that all primitive types
01487    must be submitted grouped together. If you have 10 polygons for each
01488    list type, then the PVR must receive them via the TA by list type,
01489    with a list delimiter in between.
01490
01491    So there are two modes you can use here. The first mode allows you to
01492    submit data directly to the TA. Your data will be forwarded to the
01493    chip for processing as it is fed to the PVR module. If your data
01494    is easily sorted into the primitive types, then this is the fastest
01495    mode for submitting data.
01496
01497    The second mode allows you to submit data via main-RAM vertex buffers,
01498    which will be queued until the proper primitive type is active. In this
01499    case, each piece of data is copied into the vertex buffer while the
01500    wrong list is activated, and when the proper list becomes activated,
01501    the data is all sent at once. Ideally this would be via DMA, right
01502    now it is by store queues. This has the advantage of allowing you to
01503    send data in any order and have the PVR functions resolve how it should
01504    get sent to the hardware, but it is slower.
01505
01506    The nice thing is that any combination of these modes can be used. You
01507    can assign a vertex buffer for any list, and it will be used to hold the
01508    incoming vertex data until the proper list has come up. Or if the proper
01509    list is already up, the data will be submitted directly. So if most of
01510    your polygons are opaque, and you only have a couple of translucents,
01511    you can set a small buffer to gather translucent data and then it will
01512    get sent when you do a pvr_end_scene().
01513
01514    Thanks to Mikael Kalms for the idea for this API.
01515
01516    Another somewhat subtle point that bears mentioning is that in the normal
01517    case (interrupts enabled) an interrupt handler will automatically take
01518    care of starting a frame rendering (after scene_finish()) and also
01519    flipping pages when appropriate. */
01520
01521 /** \brief  Is vertex DMA enabled?
01522     \return                 Non-zero if vertex DMA was enabled at init time
01523 */
01524 int pvr_vertex_dma_enabled(void);
01525
01526 /** \brief  Setup a vertex buffer for one of the list types.
01527
01528     If the specified list type already has a vertex buffer, it will be replaced
01529     by the new one. Note that each buffer should actually be twice as long as
01530     what you will need to hold two frames worth of data).
01531
01532     You should generally not try to do this at any time besides before a frame
01533     is begun, or Bad Things May Happen.
01534
01535     \param  list            The primitive list to set the buffer for.
01536     \param  buffer          The location of the buffer in main RAM. This must be
01537                             aligned to a 32-byte boundary.
01538     \param  len             The length of the buffer. This must be a multiple of
01539                             64, and must be at least 128 (even if you're not
01540                             using the list).
01541     \return                 The old buffer location (if any)
01542 */
01543 void *pvr_set_vertbuf(pvr_list_t list, void *buffer, int len);
01544
01545 /** \brief  Retrieve a pointer to the current output location in the DMA buffer
01546             for the requested list.
01547
```

```
01548     Vertex DMA must globally be enabled for this to work. Data may be added to
01549     this buffer by the user program directly; however, make sure to call
01550     pvr_vertbuf_written() to notify the system of any such changes.
01551
01552     \param  list            The primitive list to get the buffer for.
01553     \return                 The tail of that list's buffer.
01554 */
01555 void *pvr_vertbuf_tail(pvr_list_t list);
01556
01557 /** \brief  Notify the PVR system that data have been written into the output
01558             buffer for the given list.
01559
01560     This should always be done after writing data directly to these buffers or
01561     it will get overwritten by other data.
01562
01563     \param  list            The primitive list that was modified.
01564     \param  amt             Number of bytes written. Must be a multiple of 32.
01565 */
01566 void pvr_vertbuf_written(pvr_list_t list, uint32_t amt);
01567
01568 /** \brief  Set the translucent polygon sort mode for the next frame.
01569
01570     This function sets the translucent polygon sort mode for the next frame of
01571     output, potentially switching between autosort and presort mode.
01572
01573     For most programs, you'll probably want to set this at initialization time
01574     (with the autosort_disabled field in the pvr_init_params_t structure) and
01575     not mess with it per-frame. It is recommended that if you do use this
01576     function to change the mode that you should set it each frame to ensure that
01577     the mode is set properly.
01578
01579     \param  presort         Set to 1 to set the presort mode for translucent
01580                             polygons, set to 0 to use autosort mode.
01581 */
01582 void pvr_set_presort_mode(int presort);
01583
01584 /** \brief  Begin collecting data for a frame of 3D output to the off-screen
01585             frame buffer.
01586
01587     You must call this function (or pvr_scene_begin_txr()) for ever frame of
01588     output.
01589 */
01590 void pvr_scene_begin(void);
01591
01592 /** \brief  Begin collecting data for a frame of 3D output to the specified
01593             texture.
01594
01595     This function currently only supports outputting at the same size as the
01596     actual screen. Thus, make sure rx and ry are at least large enough for that.
01597     For a 640x480 output, rx will generally be 1024 on input and ry 512, as
01598     these are the smallest values that are powers of two and will hold the full
01599     screen sized output.
01600
01601     \param  txr             The texture to render to.
01602     \param  rx              Width of the texture buffer (in pixels).
01603     \param  ry              Height of the texture buffer (in pixels).
01604 */
01605 void pvr_scene_begin_txr(pvr_ptr_t txr, uint32_t *rx, uint32_t *ry);
01606
01607 /** \brief  Begin collecting data for the given list type.
01608
01609     Lists do not have to be submitted in any particular order, but all types of
01610     a list must be submitted at once (unless vertex DMA mode is enabled).
01611
01612     Note that there is no need to call this function in DMA mode unless you want
01613     to make use of pvr_prim() for compatibility. This function will
01614     automatically call pvr_list_finish() if a list is already opened before
01615     opening the new list.
01616
01617     \param  list            The list to open.
01618     \retval 0               On success.
01619     \retval -1              If the specified list has already been closed.
01620 */
01621 int pvr_list_begin(pvr_list_t list);
01622
01623 /** \brief  End collecting data for the current list type.
01624
01625     Lists can never be opened again within a single frame once they have been
01626     closed. Thus submitting a primitive that belongs in a closed list is
01627     considered an error. Closing a list that is already closed is also an error.
01628
```

```
01629     Note that if you open a list but do not submit any primitives, a blank one
01630     will be submitted to satisfy the hardware. If vertex DMA mode is enabled,
01631     then this simply sets the current list pointer to no list, and none of the
01632     above restrictions apply.
01633
01634     \retval 0               On success.
01635     \retval -1              On error.
01636 */
01637 int pvr_list_finish(void);
01638
01639 /** \brief  Submit a primitive of the current list type.
01640
01641     Note that any values submitted in this fashion will go directly to the
01642     hardware without any sort of buffering, and submitting a primitive of the
01643     wrong type will quite likely ruin your scene. Note that this also will not
01644     work if you haven't begun any list types (i.e., all data is queued). If DMA
01645     is enabled, the primitive will be appended to the end of the currently
01646     selected list's buffer.
01647
01648     \param  data            The primitive to submit.
01649     \param  size            The length of the primitive, in bytes. Must be a
01650                             multiple of 32.
01651     \retval 0               On success.
01652     \retval -1              On error.
01653 */
01654 int pvr_prim(void *data, int size);
01655
01656 /** \brief  Direct Rendering state variable type. */
01657 typedef uint32_t pvr_dr_state_t;
01658
01659 /** \brief  Initialize a state variable for Direct Rendering.
01660
01661     \param  vtx_buf_ptr     A variable of type pvr_dr_state_t to init.
01662 */
01663 #define pvr_dr_init(vtx_buf_ptr) do { \
01664         (vtx_buf_ptr) = 0; \
01665         QACR0 = ((((uint32)PVR_TA_INPUT) >> 26) << 2) & 0x1c; \
01666         QACR1 = ((((uint32)PVR_TA_INPUT) >> 26) << 2) & 0x1c; \
01667     } while(0)
01668
01669 /** \brief  Obtain the target address for Direct Rendering.
01670
01671     \param  vtx_buf_ptr     State variable for Direct Rendering. Should be of
01672                             type pvr_dr_state_t, and must have been initialized
01673                             previously in the scene with pvr_dr_init().
01674     \return                 A write-only destination address where a primitive
01675                             should be written to get ready to submit it to the
01676                             TA in DR mode.
01677 */
01678 #define pvr_dr_target(vtx_buf_ptr) \
01679     ({ (vtx_buf_ptr) ^= 32; \
01680         (pvr_vertex_t *)(MEM_AREA_P4_BASE | (vtx_buf_ptr)); \
01681     })
01682
01683 /** \brief  Commit a primitive written into the Direct Rendering target address.
01684
01685     \param  addr            The address returned by pvr_dr_target(), after you
01686                             have written the primitive to it.
01687 */
01688 #define pvr_dr_commit(addr) __asm__ __volatile__("pref @%0" : : "r" (addr))
01689
01690 /** \brief  Submit a primitive of the given list type.
01691
01692     Data will be queued in a vertex buffer, thus one must be available for the
01693     list specified (will be asserted by the code).
01694
01695     \param  list            The list to submit to.
01696     \param  data            The primitive to submit.
01697     \param  size            The size of the primitive in bytes. This must be a
01698                             multiple of 32.
01699     \retval 0               On success.
01700     \retval -1              On error.
01701 */
01702 int pvr_list_prim(pvr_list_t list, void *data, int size);
01703
01704 /** \brief  Flush the buffered data of the given list type to the TA.
01705
01706     This function is currently not implemented, and calling it will result in an
01707     assertion failure. It is intended to be used later in a "hybrid" mode where
01708     both direct and DMA TA submission is possible.
01709
```

```
01710     \param  list            The list to flush.
01711     \retval -1              On error (it is not possible to succeed).
01712 */
01713 int pvr_list_flush(pvr_list_t list);
01714
01715 /** \brief  Call this after you have finished submitting all data for a frame.
01716
01717     Once this has been called, you can not submit any more data until one of the
01718     pvr_scene_begin() or pvr_scene_begin_txr() functions is called again.
01719
01720     \retval 0               On success.
01721     \retval -1              On error (no scene started).
01722 */
01723 int pvr_scene_finish(void);
01724
01725 /** \brief  Block the caller until the PVR system is ready for another frame to
01726             be submitted.
01727
01728     The PVR system allocates enough space for two frames: one in data collection
01729     mode, and another in rendering mode. If a frame is currently rendering, and
01730     another frame has already been closed, then the caller cannot do anything
01731     else until the rendering frame completes. Note also that the new frame
01732     cannot be activated except during a vertical blanking period, so this
01733     essentially waits until a rendered frame is complete and a vertical blank
01734     happens.
01735
01736     \retval 0               On success. A new scene can be started now.
01737     \retval -1              On error. Something is probably very wrong...
01738 */
01739 int pvr_wait_ready(void);
01740
01741 /** \brief  Check if the PVR system is ready for another frame to be submitted.
01742
01743     \retval 0               If the PVR is ready for a new scene. You must call
01744                             pvr_wait_ready() afterwards, before starting a new
01745                             scene.
01746     \retval -1              If the PVR is not ready for a new scene yet.
01747 */
01748 int pvr_check_ready(void);
01749
01750
01751 /* Primitive handling ***********************************************/
01752
01753 /* These functions help you prepare primitives for loading into the
01754    PVR for scene processing. */
01755
01756 /** \brief  Compile a polygon context into a polygon header.
01757
01758     This function compiles a pvr_poly_cxt_t into the form needed by the hardware
01759     for rendering. This is for use with normal polygon headers.
01760
01761     \param  dst             Where to store the compiled header.
01762     \param  src             The context to compile.
01763 */
01764 void pvr_poly_compile(pvr_poly_hdr_t *dst, pvr_poly_cxt_t *src);
01765
01766 /** \brief  Fill in a polygon context for non-textured polygons.
01767
01768     This function fills in a pvr_poly_cxt_t with default parameters appropriate
01769     for rendering a non-textured polygon in the given list.
01770
01771     \param  dst             Where to store the polygon context.
01772     \param  list            The primitive list to be used.
01773 */
01774 void pvr_poly_cxt_col(pvr_poly_cxt_t *dst, pvr_list_t list);
01775
01776 /** \brief  Fill in a polygon context for a textured polygon.
01777
01778     This function fills in a pvr_poly_cxt_t with default parameters appropriate
01779     for rendering a textured polygon in the given list.
01780
01781     \param  dst             Where to store the polygon context.
01782     \param  list            The primitive list to be used.
01783     \param  textureformat   The format of the texture used.
01784     \param  tw              The width of the texture, in pixels.
01785     \param  th              The height of the texture, in pixels.
01786     \param  textureaddr     A pointer to the texture.
01787     \param  filtering       The type of filtering to use.
01788
01789     \see    pvr_txr_fmts
01790     \see    pvr_filter_modes
```

```
01791 */
01792 void pvr_poly_cxt_txr(pvr_poly_cxt_t *dst, pvr_list_t list,
01793                       int textureformat, int tw, int th, pvr_ptr_t textureaddr,
01794                       int filtering);
01795
01796 /** \brief  Compile a sprite context into a sprite header.
01797
01798     This function compiles a pvr_sprite_cxt_t into the form needed by the
01799     hardware for rendering. This is for use with sprite headers.
01800
01801     \param  dst             Where to store the compiled header.
01802     \param  src             The context to compile.
01803 */
01804 void pvr_sprite_compile(pvr_sprite_hdr_t *dst,
01805                         pvr_sprite_cxt_t *src);
01806
01807 /** \brief  Fill in a sprite context for non-textured sprites.
01808
01809     This function fills in a pvr_sprite_cxt_t with default parameters
01810     appropriate for rendering a non-textured sprite in the given list.
01811
01812     \param  dst             Where to store the sprite context.
01813     \param  list            The primitive list to be used.
01814 */
01815 void pvr_sprite_cxt_col(pvr_sprite_cxt_t *dst, pvr_list_t list);
01816
01817 /** \brief  Fill in a sprite context for a textured sprite.
01818
01819     This function fills in a pvr_sprite_cxt_t with default parameters
01820     appropriate for rendering a textured sprite in the given list.
01821
01822     \param  dst             Where to store the sprite context.
01823     \param  list            The primitive list to be used.
01824     \param  textureformat   The format of the texture used.
01825     \param  tw              The width of the texture, in pixels.
01826     \param  th              The height of the texture, in pixels.
01827     \param  textureaddr     A pointer to the texture.
01828     \param  filtering       The type of filtering to use.
01829
01830     \see    pvr_txr_fmts
01831     \see    pvr_filter_modes
01832 */
01833 void pvr_sprite_cxt_txr(pvr_sprite_cxt_t *dst, pvr_list_t list,
01834                         int textureformat, int tw, int th, pvr_ptr_t textureaddr,
01835                         int filtering);
01836
01837 /** \brief  Create a modifier volume header.
01838
01839     This function fills in a modifier volume header with the parameters
01840     specified. Note that unlike for polygons and sprites, there is no context
01841     step for modifiers.
01842
01843     \param  dst             Where to store the modifier header.
01844     \param  list            The primitive list to be used.
01845     \param  mode            The mode for this modifier.
01846     \param  cull            The culling mode to use.
01847
01848     \see    pvr_mod_modes
01849     \see    pvr_cull_modes
01850 */
01851 void pvr_mod_compile(pvr_mod_hdr_t *dst, pvr_list_t list, uint32_t mode,
01852                      uint32_t cull);
01853
01854 /** \brief  Compile a polygon context into a polygon header that is affected by
01855             modifier volumes.
01856
01857     This function works pretty similarly to pvr_poly_compile(), but compiles
01858     into the header type that is affected by a modifier volume. The context
01859     should have been created with either pvr_poly_cxt_col_mod() or
01860     pvr_poly_cxt_txr_mod().
01861
01862     \param  dst             Where to store the compiled header.
01863     \param  src             The context to compile.
01864 */
01865 void pvr_poly_mod_compile(pvr_poly_mod_hdr_t *dst, pvr_poly_cxt_t *src);
01866
01867 /** \brief  Fill in a polygon context for non-textured polygons affected by a
01868             modifier volume.
01869
01870     This function fills in a pvr_poly_cxt_t with default parameters appropriate
01871     for rendering a non-textured polygon in the given list that will be affected
```

```
01872     by modifier volumes.
01873
01874     \param  dst             Where to store the polygon context.
01875     \param  list            The primitive list to be used.
01876 */
01877 void pvr_poly_cxt_col_mod(pvr_poly_cxt_t *dst, pvr_list_t list);
01878
01879 /** \brief  Fill in a polygon context for a textured polygon affected by
01880             modifier volumes.
01881
01882     This function fills in a pvr_poly_cxt_t with default parameters appropriate
01883     for rendering a textured polygon in the given list and being affected by
01884     modifier volumes.
01885
01886     \param  dst             Where to store the polygon context.
01887     \param  list            The primitive list to be used.
01888     \param  textureformat   The format of the texture used (outside).
01889     \param  tw              The width of the texture, in pixels (outside).
01890     \param  th              The height of the texture, in pixels (outside).
01891     \param  textureaddr     A pointer to the texture (outside).
01892     \param  filtering       The type of filtering to use (outside).
01893     \param  textureformat2  The format of the texture used (inside).
01894     \param  tw2             The width of the texture, in pixels (inside).
01895     \param  th2             The height of the texture, in pixels (inside).
01896     \param  textureaddr2    A pointer to the texture (inside).
01897     \param  filtering2      The type of filtering to use (inside).
01898
01899     \see    pvr_txr_fmts
01900     \see    pvr_filter_modes
01901 */
01902 void pvr_poly_cxt_txr_mod(pvr_poly_cxt_t *dst, pvr_list_t list,
01903                           int textureformat, int tw, int th,
01904                           pvr_ptr_t textureaddr, int filtering,
01905                           int textureformat2, int tw2, int th2,
01906                           pvr_ptr_t textureaddr2, int filtering2);
01907
01908 /* Texture handling ************************************************/
01909
01910 /* Helper functions for handling texture tasks of various kinds. */
01911
01912 /** \brief  Load raw texture data from an SH-4 buffer into PVR RAM.
01913
01914     This essentially just acts as a memcpy() from main RAM to PVR RAM, using
01915     the store queues.
01916
01917     \param  src             The location in main RAM holding the texture.
01918     \param  dst             The location in PVR RAM to copy to.
01919     \param  count           The size of the texture in bytes (must be a multiple
01920                             of 32).
01921 */
01922 void pvr_txr_load(void *src, pvr_ptr_t dst, uint32_t count);
01923
01924 /** \defgroup pvr_txrload_constants     Texture loading constants
01925
01926     These are constants for the flags parameter to pvr_txr_load_ex() or
01927     pvr_txr_load_kimg().
01928
01929     @{
01930 */
01931 #define PVR_TXRLOAD_4BPP            0x01    /**< \brief 4BPP format */
01932 #define PVR_TXRLOAD_8BPP            0x02    /**< \brief 8BPP format */
01933 #define PVR_TXRLOAD_16BPP           0x03    /**< \brief 16BPP format */
01934 #define PVR_TXRLOAD_FMT_MASK        0x0f    /**< \brief Bits used for basic formats */
01935
01936 #define PVR_TXRLOAD_VQ_LOAD         0x10    /**< \brief Do VQ encoding (not supported yet, if ever) */
01937 #define PVR_TXRLOAD_INVERT_Y        0x20    /**< \brief Invert the Y axis while loading */
01938 #define PVR_TXRLOAD_FMT_VQ          0x40    /**< \brief Texture is already VQ encoded */
01939 #define PVR_TXRLOAD_FMT_TWIDDLED    0x80    /**< \brief Texture is already twiddled */
01940 #define PVR_TXRLOAD_FMT_NOTWIDDLE   0x80    /**< \brief Don't twiddle the texture while loading */
01941 #define PVR_TXRLOAD_DMA             0x8000  /**< \brief Use DMA to load the texture */
01942 #define PVR_TXRLOAD_NONBLOCK        0x4000  /**< \brief Use non-blocking loads (only for DMA) */
01943 #define PVR_TXRLOAD_SQ              0x2000  /**< \brief Use store queues to load */
01944 /** @} */
01945
01946 /** \brief  Load texture data from an SH-4 buffer into PVR RAM, twiddling it in
01947             the process.
01948
01949     This function loads a texture to the PVR's RAM with the specified set of
01950     flags. It will currently always twiddle the data, whether you ask it to or
01951     not, and many of the parameters are just plain not supported at all...
01952     Pretty much the only supported flag, other than the format ones is the
```

```
01953     PVR_TXRLOAD_INVERT_Y one.
01954
01955     This will be slower than using pvr_txr_load() in pretty much all cases, so
01956     unless you need to twiddle your texture, just use that instead.
01957
01958     \param  src             The location to copy from.
01959     \param  dst             The location to copy to.
01960     \param  w               The width of the texture, in pixels.
01961     \param  h               The height of the texture, in pixels.
01962     \param  flags           Some set of flags, ORed together.
01963
01964     \see    pvr_txrload_constants
01965 */
01966 void pvr_txr_load_ex(void *src, pvr_ptr_t dst, uint32_t w, uint32_t h, uint32_t flags);
01967
01968 /** \brief  Load a KOS Platform Independent Image (subject to constraint
01969             checking).
01970
01971     This function loads a KOS Platform Independent image to the PVR's RAM with
01972     the specified set of flags. This function, unlike pvr_txr_load_ex() supports
01973     everything in the flags available, other than what's explicitly marked as
01974     not supported.
01975
01976     \param  img             The image to load.
01977     \param  dst             The location to copy to.
01978     \param  flags           Some set of flags, ORed together.
01979
01980     \see    pvr_txrload_constants
01981     \note                   Unless you explicitly tell this function to not
01982                             twiddle the texture (by ORing
01983                             \ref PVR_TXRLOAD_FMT_NOTWIDDLE or it's equivalent
01984                             \ref PVR_TXRLOAD_FMT_TWIDDLED with flags), this
01985                             function will twiddle the texture while loading.
01986                             Keep that in mind when setting the texture format in
01987                             polygon headers later.
01988     \note                   You cannot specify both
01989                             \ref PVR_TXRLOAD_FMT_NOTWIDDLE (or equivalently
01990                             \ref PVR_TXRLOAD_FMT_TWIDDLED) and
01991                             \ref PVR_TXRLOAD_INVERT_Y in the flags.
01992     \note                   DMA and Store Queue based loading is not available
01993                             from this function if it twiddles the texture while
01994                             loading.
01995 */
01996 void pvr_txr_load_kimg(kos_img_t *img, pvr_ptr_t dst, uint32_t flags);
01997
01998
01999 /* PVR DMA ************************************************************/
02000
02001 /** \brief  PVR DMA interrupt callback type.
02002
02003     Functions that act as callbacks when DMA completes should be of this type.
02004     These functions will be called inside an interrupt context, so don't try to
02005     use anything that might stall.
02006
02007     \param  data            User data passed in to the pvr_dma_transfer()
02008                             function.
02009 */
02010 typedef void (*pvr_dma_callback_t)(void *data);
02011
02012 /** \brief  Perform a DMA transfer to the PVR.
02013
02014     This function copies a block of data to the PVR or its memory via DMA. There
02015     are all kinds of constraints that must be fulfilled to actually do this, so
02016     make sure to read all the fine print with the parameter list.
02017
02018     If a callback is specified, it will be called in an interrupt context, so
02019     keep that in mind in writing the callback.
02020
02021     \param  src             Where to copy from. Must be 32-byte aligned.
02022     \param  dest            Where to copy to. Must be 32-byte aligned.
02023     \param  count           The number of bytes to copy. Must be a multiple of
02024                             32.
02025     \param  type            The type of DMA transfer to do (see list of modes).
02026     \param  block           Non-zero if you want the function to block until the
02027                             DMA completes.
02028     \param  callback        A function to call upon completion of the DMA.
02029     \param  cbdata          Data to pass to the callback function.
02030     \retval 0               On success.
02031     \retval -1              On failure. Sets errno as appropriate.
02032
02033     \par    Error Conditions:
```

```
02034     \em     EINPROGRESS - DMA already in progress \n
02035     \em     EFAULT - dest is not 32-byte aligned \n
02036     \em     EIO - I/O error
02037
02038     \see    pvr_dma_modes
02039 */
02040 int pvr_dma_transfer(void *src, uintptr_t dest, size_t count, int type,
02041                      int block, pvr_dma_callback_t callback, void *cbdata);
02042
02043 /** \defgroup pvr_dma_modes         Transfer modes with PVR DMA
02044     @{
02045 */
02046 #define PVR_DMA_VRAM64  0   /**< \brief Transfer to VRAM in interleaved mode */
02047 #define PVR_DMA_VRAM32  1   /**< \brief Transfer to VRAM in linear mode */
02048 #define PVR_DMA_TA      2   /**< \brief Transfer to the tile accelerator */
02049 #define PVR_DMA_YUV     3   /**< \brief Transfer to the YUV converter */
02050 /** @} */
02051
02052 /** \brief  Load a texture using PVR DMA.
02053
02054     This is essentially a convenience wrapper for pvr_dma_transfer(), so all
02055     notes that apply to it also apply here.
02056
02057     \param  src             Where to copy from. Must be 32-byte aligned.
02058     \param  dest            Where to copy to. Must be 32-byte aligned.
02059     \param  count           The number of bytes to copy. Must be a multiple of
02060                             32.
02061     \param  block           Non-zero if you want the function to block until the
02062                             DMA completes.
02063     \param  callback        A function to call upon completion of the DMA.
02064     \param  cbdata          Data to pass to the callback function.
02065     \retval 0               On success.
02066     \retval -1              On failure. Sets errno as appropriate.
02067
02068     \par    Error Conditions:
02069     \em     EINPROGRESS - DMA already in progress \n
02070     \em     EFAULT - dest is not 32-byte aligned \n
02071     \em     EIO - I/O error
02072 */
02073 int pvr_txr_load_dma(void *src, pvr_ptr_t dest, size_t count, int block,
02074                      pvr_dma_callback_t callback, void *cbdata);
02075
02076 /** \brief  Load vertex data to the TA using PVR DMA.
02077
02078     This is essentially a convenience wrapper for pvr_dma_transfer(), so all
02079     notes that apply to it also apply here.
02080
02081     \param  src             Where to copy from. Must be 32-byte aligned.
02082     \param  count           The number of bytes to copy. Must be a multiple of
02083                             32.
02084     \param  block           Non-zero if you want the function to block until the
02085                             DMA completes.
02086     \param  callback        A function to call upon completion of the DMA.
02087     \param  cbdata          Data to pass to the callback function.
02088     \retval 0               On success.
02089     \retval -1              On failure. Sets errno as appropriate.
02090
02091     \par    Error Conditions:
02092     \em     EINPROGRESS - DMA already in progress \n
02093     \em     EFAULT - dest is not 32-byte aligned \n
02094     \em     EIO - I/O error
02095  */
02096 int pvr_dma_load_ta(void *src, size_t count, int block,
02097                     pvr_dma_callback_t callback, void *cbdata);
02098
02099 /** \brief  Load yuv data to the YUV converter using PVR DMA.
02100
02101     This is essentially a convenience wrapper for pvr_dma_transfer(), so all
02102     notes that apply to it also apply here.
02103
02104     \param  src             Where to copy from. Must be 32-byte aligned.
02105     \param  count           The number of bytes to copy. Must be a multiple of
02106                             32.
02107     \param  block           Non-zero if you want the function to block until the
02108                             DMA completes.
02109     \param  callback        A function to call upon completion of the DMA.
02110     \param  cbdata          Data to pass to the callback function.
02111     \retval 0               On success.
02112     \retval -1              On failure. Sets errno as appropriate.
02113
02114     \par    Error Conditions:
```

```
02115     \em     EINPROGRESS - DMA already in progress \n
02116     \em     EFAULT - dest is not 32-byte aligned \n
02117     \em     EIO - I/O error
02118 */
02119 int pvr_dma_yuv_conv(void *src, size_t count, int block,
02120                      pvr_dma_callback_t callback, void *cbdata);
02121
02122 /** \brief  Is PVR DMA is inactive?
02123     \return                 Non-zero if there is no PVR DMA active, thus a DMA
02124                             can begin or 0 if there is an active DMA.
02125 */
02126 int pvr_dma_ready(void);
02127
02128 /** \brief  Initialize PVR DMA. */
02129 void pvr_dma_init(void);
02130
02131 /** \brief  Shut down PVR DMA. */
02132 void pvr_dma_shutdown(void);
02133
02134 /*****************************************************************/
02135
02136
02137 __END_DECLS
02138
02139 #endif
```

9.228 kernel/arch/dreamcast/include/dc/scif.h File Reference

Serial port functionality.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <kos/dbgio.h>
```

Functions

- void scif_set_parameters (int baud, int fifo)

  Set serial parameters.
- int scif_set_irq_usage (int on)

  Enable or disable SCIF IRQ usage.
- int scif_detected (void)

  Is the SCIF port detected? Of course it is!
- int scif_init (void)

  Initialize the SCIF port.
- int scif_shutdown (void)

  Shutdown the SCIF port.
- int scif_read (void)

  Read a single character from the SCIF port.
- int scif_write (int c)

  Write a single character to the SCIF port.
- int scif_flush (void)

  Flush any FIFO'd bytes out of the buffer.
- int scif_write_buffer (const uint8 *data, int len, int xlat)

  Write a whole buffer of data to the SCIF port.
- int scif_read_buffer (uint8 *data, int len)

Read a buffer of data from the SCIF port.

- int scif_spi_init (void)

  Initialize the SCIF port for use of an SPI peripheral.
- int scif_spi_shutdown (void)

  Shut down SPI card support over the SCIF port.
- void scif_spi_set_cs (int v)

  Set or clear the SPI /CS line.
- uint8 scif_spi_rw_byte (uint8 b)

  Read and write one byte from the SPI port.
- uint8 scif_spi_slow_rw_byte (uint8 b)

  Read and write one byte from the SPI device, slowly.
- void scif_spi_write_byte (uint8 b)

  Write a byte to the SPI device.
- uint8 scif_spi_read_byte (void)

  Read a byte from the SPI device.
- void scif_spi_read_data (uint8 *buffer, size_t len)

  Read a data from the SPI device.

Variables

- dbgio_handler_t dbgio_scif

  SCIF debug I/O handler. Do not modify!

9.228.1 Detailed Description

Serial port functionality.

This file deals with raw access to the serial port on the Dreamcast.

Author

Megan Potter

Lawrence Sebald

Ruslan Rostovtsev

9.228.2 Function Documentation

scif_detected()

```
int scif_detected (
            void  )
```

Is the SCIF port detected? Of course it is!

Returns

1

scif_flush()

```
int scif_flush (
            void  )
```

Flush any FIFO'd bytes out of the buffer.

This function sends any bytes that have been queued up for transmission but have not left yet in FIFO mode.

Return values

0	On success.
-1	If the SCIF port is disabled (errno set to EIO).

scif_init()

```
int scif_init (
            void  )
```

Initialize the SCIF port.

This function initializes the SCIF port to a sane state. If dcload-serial is in use, this is effectively a no-op.

Return values

0	On success (no error conditions defined).

scif_read()

```
int scif_read (
            void  )
```

Read a single character from the SCIF port.

Returns

The character read if one is available, otherwise -1 and errno is set to EAGAIN.

scif_read_buffer()

```
int scif_read_buffer (
            uint8 * data,
            int len )
```

Read a buffer of data from the SCIF port.

This function reads a whole buffer of data from the SCIF port, blocking until it has been filled.

Parameters

data	The buffer to read into.
len	The number of bytes to read.

Returns

The number of bytes read on success, -1 on error.

scif_set_irq_usage()

```
int scif_set_irq_usage (
            int on )
```

Enable or disable SCIF IRQ usage.

Parameters

on	1 to enable IRQ usage, 0 for polled I/O.

Return values

0	On success (no error conditions defined).

scif_set_parameters()

```
void scif_set_parameters (
            int baud,
            int fifo )
```

Set serial parameters.

Parameters

baud	The bitrate to set.
fifo	1 to enable FIFO mode.

scif_shutdown()

```
int scif_shutdown (
            void  )
```

Shutdown the SCIF port.

This function disables SCIF IRQs, if they were enabled and cleans up.

Return values

0	On success (no error conditions defined).

scif_spi_init()

```
int scif_spi_init (
            void  )
```

Initialize the SCIF port for use of an SPI peripheral.

This function initializes the SCIF port for accessing the an SPI peripheral that has been connected to the serial port. The design of the SCIF->SPI wiring follows the wiring of the SD card adapter which is (at least now) somewhat commonly available online and is the same as the one designed by jj1odm.

Return values

0	On success.
-1	On error (if dcload-serial is detected).

scif_spi_read_byte()

```
uint8 scif_spi_read_byte (
            void  )
```

Read a byte from the SPI device.

This function reads a byte from the SPI device, one bit at a time. Timing is similar to (but slightly faster than) the scif_spi_rw_byte() function.

Returns

The byte returned from the device.

scif_spi_read_data()

```
void scif_spi_read_data (
            uint8 * buffer,
            size_t len )
```

Read a data from the SPI device.

This function reads data from the SPI device. If the buffer is aligned and len is divisible by 4, optimizations are applied.

Parameters

buffer	Buffer to store read data into.
len	Number of bytes to read from the device.

scif_spi_rw_byte()

```
uint8 scif_spi_rw_byte (
            uint8 b )
```

Read and write one byte from the SPI port.

This function writes one byte and reads one back from the SPI device simultaneously.

Parameters

b	The byte to write out to the port.

Returns

The byte returned from the card.

scif_spi_set_cs()

```
void scif_spi_set_cs (
            int v )
```

Set or clear the SPI /CS line.

This function sets or clears the /CS line (connected to the RTS line of the SCIF port).

Parameters

v	Non-zero to output 1 on the line, zero to output 0.

scif_spi_shutdown()

```
int scif_spi_shutdown (
            void  )
```

Shut down SPI card support over the SCIF port.

This function shuts down SPI support on the SCIF port. If you want to get regular usage of the port back, you must call scif_init() after shutting down SPI support.

Return values

0	On success (no errors defined).

scif_spi_slow_rw_byte()

```
uint8 scif_spi_slow_rw_byte (
            uint8 b )
```

Read and write one byte from the SPI device, slowly.

This function does the same thing as the scif_sd_rw_byte() function, but with a 1.5usec delay between asserting the CLK line and reading back the bit and a 1.5usec delay between clearing the CLK line and writing the next bit out.

This ends up working out to a clock of about 333khz, or so.

Parameters

b	The byte to write out to the port.

Returns

The byte returned from the card.

scif_spi_write_byte()

```
void scif_spi_write_byte (
            uint8 b )
```

Write a byte to the SPI device.

This function writes out the specified byte to the SPI device, one bit at a time. The timing follows that of the scif_spi_rw_byte() function.

Parameters

b	The byte to write out to the port.

scif_write()

```
int scif_write (
            int c )
```

Write a single character to the SCIF port.

Parameters

c	The character to write (only the low 8-bits are written).

Return values

1	On success.
-1	If the SCIF port is disabled (errno set to EIO).

scif_write_buffer()

```
int scif_write_buffer (
            const uint8 * data,
            int len,
            int xlat )
```

Write a whole buffer of data to the SCIF port.

This function writes a whole buffer of data to the SCIF port, optionally making all newlines into carriage return + newline pairs.

Parameters

data	The buffer to write.
len	The length of the buffer, in bytes.
xlat	If set to 1, all newlines will be written as CRLF.

Returns

The number of bytes written on success, -1 on error.

9.228.3 Variable Documentation

dbgio_scif

```
dbgio_handler_t dbgio_scif  [extern]
```

SCIF debug I/O handler. Do not modify!

9.229 scif.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/scif.h
```

```
00004    Copyright (C) 2000,2001,2004 Megan Potter
00005    Copyright (C) 2012 Lawrence Sebald
00006    Copyright (C) 2023 Ruslan Rostovtsev
00007
00008 */
00009
00010 /** \file   dc/scif.h
00011     \brief  Serial port functionality.
00012
00013     This file deals with raw access to the serial port on the Dreamcast.
00014
00015     \author Megan Potter
00016     \author Lawrence Sebald
00017     \author Ruslan Rostovtsev
00018 */
00019
00020 #ifndef __DC_SCIF_H
00021 #define __DC_SCIF_H
00022
00023 #include <sys/cdefs.h>
00024 __BEGIN_DECLS
00025
00026 #include <arch/types.h>
00027 #include <kos/dbgio.h>
00028
00029 /** \brief  Set serial parameters.
00030     \param  baud            The bitrate to set.
00031     \param  fifo            1 to enable FIFO mode.
00032 */
00033 void scif_set_parameters(int baud, int fifo);
00034
00035 // The rest of these are the standard dbgio interface.
00036
00037 /** \brief  Enable or disable SCIF IRQ usage.
00038     \param  on              1 to enable IRQ usage, 0 for polled I/O.
00039     \retval 0               On success (no error conditions defined).
00040 */
00041 int scif_set_irq_usage(int on);
00042
00043 /** \brief  Is the SCIF port detected? Of course it is!
00044     \return                 1
00045 */
00046 int scif_detected(void);
00047
00048 /** \brief  Initialize the SCIF port.
00049
00050     This function initializes the SCIF port to a sane state. If dcload-serial is
00051     in use, this is effectively a no-op.
00052
00053     \retval 0               On success (no error conditions defined).
00054 */
00055 int scif_init(void);
00056
00057 /** \brief  Shutdown the SCIF port.
00058
00059     This function disables SCIF IRQs, if they were enabled and cleans up.
00060
00061     \retval 0               On success (no error conditions defined).
00062 */
00063 int scif_shutdown(void);
00064
00065 /** \brief  Read a single character from the SCIF port.
00066     \return                 The character read if one is available, otherwise -1
00067                             and errno is set to EAGAIN.
00068 */
00069 int scif_read(void);
00070
00071 /** \brief  Write a single character to the SCIF port.
00072     \param  c               The character to write (only the low 8-bits are
00073                             written).
00074     \retval 1               On success.
00075     \retval -1              If the SCIF port is disabled (errno set to EIO).
00076 */
00077 int scif_write(int c);
00078
00079 /** \brief  Flush any FIFO'd bytes out of the buffer.
00080
00081     This function sends any bytes that have been queued up for transmission but
00082     have not left yet in FIFO mode.
00083
00084     \retval 0               On success.
```

```
00085     \retval -1              If the SCIF port is disabled (errno set to EIO).
00086 */
00087 int scif_flush(void);
00088
00089 /** \brief  Write a whole buffer of data to the SCIF port.
00090
00091     This function writes a whole buffer of data to the SCIF port, optionally
00092     making all newlines into carriage return + newline pairs.
00093
00094     \param  data            The buffer to write.
00095     \param  len             The length of the buffer, in bytes.
00096     \param  xlat            If set to 1, all newlines will be written as CRLF.
00097     \return                 The number of bytes written on success, -1 on error.
00098 */
00099 int scif_write_buffer(const uint8 *data, int len, int xlat);
00100
00101 /** \brief  Read a buffer of data from the SCIF port.
00102
00103     This function reads a whole buffer of data from the SCIF port, blocking
00104     until it has been filled.
00105
00106     \param  data            The buffer to read into.
00107     \param  len             The number of bytes to read.
00108     \return                 The number of bytes read on success, -1 on error.
00109 */
00110 int scif_read_buffer(uint8 *data, int len);
00111
00112 /** \brief  SCIF debug I/O handler. Do not modify! */
00113 extern dbgio_handler_t dbgio_scif;
00114
00115 /* Low-level SPI related functionality below here... */
00116 /** \brief  Initialize the SCIF port for use of an SPI peripheral.
00117
00118     This function initializes the SCIF port for accessing the an SPI peripheral
00119     that has been connected to the serial port. The design of the SCIF->SPI
00120     wiring follows the wiring of the SD card adapter which is (at least now)
00121     somewhat commonly available online and is the same as the one designed by
00122     jj1odm.
00123
00124     \retval 0               On success.
00125     \retval -1              On error (if dcload-serial is detected).
00126 */
00127 int scif_spi_init(void);
00128
00129 /** \brief  Shut down SPI card support over the SCIF port.
00130
00131     This function shuts down SPI support on the SCIF port. If you want to get
00132     regular usage of the port back, you must call scif_init() after shutting
00133     down SPI support.
00134
00135     \retval 0               On success (no errors defined).
00136 */
00137 int scif_spi_shutdown(void);
00138
00139 /** \brief  Set or clear the SPI /CS line.
00140
00141     This function sets or clears the /CS line (connected to the RTS line of the
00142     SCIF port).
00143
00144     \param  v               Non-zero to output 1 on the line, zero to output 0.
00145 */
00146 void scif_spi_set_cs(int v);
00147
00148 /** \brief  Read and write one byte from the SPI port.
00149
00150     This function writes one byte and reads one back from the SPI device
00151     simultaneously.
00152
00153     \param  b               The byte to write out to the port.
00154     \return                 The byte returned from the card.
00155 */
00156 uint8 scif_spi_rw_byte(uint8 b);
00157
00158 /** \brief  Read and write one byte from the SPI device, slowly.
00159
00160     This function does the same thing as the scif_sd_rw_byte() function, but
00161     with a 1.5usec delay between asserting the CLK line and reading back the bit
00162     and a 1.5usec delay between clearing the CLK line and writing the next bit
00163     out.
00164
00165     This ends up working out to a clock of about 333khz, or so.
```

```
00166
00167     \param  b               The byte to write out to the port.
00168     \return                 The byte returned from the card.
00169 */
00170 uint8 scif_spi_slow_rw_byte(uint8 b);
00171
00172
00173 /** \brief  Write a byte to the SPI device.
00174
00175     This function writes out the specified byte to the SPI device, one bit at a
00176     time. The timing follows that of the scif_spi_rw_byte() function.
00177
00178     \param  b               The byte to write out to the port.
00179 */
00180 void scif_spi_write_byte(uint8 b);
00181
00182 /** \brief  Read a byte from the SPI device.
00183
00184     This function reads a byte from the SPI device, one bit at a time. Timing
00185     is similar to (but slightly faster than) the scif_spi_rw_byte() function.
00186
00187     \return                 The byte returned from the device.
00188 */
00189 uint8 scif_spi_read_byte(void);
00190
00191 /** \brief  Read a data from the SPI device.
00192
00193     This function reads data from the SPI device. If the buffer is aligned and
00194     len is divisible by 4, optimizations are applied.
00195
00196     \param  buffer          Buffer to store read data into.
00197     \param  len             Number of bytes to read from the device.
00198 */
00199 void scif_spi_read_data(uint8 *buffer, size_t len);
00200
00201 __END_DECLS
00202
00203 #endif  /* __DC_SCIF_H */
```

9.230 kernel/arch/dreamcast/include/dc/sd.h File Reference

Block-level access to an SD card attached to the SCIF port.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <kos/blockdev.h>
```

Functions

- uint8 sd_crc7 (const uint8 *data, int size, uint8 crc)

  Calculate a SD/MMC-style CRC over a block of data.
- int sd_init (void)

  Initialize the SD card for use.
- int sd_shutdown (void)

  Shut down SD card support.
- int sd_read_blocks (uint32 block, size_t count, uint8 *buf)

  Read one or more blocks from the SD card.
- int sd_write_blocks (uint32 block, size_t count, const uint8 *buf)

  Write one or more blocks to the SD card.
- uint64 sd_get_size (void)

  Retrieve the size of the SD card.
- int sd_blockdev_for_partition (int partition, kos_blockdev_t *rv, uint8 *partition_type)

  Get a block device for a given partition on the SD card.

9.230.1 Detailed Description

Block-level access to an SD card attached to the SCIF port.

This file contains the interface to working with the SD card reader that was designed by jj1odm. The SD card reader itself connects to the SCIF port and uses it basically as a simple SPI bus.

For reference, all I/O through this code should be done in the order of SD card blocks (which are 512 bytes a piece). Also, this should adequately support SD and SDHC cards (and possibly SDXC, but I don't have any of them to try out).

This code doesn't directly implement any filesystems on top of the SD card, but rather provides you with direct block-level access. This probably will not be useful to most people in its current form (without a filesystem), but this will provide you with all of the building blocks you should need to actually make it work for you.

Due to the patent-encumbered nature of certain parts of the FAT32 filesystem, that filesystem will likely never be supported in KOS proper (unless, of course, people are still using KOS after those patents expire). I'm not going to encourage anyone to violate Microsoft's patents on FAT32 and I'm not going to be the enabler for anyone to do so either. So, if you want FAT32, you're on your own.

Author

Lawrence Sebald

See also

dc/scif.h

9.230.2 Function Documentation

sd_blockdev_for_partition()

```
int sd_blockdev_for_partition (
            int partition,
            kos_blockdev_t * rv,
            uint8 * partition_type )
```

Get a block device for a given partition on the SD card.

This function creates a block device descriptor for the given partition on the attached SD card. This block device is used to interface with various filesystems on the device.

Parameters

partition	The partition number (0-3) to use.
rv	Used to return the block device. Must be non-NULL.
partition_type	Used to return the partition type. Must be non-NULL.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

ENXIO - SD card support was not initialized
EIO - an I/O error occurred in reading data
EINVAL - invalid partition number was given
EFAULT - rv or partition_type was NULL
ENOENT - no MBR found
ENOENT - no partition at the specified position
ENOMEM - out of memory

Note

This interface currently only supports MBR-formatted SD cards. There is currently no support for GPT partition tables.

sd_crc7()

```
uint8 sd_crc7 (
            const uint8 * data,
            int size,
            uint8 crc )
```

Calculate a SD/MMC-style CRC over a block of data.

This function calculates a 7-bit CRC over a given block of data. The polynomial of this CRC is x^7 + x^3 + 1. The CRC is shifted into the upper 7 bits of the return value, so bit 0 should always be 0.

Parameters

data	The block of data to calculate the CRC over.
size	The number of bytes in the block of data.
crc	The starting value of the calculation. If you're passing in a full block, this will probably be 0.

Returns

The calculated CRC.

sd_get_size()

```
uint64 sd_get_size (
            void  )
```

Retrieve the size of the SD card.

This function reads the size of the SD card from the card's CSD register. This is the raw size of the card, not its formatted capacity. To get the number of blocks from this, divide by 512.

Returns

On succes, the raw size of the SD card in bytes. On error, (uint64)-1.

Error Conditions:

EIO - an I/O error occurred in reading data
ENXIO - SD card support was not initialized

sd_init()

```
int sd_init (
            void  )
```

Initialize the SD card for use.

This function initializes the SD card for first use. This includes all steps of the basic initialization sequence for SPI mode, as documented in the SD card spec and at http://elm-chan.org/docs/mmc/mmc_e.html . This also will call scif_sd_init() for you, so you don't have to worry about that ahead of time.

Return values

0	On success.
-1	On failure. This could indicate any number of problems, but probably means that no SD card was detected.

sd_read_blocks()

```
int sd_read_blocks (
            uint32 block,
            size_t count,
            uint8 * buf )
```

Read one or more blocks from the SD card.

This function reads the specified number of blocks from the SD card from the beginning block specified into the buffer passed in. It is your responsibility to allocate the buffer properly for the number of bytes that is to be read (512 * the number of blocks requested).

Parameters

block	The starting block number to read from.
count	The number of 512 byte blocks of data to read.
buf	The buffer to read into.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

EIO - an I/O error occurred in reading data
ENXIO - SD card support was not initialized

sd_shutdown()

```
int sd_shutdown (
            void  )
```

Shut down SD card support.

This function shuts down SD card support, and cleans up anything that the sd_init() function set up.

Return values

0	On success.
-1	On failure. The only currently defined error is if the card was never initialized to start with.

sd_write_blocks()

```
int sd_write_blocks (
            uint32 block,
            size_t count,
            const uint8 * buf )
```

Write one or more blocks to the SD card.

This function writes the specified number of blocks to the SD card at the beginning block specified from the buffer passed in. Each block is 512 bytes in length, and you must write at least one block at a time. You cannot write partial blocks.

If this function returns an error, you have quite possibly corrupted something on the card or have a damaged card in general (unless errno is ENXIO).

Parameters

block	The starting block number to write to.
count	The number of 512 byte blocks of data to write.
buf	The buffer to write from.

Return values

0	On success.
-1	On error, errno will be set as appropriate.

Error Conditions:

EIO - an I/O error occurred in reading data
ENXIO - SD card support was not initialized

9.231 sd.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/sd.h
00004    Copyright (C) 2012 Lawrence Sebald
00005 */
00006
00007 #ifndef __DC_SD_H
00008 #define __DC_SD_H
00009
00010 #include <sys/cdefs.h>
00011 __BEGIN_DECLS
00012
00013 #include <arch/types.h>
00014 #include <kos/blockdev.h>
00015
00016 /** \file   dc/sd.h
00017     \brief  Block-level access to an SD card attached to the SCIF port.
00018
00019     This file contains the interface to working with the SD card reader that was
00020     designed by jj1odm. The SD card reader itself connects to the SCIF port and
00021     uses it basically as a simple SPI bus.
00022
00023     For reference, all I/O through this code should be done in the order of SD
00024     card blocks (which are 512 bytes a piece). Also, this should adequately
00025     support SD and SDHC cards (and possibly SDXC, but I don't have any of them
00026     to try out).
00027
00028     This code doesn't directly implement any filesystems on top of the SD card,
00029     but rather provides you with direct block-level access. This probably will
00030     not be useful to most people in its current form (without a filesystem), but
00031     this will provide you with all of the building blocks you should need to
00032     actually make it work for you.
00033
00034     Due to the patent-encumbered nature of certain parts of the FAT32
00035     filesystem, that filesystem will likely never be supported in KOS proper
00036     (unless, of course, people are still using KOS after those patents expire).
00037     I'm not going to encourage anyone to violate Microsoft's patents on FAT32
00038     and I'm not going to be the enabler for anyone to do so either. So, if you
00039     want FAT32, you're on your own.
00040
00041     \author Lawrence Sebald
00042     \see    dc/scif.h
00043 */
00044
00045 /** \brief  Calculate a SD/MMC-style CRC over a block of data.
00046
00047     This function calculates a 7-bit CRC over a given block of data. The
00048     polynomial of this CRC is x^7 + x^3 + 1. The CRC is shifted into the upper
00049     7 bits of the return value, so bit 0 should always be 0.
00050
00051     \param  data            The block of data to calculate the CRC over.
00052     \param  size            The number of bytes in the block of data.
00053     \param  crc             The starting value of the calculation. If you're
00054                             passing in a full block, this will probably be 0.
00055     \return                 The calculated CRC.
00056 */
00057 uint8 sd_crc7(const uint8 *data, int size, uint8 crc);
```

```
00058
00059 /** \brief  Initialize the SD card for use.
00060
00061     This function initializes the SD card for first use. This includes all steps
00062     of the basic initialization sequence for SPI mode, as documented in the SD
00063     card spec and at http://elm-chan.org/docs/mmc/mmc_e.html . This also will
00064     call scif_sd_init() for you, so you don't have to worry about that ahead of
00065     time.
00066
00067     \retval 0               On success.
00068     \retval -1              On failure. This could indicate any number of
00069                             problems, but probably means that no SD card was
00070                             detected.
00071 */
00072 int sd_init(void);
00073
00074 /** \brief  Shut down SD card support.
00075
00076     This function shuts down SD card support, and cleans up anything that the
00077     sd_init() function set up.
00078
00079     \retval 0               On success.
00080     \retval -1              On failure. The only currently defined error is if
00081                             the card was never initialized to start with.
00082 */
00083 int sd_shutdown(void);
00084
00085 /** \brief  Read one or more blocks from the SD card.
00086
00087     This function reads the specified number of blocks from the SD card from the
00088     beginning block specified into the buffer passed in. It is your
00089     responsibility to allocate the buffer properly for the number of bytes that
00090     is to be read (512 * the number of blocks requested).
00091
00092     \param  block           The starting block number to read from.
00093     \param  count           The number of 512 byte blocks of data to read.
00094     \param  buf             The buffer to read into.
00095     \retval 0               On success.
00096     \retval -1              On error, errno will be set as appropriate.
00097
00098     \par    Error Conditions:
00099     \em     EIO - an I/O error occurred in reading data \n
00100     \em     ENXIO - SD card support was not initialized
00101 */
00102 int sd_read_blocks(uint32 block, size_t count, uint8 *buf);
00103
00104 /** \brief  Write one or more blocks to the SD card.
00105
00106     This function writes the specified number of blocks to the SD card at the
00107     beginning block specified from the buffer passed in. Each block is 512 bytes
00108     in length, and you must write at least one block at a time. You cannot write
00109     partial blocks.
00110
00111     If this function returns an error, you have quite possibly corrupted
00112     something on the card or have a damaged card in general (unless errno is
00113     ENXIO).
00114
00115     \param  block           The starting block number to write to.
00116     \param  count           The number of 512 byte blocks of data to write.
00117     \param  buf             The buffer to write from.
00118     \retval 0               On success.
00119     \retval -1              On error, errno will be set as appropriate.
00120
00121     \par    Error Conditions:
00122     \em     EIO - an I/O error occurred in reading data \n
00123     \em     ENXIO - SD card support was not initialized
00124 */
00125 int sd_write_blocks(uint32 block, size_t count, const uint8 *buf);
00126
00127 /** \brief  Retrieve the size of the SD card.
00128
00129     This function reads the size of the SD card from the card's CSD register.
00130     This is the raw size of the card, not its formatted capacity. To get the
00131     number of blocks from this, divide by 512.
00132
00133     \return                 On succes, the raw size of the SD card in bytes. On
00134                             error, (uint64)-1.
00135
00136     \par    Error Conditions:
00137     \em     EIO - an I/O error occurred in reading data \n
00138     \em     ENXIO - SD card support was not initialized
```

```
00139 */
00140 uint64 sd_get_size(void);
00141
00142 /** \brief  Get a block device for a given partition on the SD card.
00143
00144     This function creates a block device descriptor for the given partition on
00145     the attached SD card. This block device is used to interface with various
00146     filesystems on the device.
00147
00148     \param  partition       The partition number (0-3) to use.
00149     \param  rv              Used to return the block device. Must be non-NULL.
00150     \param  partition_type  Used to return the partition type. Must be non-NULL.
00151     \retval 0               On success.
00152     \retval -1              On error, errno will be set as appropriate.
00153
00154     \par    Error Conditions:
00155     \em     ENXIO - SD card support was not initialized \n
00156     \em     EIO - an I/O error occurred in reading data \n
00157     \em     EINVAL - invalid partition number was given \n
00158     \em     EFAULT - rv or partition_type was NULL \n
00159     \em     ENOENT - no MBR found \n
00160     \em     ENOENT - no partition at the specified position \n
00161     \em     ENOMEM - out of memory
00162
00163     \note   This interface currently only supports MBR-formatted SD cards. There
00164             is currently no support for GPT partition tables.
00165 */
00166 int sd_blockdev_for_partition(int partition, kos_blockdev_t *rv,
00167                               uint8 *partition_type);
00168
00169 __END_DECLS
00170 #endif /* !__DC_SD_H */
```

9.232 kernel/arch/dreamcast/include/dc/sound/aica_comm.h File Reference

Data Structures

- struct aica_queue_t
- struct aica_cmd_t
- struct aica_channel_t

Macros

- #define AICA_CMD_MAX_SIZE 256
- #define AICA_CMDSTR_CHANNEL(T, CMDR, CHANR)
- #define AICA_CMDSTR_CHANNEL_SIZE ((sizeof(aica_cmd_t) + sizeof(aica_channel_t))/4)
- #define AICA_CMD_NONE 0x00000000 /* No command (dummy packet) */
- #define AICA_CMD_PING 0x00000001 /* Check for signs of life */
- #define AICA_CMD_CHAN 0x00000002 /* Perform a wavetable action */
- #define AICA_CMD_SYNC_CLOCK 0x00000003 /* Reset the millisecond clock */
- #define AICA_RESP_NONE 0x00000000
- #define AICA_RESP_PONG 0x00000001 /* Response to CMD_PING */
- #define AICA_RESP_DBGPRINT 0x00000002 /* Entire payload is a null-terminated string */
- #define AICA_CH_CMD_MASK 0x0000000f
- #define AICA_CH_CMD_NONE 0x00000000
- #define AICA_CH_CMD_START 0x00000001
- #define AICA_CH_CMD_STOP 0x00000002
- #define AICA_CH_CMD_UPDATE 0x00000003
- #define AICA_CH_START_MASK 0x00300000
- #define AICA_CH_START_DELAY 0x00100000 /* Set params, but delay key-on */

- #define AICA_CH_START_SYNC 0x00200000 /* Set key-on for all selected channels */
- #define AICA_CH_UPDATE_MASK 0x000ff000
- #define AICA_CH_UPDATE_SET_FREQ 0x00001000 /* frequency */
- #define AICA_CH_UPDATE_SET_VOL 0x00002000 /* volume */
- #define AICA_CH_UPDATE_SET_PAN 0x00004000 /* panning */
- #define AICA_SM_16BIT 0 /* Linear PCM 16-bit */
- #define AICA_SM_8BIT 1 /* Linear PCM 8-bit */
- #define AICA_SM_ADPCM 2 /* Yamaha ADPCM 4-bit */
- #define AICA_SM_ADPCM_LS 3 /* Long stream ADPCM 4-bit */

Typedefs

- typedef unsigned long uint8
- typedef unsigned long uint32

9.232.1 Macro Definition Documentation

AICA_CH_CMD_MASK

```
#define AICA_CH_CMD_MASK 0x0000000f
```

AICA_CH_CMD_NONE

```
#define AICA_CH_CMD_NONE 0x00000000
```

AICA_CH_CMD_START

```
#define AICA_CH_CMD_START 0x00000001
```

AICA_CH_CMD_STOP

```
#define AICA_CH_CMD_STOP 0x00000002
```

AICA_CH_CMD_UPDATE

```
#define AICA_CH_CMD_UPDATE 0x00000003
```

AICA_CH_START_DELAY

```
#define AICA_CH_START_DELAY 0x00100000 /* Set params, but delay key-on */
```

AICA_CH_START_MASK

```
#define AICA_CH_START_MASK 0x00300000
```

AICA_CH_START_SYNC

```
#define AICA_CH_START_SYNC 0x00200000 /* Set key-on for all selected channels */
```

AICA_CH_UPDATE_MASK

```
#define AICA_CH_UPDATE_MASK 0x000ff000
```

AICA_CH_UPDATE_SET_FREQ

```
#define AICA_CH_UPDATE_SET_FREQ 0x00001000 /* frequency */
```

AICA_CH_UPDATE_SET_PAN

```
#define AICA_CH_UPDATE_SET_PAN 0x00004000 /* panning */
```

AICA_CH_UPDATE_SET_VOL

```
#define AICA_CH_UPDATE_SET_VOL 0x00002000 /* volume */
```

AICA_CMD_CHAN

```
#define AICA_CMD_CHAN 0x00000002 /* Perform a wavetable action */
```

AICA_CMD_MAX_SIZE

```
#define AICA_CMD_MAX_SIZE 256
```

AICA_CMD_NONE

```
#define AICA_CMD_NONE 0x00000000 /* No command (dummy packet) */
```

AICA_CMD_PING

```
#define AICA_CMD_PING 0x00000001 /* Check for signs of life */
```

AICA_CMD_SYNC_CLOCK

```
#define AICA_CMD_SYNC_CLOCK 0x00000003 /* Reset the millisecond clock */
```

AICA_CMDSTR_CHANNEL

```
#define AICA_CMDSTR_CHANNEL(
              T,
              CMDR,
              CHANR )
```

Value:

```
    uint8   T[sizeof(aica_cmd_t) + sizeof(aica_channel_t)]; \
    aica_cmd_t  * CMDR = (aica_cmd_t *)T; \
    aica_channel_t  * CHANR = (aica_channel_t *)(CMDR->cmd_data);
```

AICA_CMDSTR_CHANNEL_SIZE

```
#define AICA_CMDSTR_CHANNEL_SIZE ((sizeof(aica_cmd_t) + sizeof(aica_channel_t))/4)
```

AICA_RESP_DBGPRINT

```
#define AICA_RESP_DBGPRINT 0x00000002 /* Entire payload is a null-terminated string */
```

AICA_RESP_NONE

```
#define AICA_RESP_NONE 0x00000000
```

AICA_RESP_PONG

```
#define AICA_RESP_PONG 0x00000001 /* Response to CMD_PING */
```

AICA_SM_16BIT

```
#define AICA_SM_16BIT 0 /* Linear PCM 16-bit */
```

AICA_SM_8BIT

```
#define AICA_SM_8BIT 1 /* Linear PCM 8-bit */
```

AICA_SM_ADPCM

```
#define AICA_SM_ADPCM 2 /* Yamaha ADPCM 4-bit */
```

AICA_SM_ADPCM_LS

```
#define AICA_SM_ADPCM_LS 3 /* Long stream ADPCM 4-bit */
```

9.232.2 Typedef Documentation

uint32

```
typedef unsigned long uint32
```

uint8

```
typedef unsigned long uint8
```

9.233 aica_comm.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    aica_comm.h
00004    Copyright (C) 2000-2002 Megan Potter
00005    Copyright (C) 2023 Ruslan Rostovtsev
00006
00007    Structure and constant definitions for the SH-4/AICA interface. This file is
00008    included from both the ARM and SH-4 sides of the fence.
00009 */
00010
00011 #ifndef __DC_SOUND_AICA_COMM_H
00012 #define __DC_SOUND_AICA_COMM_H
00013
00014 #ifndef __ARCH_TYPES_H
00015 typedef unsigned long uint8;
00016 typedef unsigned long uint32;
00017 #endif
00018
00019 /* Command queue; one of these for passing data from the SH-4 to the
00020    AICA, and another for the other direction. If a command is written
00021    to the queue and it is longer than the amount of space between the
00022    head point and the queue size, the command will wrap around to
00023    the beginning (i.e., queue commands _can_ be split up). */
00024 typedef struct aica_queue {
00025     uint32      head;       /* Insertion point offset (in bytes) */
00026     uint32      tail;       /* Removal point offset (in bytes) */
00027     uint32      size;       /* Queue size (in bytes) */
00028     uint32      valid;      /* 1 if the queue structs are valid */
00029     uint32      process_ok; /* 1 if it's ok to process the data */
00030     uint32      data;       /* Pointer to queue data buffer */
00031 } aica_queue_t;
00032
00033 /* Command queue struct for commanding the AICA from the SH-4 */
00034 typedef struct aica_cmd {
00035     uint32      size;       /* Command data size in dwords */
00036     uint32      cmd;        /* Command ID */
00037     uint32      timestamp;  /* When to execute the command (0 == now) */
```

```
00038     uint32      cmd_id;     /* Command ID, for cmd/response pairs, or channel id */
00039     uint32      misc[4];    /* Misc Parameters / Padding */
00040     uint8       cmd_data[]; /* Command data */
00041 } aica_cmd_t;
00042
00043 /* Maximum command size -- 256 dwords */
00044 #define AICA_CMD_MAX_SIZE   256
00045
00046 /* This is the cmd_data for AICA_CMD_CHAN. Make this 16 dwords long
00047    for two aica bus queues. */
00048 typedef struct aica_channel {
00049     uint32      cmd;        /* Command ID */
00050     uint32      base;       /* Sample base in RAM */
00051     uint32      type;       /* (8/16bit/ADPCM) */
00052     uint32      length;     /* Sample length */
00053     uint32      loop;       /* Sample looping */
00054     uint32      loopstart;  /* Sample loop start */
00055     uint32      loopend;    /* Sample loop end */
00056     uint32      freq;       /* Frequency */
00057     uint32      vol;        /* Volume 0-255 */
00058     uint32      pan;        /* Pan 0-255 */
00059     uint32      pos;        /* Sample playback pos */
00060     uint32      pad[5];     /* Padding */
00061 } aica_channel_t;
00062
00063 /* Declare an aica_cmd_t big enough to hold an aica_channel_t
00064    using temp name T, aica_cmd_t name CMDR, and aica_channel_t name CHANR */
00065 #define AICA_CMDSTR_CHANNEL(T, CMDR, CHANR) \
00066     uint8   T[sizeof(aica_cmd_t) + sizeof(aica_channel_t)]; \
00067     aica_cmd_t  * CMDR = (aica_cmd_t *)T; \
00068     aica_channel_t  * CHANR = (aica_channel_t *)(CMDR->cmd_data);
00069 #define AICA_CMDSTR_CHANNEL_SIZE    ((sizeof(aica_cmd_t) + sizeof(aica_channel_t))/4)
00070
00071 /* Command values (for aica_cmd_t) */
00072 #define AICA_CMD_NONE       0x00000000  /* No command (dummy packet)    */
00073 #define AICA_CMD_PING       0x00000001  /* Check for signs of life  */
00074 #define AICA_CMD_CHAN       0x00000002  /* Perform a wavetable action   */
00075 #define AICA_CMD_SYNC_CLOCK 0x00000003  /* Reset the millisecond clock  */
00076
00077 /* Response values (for aica_cmd_t) */
00078 #define AICA_RESP_NONE      0x00000000
00079 #define AICA_RESP_PONG      0x00000001  /* Response to CMD_PING             */
00080 #define AICA_RESP_DBGPRINT  0x00000002  /* Entire payload is a null-terminated string   */
00081
00082 /* Command values (for aica_channel_t commands) */
00083 #define AICA_CH_CMD_MASK    0x0000000f
00084
00085 #define AICA_CH_CMD_NONE    0x00000000
00086 #define AICA_CH_CMD_START   0x00000001
00087 #define AICA_CH_CMD_STOP    0x00000002
00088 #define AICA_CH_CMD_UPDATE  0x00000003
00089
00090 /* Start values */
00091 #define AICA_CH_START_MASK  0x00300000
00092
00093 #define AICA_CH_START_DELAY 0x00100000 /* Set params, but delay key-on */
00094 #define AICA_CH_START_SYNC  0x00200000 /* Set key-on for all selected channels */
00095
00096 /* Update values */
00097 #define AICA_CH_UPDATE_MASK 0x000ff000
00098
00099 #define AICA_CH_UPDATE_SET_FREQ 0x00001000 /* frequency     */
00100 #define AICA_CH_UPDATE_SET_VOL  0x00002000 /* volume        */
00101 #define AICA_CH_UPDATE_SET_PAN  0x00004000 /* panning       */
00102
00103 /* Sample types */
00104 #define AICA_SM_16BIT    0 /* Linear PCM 16-bit */
00105 #define AICA_SM_8BIT     1 /* Linear PCM 8-bit */
00106 #define AICA_SM_ADPCM    2 /* Yamaha ADPCM 4-bit */
00107 #define AICA_SM_ADPCM_LS 3 /* Long stream ADPCM 4-bit */
00108
00109 #endif /* !__DC_SOUND_AICA_COMM_H */
```

9.234 kernel/arch/dreamcast/include/dc/sound/sfxmgr.h File Reference

Basic sound effect support.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <stdint.h>
```

Macros

- #define SFXHND_INVALID 0

  Invalid sound effect handle value.

Typedefs

- typedef uint32_t sfxhnd_t

  Sound effect handle type.

Functions

- sfxhnd_t snd_sfx_load (const char *fn)

  Load a sound effect.
- void snd_sfx_unload (sfxhnd_t idx)

  Unload a sound effect.
- void snd_sfx_unload_all (void)

  Unload all loaded sound effects.
- int snd_sfx_play (sfxhnd_t idx, int vol, int pan)

  Play a sound effect.
- int snd_sfx_play_chn (int chn, sfxhnd_t idx, int vol, int pan)

  Play a sound effect on a specific channel.
- void snd_sfx_stop (int chn)

  Stop a single channel of sound.
- void snd_sfx_stop_all (void)

  Stop all channels playing sound effects.
- int snd_sfx_chn_alloc (void)

  Allocate a sound channel for use outside the sound effect system.
- void snd_sfx_chn_free (int chn)

  Free a previously allocated channel.

9.234.1 Detailed Description

Basic sound effect support.

This file contains declarations for doing simple sound effects. This code is only usable for simple WAV files containing either 8-bit or 16-bit samples (stereo or mono) or Yamaha ADPCM (4-bits, stereo or mono). Also, all sounds played in this manner must be at most 65534 samples in length, as this does not handle buffer chaining or anything else complex. For more interesting stuff, you should probably look at the sound stream stuff instead.

Author

Megan Potter

9.234.2 Macro Definition Documentation

SFXHND_INVALID

```
#define SFXHND_INVALID 0
```

Invalid sound effect handle value.

If a sound effect cannot be loaded, this value will be returned as the error condition.

9.234.3 Typedef Documentation

sfxhnd_t

```
typedef uint32_t sfxhnd_t
```

Sound effect handle type.

Each loaded sound effect will be assigned one of these, which is to be used for operations related to the effect, including playing it or unloading it.

9.234.4 Function Documentation

snd_sfx_chn_alloc()

```
int snd_sfx_chn_alloc (
            void  )
```

Allocate a sound channel for use outside the sound effect system.

This function finds and allocates a channel for use for things other than sound effects. This is useful for, for instance, the streaming code.

Returns

The allocated channel on success, -1 on failure.

snd_sfx_chn_free()

```
void snd_sfx_chn_free (
            int chn )
```

Free a previously allocated channel.

This function frees a channel that was allocated with snd_sfx_chn_alloc(), returning it to the pool of available channels for sound effect use.

Parameters

chn	The channel to free.

snd_sfx_load()

```
sfxhnd_t snd_sfx_load (
            const char * fn )
```

Load a sound effect.

This function loads a sound effect from a WAV file and returns a handle to it. The sound effect can be either stereo or mono, and must either be 8-bit or 16-bit uncompressed PCM samples, or 4-bit Yamaha ADPCM.

Warning

The sound effect you are loading must be at most 65534 samples in length.

Parameters

fn	The file to load.

Returns

A handle to the sound effect on success. On error, SFXHND_INVALID is returned.

snd_sfx_play()

```
int snd_sfx_play (
            sfxhnd_t idx,
            int vol,
            int pan )
```

Play a sound effect.

This function plays a loaded sound effect with the specified volume (for both stereo or mono) and panning values (for mono sounds only).

Parameters

idx	The handle to the sound effect to play.
vol	The volume to play at (between 0 and 255).
pan	The panning value of the sound effect. 0 is all the way to the left, 128 is center, 255 is all the way to the right.

Returns

The channel used to play the sound effect (or the left channel in the case of a stereo sound, the right channel will be the next one) on success, or -1 on failure.

snd_sfx_play_chn()

```
int snd_sfx_play_chn (
            int chn,
            sfxhnd_t idx,
            int vol,
            int pan )
```

Play a sound effect on a specific channel.

This function works similar to snd_sfx_play(), but allows you to specify the channel to play on. No error checking is done with regard to the channel, so be sure its safe to play on that channel before trying.

Parameters

chn	The channel to play on (or in the case of stereo, the left channel).
idx	The handle to the sound effect to play.
vol	The volume to play at (between 0 and 255).
pan	The panning value of the sound effect. 0 is all the way to the left, 128 is center, 255 is all the way to the right.

Returns

chn

snd_sfx_stop()

```
void snd_sfx_stop (
            int chn )
```

Stop a single channel of sound.

This function stops the specified channel of sound from playing. It does no checking to make sure that a sound effect is playing on the channel specified, and thus can be used even if you're using the channel for some other purpose than sound effects.

Parameters

chn	The channel to stop.

snd_sfx_stop_all()

```
void snd_sfx_stop_all (
             void  )
```

Stop all channels playing sound effects.

This function stops all channels currently allocated to sound effects from playing. It does not affect channels allocated for use by something other than sound effects..

snd_sfx_unload()

```
void snd_sfx_unload (
             sfxhnd_t idx )
```

Unload a sound effect.

This function unloads a previously loaded sound effect, and frees the memory associated with it.

Parameters

idx	A handle to the sound effect to unload.

snd_sfx_unload_all()

```
void snd_sfx_unload_all (
             void  )
```

Unload all loaded sound effects.

This function unloads all previously loaded sound effect, and frees the memory associated with them.

9.235 sfxmgr.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/sound/sfxmgr.h
00004    Copyright (C) 2002 Megan Potter
00005    Copyright (C) 2023 Ruslan Rostovtsev
00006
00007 */
00008
00009 /** \file   dc/sound/sfxmgr.h
00010     \brief  Basic sound effect support.
00011
00012     This file contains declarations for doing simple sound effects. This code is
00013     only usable for simple WAV files containing either 8-bit or 16-bit samples (stereo
00014     or mono) or Yamaha ADPCM (4-bits, stereo or mono). Also, all sounds played in
00015     this manner must be at most 65534 samples in length, as this does not handle
00016     buffer chaining or anything else complex. For more interesting stuff, you
00017     should probably look at the sound stream stuff instead.
```

```
00018
00019     \author Megan Potter
00020 */
00021
00022 #ifndef __DC_SOUND_SFXMGR_H
00023 #define __DC_SOUND_SFXMGR_H
00024
00025 #include <sys/cdefs.h>
00026 __BEGIN_DECLS
00027
00028 #include <arch/types.h>
00029 #include <stdint.h>
00030
00031 /** \brief  Sound effect handle type.
00032
00033     Each loaded sound effect will be assigned one of these, which is to be used
00034     for operations related to the effect, including playing it or unloading it.
00035 */
00036 typedef uint32_t sfxhnd_t;
00037
00038 /** \brief  Invalid sound effect handle value.
00039
00040     If a sound effect cannot be loaded, this value will be returned as the error
00041     condition.
00042 */
00043 #define SFXHND_INVALID 0
00044
00045 /** \brief  Load a sound effect.
00046
00047     This function loads a sound effect from a WAV file and returns a handle to
00048     it. The sound effect can be either stereo or mono, and must either be 8-bit
00049     or 16-bit uncompressed PCM samples, or 4-bit Yamaha ADPCM.
00050
00051     \warning The sound effect you are loading must be at most 65534 samples
00052     in length.
00053
00054     \param  fn              The file to load.
00055     \return                 A handle to the sound effect on success. On error,
00056                             SFXHND_INVALID is returned.
00057 */
00058 sfxhnd_t snd_sfx_load(const char *fn);
00059
00060 /** \brief  Unload a sound effect.
00061
00062     This function unloads a previously loaded sound effect, and frees the memory
00063     associated with it.
00064
00065     \param  idx             A handle to the sound effect to unload.
00066 */
00067 void snd_sfx_unload(sfxhnd_t idx);
00068
00069 /** \brief  Unload all loaded sound effects.
00070
00071     This function unloads all previously loaded sound effect, and frees the
00072     memory associated with them.
00073 */
00074 void snd_sfx_unload_all(void);
00075
00076 /** \brief  Play a sound effect.
00077
00078     This function plays a loaded sound effect with the specified volume (for
00079     both stereo or mono) and panning values (for mono sounds only).
00080
00081     \param  idx             The handle to the sound effect to play.
00082     \param  vol             The volume to play at (between 0 and 255).
00083     \param  pan             The panning value of the sound effect. 0 is all the
00084                             way to the left, 128 is center, 255 is all the way
00085                             to the right.
00086
00087     \return                 The channel used to play the sound effect (or the
00088                             left channel in the case of a stereo sound, the
00089                             right channel will be the next one) on success, or
00090                             -1 on failure.
00091 */
00092 int snd_sfx_play(sfxhnd_t idx, int vol, int pan);
00093
00094 /** \brief  Play a sound effect on a specific channel.
00095
00096     This function works similar to snd_sfx_play(), but allows you to specify the
00097     channel to play on. No error checking is done with regard to the channel, so
00098     be sure its safe to play on that channel before trying.
```

```
00099
00100     \param  chn             The channel to play on (or in the case of stereo,
00101                             the left channel).
00102     \param  idx             The handle to the sound effect to play.
00103     \param  vol             The volume to play at (between 0 and 255).
00104     \param  pan             The panning value of the sound effect. 0 is all the
00105                             way to the left, 128 is center, 255 is all the way
00106                             to the right.
00107
00108     \return                 chn
00109 */
00110 int snd_sfx_play_chn(int chn, sfxhnd_t idx, int vol, int pan);
00111
00112 /** \brief  Stop a single channel of sound.
00113
00114     This function stops the specified channel of sound from playing. It does no
00115     checking to make sure that a sound effect is playing on the channel
00116     specified, and thus can be used even if you're using the channel for some
00117     other purpose than sound effects.
00118
00119     \param  chn             The channel to stop.
00120 */
00121 void snd_sfx_stop(int chn);
00122
00123 /** \brief  Stop all channels playing sound effects.
00124
00125     This function stops all channels currently allocated to sound effects from
00126     playing. It does not affect channels allocated for use by something other
00127     than sound effects..
00128 */
00129 void snd_sfx_stop_all(void);
00130
00131 /** \brief  Allocate a sound channel for use outside the sound effect system.
00132
00133     This function finds and allocates a channel for use for things other than
00134     sound effects. This is useful for, for instance, the streaming code.
00135
00136     \returns                The allocated channel on success, -1 on failure.
00137 */
00138 int snd_sfx_chn_alloc(void);
00139
00140 /** \brief  Free a previously allocated channel.
00141
00142     This function frees a channel that was allocated with snd_sfx_chn_alloc(),
00143     returning it to the pool of available channels for sound effect use.
00144
00145     \param  chn             The channel to free.
00146 */
00147 void snd_sfx_chn_free(int chn);
00148
00149 __END_DECLS
00150
00151 #endif  /* __DC_SOUND_SFXMGR_H */
00152
```

9.236 kernel/arch/dreamcast/include/dc/sound/sound.h File Reference

Low-level sound support and memory management.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <stdint.h>
```

Functions

- uint32 snd_mem_malloc (size_t size)

  Allocate memory in the SPU RAM pool.

- void snd_mem_free (uint32 addr)

  Free a block of allocated memory in the SPU RAM pool.
- uint32 snd_mem_available (void)

  Get the size of the largest allocateable block in the SPU RAM pool.
- int snd_mem_init (uint32 reserve)

  Reinitialize the SPU RAM pool.
- void snd_mem_shutdown (void)

  Shutdown the SPU RAM allocator.
- int snd_init (void)

  Initialize the sound system.
- void snd_shutdown (void)

  Shut down the sound system.
- int snd_sh4_to_aica (void *packet, uint32 size)

  Copy a request packet to the AICA queue.
- void snd_sh4_to_aica_start (void)

  Begin processing AICA queue requests.
- void snd_sh4_to_aica_stop (void)

  Stop processing AICA queue requests.
- int snd_aica_to_sh4 (void *packetout)

  Transfer a packet of data from the AICA's SH4 queue.
- void snd_poll_resp (void)

  Poll for a response from the AICA.
- void snd_pcm16_split (uint32_t *data, uint32_t *left, uint32_t *right, size_t size)

  Separates stereo PCM samples into 2 mono channels.
- void snd_pcm16_split_sq (uint32_t *data, uintptr_t left, uintptr_t right, size_t size)

  Separates stereo PCM samples into 2 mono channels with SQ transfer.
- void snd_pcm8_split (uint32_t *data, uint32_t *left, uint32_t *right, size_t size)

  Separates stereo PCM samples into 2 mono channels.
- void snd_adpcm_split (uint32_t *data, uint32_t *left, uint32_t *right, size_t size)

  Separates stereo ADPCM samples into 2 mono channels.

9.236.1 Detailed Description

Low-level sound support and memory management.

This file contains declarations for low-level sound operations and for SPU RAM pool memory management. Most of the time you'll be better off using the higher-level functionality in the sound effect support or streaming support, but this stuff can be very useful for some things.

Author

Megan Potter

9.236.2 Function Documentation

snd_adpcm_split()

```
void snd_adpcm_split (
            uint32_t * data,
            uint32_t * left,
            uint32_t * right,
            size_t size )
```

Separates stereo ADPCM samples into 2 mono channels.

Splits a buffer containing 2 interleaved channels of 4-bit ADPCM samples into 2 separate buffers of 4-bit ADPCM samples.

Parameters

data	Source buffer of interleaved stereo samples
left	Destination buffer for left mono samples
right	Destination buffer for right mono samples
size	Size of the source buffer in bytes

See also

snd_pcm16_split()

snd_aica_to_sh4()

```
int snd_aica_to_sh4 (
            void * packetout )
```

Transfer a packet of data from the AICA's SH4 queue.

This function is used to retrieve a packet of data from the AICA back to the SH4. The buffer passed in should at least contain 1024 bytes of space to make sure any packet can fit.

Parameters

packetout	The buffer to store the retrieved packet in.

Return values

-1	On failure. Failure probably indicates the queue has been corrupted, and thus should be reinitialized.
0	If no packets are available.
1	On successful copy of one packet.

snd_init()

```
int snd_init (
            void  )
```

Initialize the sound system.

This function reinitializes the whole sound system. It will not do anything unless the sound system has been shut down previously or has not been initialized yet. This will implicitly replace the program running on the AICA's ARM processor when it actually initializes anything. The default snd_stream_drv will be loaded if a new program is uploaded to the SPU.

snd_mem_available()

```
uint32 snd_mem_available (
            void  )
```

Get the size of the largest allocateable block in the SPU RAM pool.

This function returns the largest size that can be currently passed to snd_mem_malloc() and expected to not return failure. There may be more memory available in the pool, especially if multiple blocks have been allocated and freed, but calls to snd_mem_malloc() for larger blocks will return failure, since the memory is not available contiguously.

Returns

The size of the largest available block of memory in the SPU RAM pool.

snd_mem_free()

```
void snd_mem_free (
            uint32 addr )
```

Free a block of allocated memory in the SPU RAM pool.

This function frees memory previously allocated with snd_mem_malloc().

Parameters

addr	The location of the start of the block to free.

snd_mem_init()

```
int snd_mem_init (
            uint32 reserve )
```

Reinitialize the SPU RAM pool.

This function reinitializes the SPU RAM pool with the given base offset within the memory space. There is generally not a good reason to do this in your own code, but the functionality is there if needed.

Parameters

reserve	The amount of memory to reserve as a base.

Return values

0	On success (no failure conditions defined).

snd_mem_malloc()

```
uint32 snd_mem_malloc (
            size_t size )
```

Allocate memory in the SPU RAM pool.

This function acts as the memory allocator for the SPU RAM pool. It acts much like one would expect a malloc() function to act, although it does not return a pointer directly, but rather an offset in SPU RAM.

Parameters

size	The amount of memory to allocate, in bytes.

Returns

The location of the start of the block on success, or 0 on failure.

snd_mem_shutdown()

```
void snd_mem_shutdown (
            void  )
```

Shutdown the SPU RAM allocator.

There is generally no reason to be calling this function in your own code, as doing so will cause problems if you try to allocate SPU memory without calling snd_mem_init() afterwards.

snd_pcm16_split()

```
void snd_pcm16_split (
            uint32_t * data,
            uint32_t * left,
            uint32_t * right,
            size_t size )
```

Separates stereo PCM samples into 2 mono channels.

Splits a buffer containing 2 interleaved channels of 16-bit PCM samples into 2 separate buffers of 16-bit PCM samples.

Warning

All arguments must be 32-byte aligned.

Parameters

data	Source buffer of interleaved stereo samples
left	Destination buffer for left mono samples
right	Destination buffer for right mono samples
size	Size of the source buffer in bytes (must be divisible by 32)

See also

snd_pcm16_split_sq()

snd_pcm16_split_sq()

```
void snd_pcm16_split_sq (
            uint32_t * data,
            uintptr_t left,
            uintptr_t right,
            size_t size )
```

Separates stereo PCM samples into 2 mono channels with SQ transfer.

Splits a buffer containing 2 interleaved channels of 16-bit PCM samples into 2 separate buffers of 16-bit PCM samples by using the store queues for data transfer.

Warning

All arguments must be 32-byte aligned.

Parameters

data	Source buffer of interleaved stereo samples
left	Destination buffer address for left mono samples
right	Destination buffer address for right mono samples
size	Size of the source buffer in bytes (must be divisible by 32)

See also

snd_pcm16_split() Store queues must be prepared before.

snd_pcm8_split()

```
void snd_pcm8_split (
```

```
            uint32_t * data,
            uint32_t * left,
            uint32_t * right,
            size_t size )
```

Separates stereo PCM samples into 2 mono channels.

Splits a buffer containing 2 interleaved channels of 8-bit PCM samples into 2 separate buffers of 8-bit PCM samples.

Parameters

data	Source buffer of interleaved stereo samples
left	Destination buffer for left mono samples
right	Destination buffer for right mono samples
size	Size of the source buffer in bytes

See also

snd_adpcm_split()

snd_poll_resp()

```
void snd_poll_resp (
            void  )
```

Poll for a response from the AICA.

This function waits for the AICA to respond to a previously sent request. This function is not safe to call in an IRQ, as it does implicitly wait.

snd_sh4_to_aica()

```
int snd_sh4_to_aica (
            void * packet,
            uint32 size )
```

Copy a request packet to the AICA queue.

This function is to put in a low-level request using the built-in streaming sound driver.

Parameters

packet	The packet of data to copy.
size	The size of the packet, in 32-bit increments.

Return values

0	On success (no error conditions defined).

snd_sh4_to_aica_start()

```
void snd_sh4_to_aica_start (
             void  )
```

Begin processing AICA queue requests.

This function begins processing of any queued requests in the AICA queue.

snd_sh4_to_aica_stop()

```
void snd_sh4_to_aica_stop (
             void  )
```

Stop processing AICA queue requests.

This function stops the processing of any queued requests in the AICA queue.

snd_shutdown()

```
void snd_shutdown (
             void  )
```

Shut down the sound system.

This function shuts down the whole sound system, freeing memory and disabling the SPU in the process. There's not generally many good reasons for doing this in your own code.

9.237 sound.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/sound/sound.h
00004    Copyright (C) 2002 Megan Potter
00005    Copyright (C) 2023 Ruslan Rostovtsev
00006
00007 */
00008
00009 /** \file   dc/sound/sound.h
00010     \brief  Low-level sound support and memory management.
00011
00012     This file contains declarations for low-level sound operations and for SPU
00013     RAM pool memory management. Most of the time you'll be better off using the
00014     higher-level functionality in the sound effect support or streaming support,
00015     but this stuff can be very useful for some things.
00016
```

```
00017     \author Megan Potter
00018 */
00019
00020 #ifndef __DC_SOUND_SOUND_H
00021 #define __DC_SOUND_SOUND_H
00022
00023 #include <sys/cdefs.h>
00024 __BEGIN_DECLS
00025
00026 #include <arch/types.h>
00027 #include <stdint.h>
00028
00029 /** \brief  Allocate memory in the SPU RAM pool
00030
00031     This function acts as the memory allocator for the SPU RAM pool. It acts
00032     much like one would expect a malloc() function to act, although it does not
00033     return a pointer directly, but rather an offset in SPU RAM.
00034
00035     \param  size            The amount of memory to allocate, in bytes.
00036     \return                 The location of the start of the block on success,
00037                             or 0 on failure.
00038 */
00039 uint32 snd_mem_malloc(size_t size);
00040
00041 /** \brief  Free a block of allocated memory in the SPU RAM pool.
00042
00043     This function frees memory previously allocated with snd_mem_malloc().
00044
00045     \param  addr            The location of the start of the block to free.
00046 */
00047 void snd_mem_free(uint32 addr);
00048
00049 /** \brief  Get the size of the largest allocateable block in the SPU RAM pool.
00050
00051     This function returns the largest size that can be currently passed to
00052     snd_mem_malloc() and expected to not return failure. There may be more
00053     memory available in the pool, especially if multiple blocks have been
00054     allocated and freed, but calls to snd_mem_malloc() for larger blocks will
00055     return failure, since the memory is not available contiguously.
00056
00057     \return                 The size of the largest available block of memory in
00058                             the SPU RAM pool.
00059 */
00060 uint32 snd_mem_available(void);
00061
00062 /** \brief  Reinitialize the SPU RAM pool.
00063
00064     This function reinitializes the SPU RAM pool with the given base offset
00065     within the memory space. There is generally not a good reason to do this in
00066     your own code, but the functionality is there if needed.
00067
00068     \param  reserve         The amount of memory to reserve as a base.
00069     \retval 0               On success (no failure conditions defined).
00070 */
00071 int snd_mem_init(uint32 reserve);
00072
00073 /** \brief  Shutdown the SPU RAM allocator.
00074
00075     There is generally no reason to be calling this function in your own code,
00076     as doing so will cause problems if you try to allocate SPU memory without
00077     calling snd_mem_init() afterwards.
00078 */
00079 void snd_mem_shutdown(void);
00080
00081 /** \brief  Initialize the sound system.
00082
00083     This function reinitializes the whole sound system. It will not do anything
00084     unless the sound system has been shut down previously or has not been
00085     initialized yet. This will implicitly replace the program running on the
00086     AICA's ARM processor when it actually initializes anything. The default
00087     snd_stream_drv will be loaded if a new program is uploaded to the SPU.
00088 */
00089 int snd_init(void);
00090
00091 /** \brief  Shut down the sound system.
00092
00093     This function shuts down the whole sound system, freeing memory and
00094     disabling the SPU in the process. There's not generally many good reasons
00095     for doing this in your own code.
00096 */
00097 void snd_shutdown(void);
```

```
00098
00099 /** \brief  Copy a request packet to the AICA queue.
00100
00101     This function is to put in a low-level request using the built-in streaming
00102     sound driver.
00103
00104     \param  packet          The packet of data to copy.
00105     \param  size            The size of the packet, in 32-bit increments.
00106     \retval 0               On success (no error conditions defined).
00107 */
00108 int snd_sh4_to_aica(void *packet, uint32 size);
00109
00110 /** \brief  Begin processing AICA queue requests.
00111
00112     This function begins processing of any queued requests in the AICA queue.
00113 */
00114 void snd_sh4_to_aica_start(void);
00115
00116 /** \brief  Stop processing AICA queue requests.
00117
00118     This function stops the processing of any queued requests in the AICA queue.
00119 */
00120 void snd_sh4_to_aica_stop(void);
00121
00122 /** \brief  Transfer a packet of data from the AICA's SH4 queue.
00123
00124     This function is used to retrieve a packet of data from the AICA back to the
00125     SH4. The buffer passed in should at least contain 1024 bytes of space to
00126     make sure any packet can fit.
00127
00128     \param  packetout       The buffer to store the retrieved packet in.
00129     \retval -1              On failure. Failure probably indicates the queue has
00130                             been corrupted, and thus should be reinitialized.
00131     \retval 0               If no packets are available.
00132     \retval 1               On successful copy of one packet.
00133 */
00134 int snd_aica_to_sh4(void *packetout);
00135
00136 /** \brief  Poll for a response from the AICA.
00137
00138     This function waits for the AICA to respond to a previously sent request.
00139     This function is not safe to call in an IRQ, as it does implicitly wait.
00140 */
00141 void snd_poll_resp(void);
00142
00143 /** \brief  Separates stereo PCM samples into 2 mono channels.
00144
00145     Splits a buffer containing 2 interleaved channels of 16-bit PCM samples
00146     into 2 separate buffers of 16-bit PCM samples.
00147
00148     \warning
00149     All arguments must be 32-byte aligned.
00150
00151     \param data   Source buffer of interleaved stereo samples
00152     \param left   Destination buffer for left mono samples
00153     \param right  Destination buffer for right mono samples
00154     \param size   Size of the source buffer in bytes (must be divisible by 32)
00155
00156     \sa snd_pcm16_split_sq()
00157 */
00158 void snd_pcm16_split(uint32_t *data, uint32_t *left, uint32_t *right, size_t size);
00159
00160 /** \brief  Separates stereo PCM samples into 2 mono channels with SQ transfer.
00161
00162     Splits a buffer containing 2 interleaved channels of 16-bit PCM samples
00163     into 2 separate buffers of 16-bit PCM samples by using the store queues
00164     for data transfer.
00165
00166     \warning
00167     All arguments must be 32-byte aligned.
00168
00169     \param data   Source buffer of interleaved stereo samples
00170     \param left   Destination buffer address for left mono samples
00171     \param right  Destination buffer address for right mono samples
00172     \param size   Size of the source buffer in bytes (must be divisible by 32)
00173
00174     \sa snd_pcm16_split()
00175     Store queues must be prepared before.
00176 */
00177 void snd_pcm16_split_sq(uint32_t *data, uintptr_t left, uintptr_t right, size_t size);
00178
```

```
00179 /** \brief  Separates stereo PCM samples into 2 mono channels.
00180
00181     Splits a buffer containing 2 interleaved channels of 8-bit PCM samples
00182     into 2 separate buffers of 8-bit PCM samples.
00183
00184     \param data   Source buffer of interleaved stereo samples
00185     \param left   Destination buffer for left mono samples
00186     \param right  Destination buffer for right mono samples
00187     \param size   Size of the source buffer in bytes
00188
00189     \sa snd_adpcm_split()
00190 */
00191 void snd_pcm8_split(uint32_t *data, uint32_t *left, uint32_t *right, size_t size);
00192
00193 /** \brief  Separates stereo ADPCM samples into 2 mono channels.
00194
00195     Splits a buffer containing 2 interleaved channels of 4-bit ADPCM samples
00196     into 2 separate buffers of 4-bit ADPCM samples.
00197
00198     \param data   Source buffer of interleaved stereo samples
00199     \param left   Destination buffer for left mono samples
00200     \param right  Destination buffer for right mono samples
00201     \param size   Size of the source buffer in bytes
00202
00203     \sa snd_pcm16_split()
00204 */
00205 void snd_adpcm_split(uint32_t *data, uint32_t *left, uint32_t *right, size_t size);
00206
00207 __END_DECLS
00208
00209 #endif  /* __DC_SOUND_SOUND_H */
00210
```

9.238 kernel/arch/dreamcast/include/dc/sound/stream.h File Reference

Sound streaming support.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Macros

- #define SND_STREAM_MAX 4

  The maximum number of streams that can be allocated at once.
- #define SND_STREAM_BUFFER_MAX 0x10000

  The maximum buffer size for a stream.
- #define SND_STREAM_INVALID -1

  Invalid stream handle.

Typedefs

- typedef int snd_stream_hnd_t

  Stream handle type.
- typedef void ∗(∗ snd_stream_callback_t) (snd_stream_hnd_t hnd, int smp_req, int ∗smp_recv)

  Stream get data callback type.
- typedef void(∗ snd_stream_filter_t) (snd_stream_hnd_t hnd, void ∗obj, int hz, int channels, void ∗∗buffer, int ∗samplecnt)

  Stream filter callback type.

Functions

- void snd_stream_set_callback (snd_stream_hnd_t hnd, snd_stream_callback_t cb)

  Set the callback for a given stream.
- void snd_stream_set_userdata (snd_stream_hnd_t hnd, void *d)

  Set the user data for a given stream.
- void * snd_stream_get_userdata (snd_stream_hnd_t hnd)

  Get the user data for a given stream.
- void snd_stream_filter_add (snd_stream_hnd_t hnd, snd_stream_filter_t filtfunc, void *obj)

  Add a filter to the specified stream.
- void snd_stream_filter_remove (snd_stream_hnd_t hnd, snd_stream_filter_t filtfunc, void *obj)

  Remove a filter from the specified stream.
- void snd_stream_prefill (snd_stream_hnd_t hnd)

  Prefill the stream buffers.
- int snd_stream_init (void)

  Initialize the stream system.
- void snd_stream_shutdown (void)

  Shut down the stream system.
- snd_stream_hnd_t snd_stream_alloc (snd_stream_callback_t cb, int bufsize)

  Allocate a stream.
- int snd_stream_reinit (snd_stream_hnd_t hnd, snd_stream_callback_t cb)

  Reinitialize a stream.
- void snd_stream_destroy (snd_stream_hnd_t hnd)

  Destroy a stream.
- void snd_stream_queue_enable (snd_stream_hnd_t hnd)

  Enable queueing on a stream.
- void snd_stream_queue_disable (snd_stream_hnd_t hnd)

  Disable queueing on a stream.
- void snd_stream_queue_go (snd_stream_hnd_t hnd)

  Start a stream after queueing the request.
- void snd_stream_start (snd_stream_hnd_t hnd, uint32 freq, int st)

  Start a 16-bit PCM stream.
- void snd_stream_start_pcm8 (snd_stream_hnd_t hnd, uint32 freq, int st)

  Start a 8-bit PCM stream.
- void snd_stream_start_adpcm (snd_stream_hnd_t hnd, uint32 freq, int st)

  Start a 4-bit ADPCM stream.
- void snd_stream_stop (snd_stream_hnd_t hnd)

  Stop a stream.
- int snd_stream_poll (snd_stream_hnd_t hnd)

  Poll a stream.
- void snd_stream_volume (snd_stream_hnd_t hnd, int vol)

  Set the volume on the stream.

9.238.1 Detailed Description

Sound streaming support.

This file contains declarations for doing streams of sound. This underlies pretty much any decoded sounds you might use, including the Ogg Vorbis libraries. Note that this does not actually handle decoding, so you'll have to worry about that yourself (or use something in kos-ports).

Author

Megan Potter
Florian Schulze
Lawrence Sebald
Ruslan Rostovtsev

9.238.2 Macro Definition Documentation

SND_STREAM_BUFFER_MAX

```
#define SND_STREAM_BUFFER_MAX 0x10000
```

The maximum buffer size for a stream.

SND_STREAM_INVALID

```
#define SND_STREAM_INVALID -1
```

Invalid stream handle.

If a stream cannot be allocated, this will be returned.

SND_STREAM_MAX

```
#define SND_STREAM_MAX 4
```

The maximum number of streams that can be allocated at once.

9.238.3 Typedef Documentation

snd_stream_callback_t

```
typedef void *(* snd_stream_callback_t) (snd_stream_hnd_t hnd, int smp_req, int *smp_recv)
```

Stream get data callback type.

Functions for providing stream data will be of this type, and can be registered with snd_stream_set_callback().

Parameters

hnd	The stream handle being referred to.
smp_req	The number of samples requested.
smp_recv	Used to return the number of samples available.

Returns

A pointer to the buffer of samples. If stereo, the samples should be interleaved. For best perfomance use 32-byte aligned pointer.

snd_stream_filter_t

```
typedef void(* snd_stream_filter_t) (snd_stream_hnd_t hnd, void *obj, int hz, int channels, void
**buffer, int *samplecnt)
```

Stream filter callback type.

Functions providing filters over the stream data will be of this type, and can be set with snd_stream_filter_add().

Parameters

hnd	The stream being referred to.
obj	Filter user data.
hz	The frequency of the sound data.
channels	The number of channels in the sound data.
buffer	A pointer to the buffer to process. This is before any stereo separation is done. Can be changed by the filter, if appropriate.
samplecnt	A pointer to the number of samples. This can be modified by the filter, if appropriate.

snd_stream_hnd_t

```
typedef int snd_stream_hnd_t
```

Stream handle type.

Each stream will be assigned a handle, which will be of this type. Further operations on the stream will use the handle to identify which stream is being referred to.

9.238.4 Function Documentation

snd_stream_alloc()

```
snd_stream_hnd_t snd_stream_alloc (
            snd_stream_callback_t cb,
            int bufsize )
```

Allocate a stream.

This function allocates a stream and sets its parameters.

Parameters

cb	The get data callback for the stream.
bufsize	The size of the buffer for the stream.

Returns

A handle to the new stream on success, SND_STREAM_INVALID on failure.

snd_stream_destroy()

```
void snd_stream_destroy (
            snd_stream_hnd_t hnd )
```

Destroy a stream.

This function destroys a previously created stream, freeing all memory associated with it.

Parameters

hnd	The stream to clean up.

snd_stream_filter_add()

```
void snd_stream_filter_add (
            snd_stream_hnd_t hnd,
            snd_stream_filter_t filtfunc,
            void * obj )
```

Add a filter to the specified stream.

This function adds a filter to the specified stream. The filter will be called on each block of data input to the stream from then forward.

Parameters

hnd	The stream to add the filter to.
filtfunc	A pointer to the filter function.
obj	Filter function user data.

snd_stream_filter_remove()

```
void snd_stream_filter_remove (
             snd_stream_hnd_t hnd,
             snd_stream_filter_t filtfunc,
             void * obj )
```

Remove a filter from the specified stream.

This function removes a filter that was previously added to the specified stream.

Parameters

hnd	The stream to remove the filter from.
filtfunc	A pointer to the filter function to remove.
obj	The filter function's user data. Must be the same as what was passed as obj to snd_stream_filter_add().

snd_stream_get_userdata()

```
void * snd_stream_get_userdata (
             snd_stream_hnd_t hnd )
```

Get the user data for a given stream.

This function retrieves the set user data pointer for a given stream.

Parameters

hnd	The stream handle to look up.

Returns

The user data pointer set for this stream or NULL if no data pointer has been set.

snd_stream_init()

```
int snd_stream_init (
             void  )
```

Initialize the stream system.

This function initializes the sound stream system and allocates memory for it as needed. Note, this is not done by the default init, so if you're using the streaming support and not using something like the kos-ports Ogg Vorbis library, you'll need to call this yourself. This will implicitly call snd_init(), so it will potentially overwrite anything going on the AICA.

Return values

-1	On failure.
0	On success.

snd_stream_poll()

```
int snd_stream_poll (
            snd_stream_hnd_t hnd )
```

Poll a stream.

This function polls the specified stream to load more data if necessary. If using the streaming support, you must call this function periodically (most likely in a thread), or you won't get any sound output.

Parameters

hnd	The stream to poll.

Return values

-3	If NULL was returned from the callback.
-1	If no callback is set, or if the state has been corrupted.
0	On success.

snd_stream_prefill()

```
void snd_stream_prefill (
            snd_stream_hnd_t hnd )
```

Prefill the stream buffers.

This function prefills the stream buffers before starting it. This is implicitly called by snd_stream_start(), so there's probably no good reason to call this yourself.

Parameters

hnd	The stream to prefill buffers on.

snd_stream_queue_disable()

```
void snd_stream_queue_disable (
            snd_stream_hnd_t hnd )
```

Disable queueing on a stream.

This function disables queueing on the specified stream. This does not imply that a previously queued start on the stream will be fired if queueing was enabled before.

Parameters

hnd	The stream to disable queueing on.

snd_stream_queue_enable()

```
void snd_stream_queue_enable (
             snd_stream_hnd_t hnd )
```

Enable queueing on a stream.

This function enables queueing on the specified stream. This will make it so that you must call snd_stream_queue_go() to actually start the stream, after scheduling the start. This is useful for getting something ready but not firing it right away.

Parameters

hnd	The stream to enable queueing on.

snd_stream_queue_go()

```
void snd_stream_queue_go (
             snd_stream_hnd_t hnd )
```

Start a stream after queueing the request.

This function makes the stream start once a start request has been queued, if queueing mode is enabled on the stream.

Parameters

hnd	The stream to start the queue on.

snd_stream_reinit()

```
int snd_stream_reinit (
             snd_stream_hnd_t hnd,
             snd_stream_callback_t cb )
```

Reinitialize a stream.

This function reinitializes a stream, resetting its callback function.

Parameters

hnd	The stream handle to reinit.
cb	The new get data callback for the stream.

Returns

hnd

snd_stream_set_callback()

```
void snd_stream_set_callback (
             snd_stream_hnd_t hnd,
             snd_stream_callback_t cb )
```

Set the callback for a given stream.

This function sets the get data callback function for a given stream, overwriting any old callback that may have been in place.

Parameters

hnd	The stream handle for the callback.
cb	A pointer to the callback function.

snd_stream_set_userdata()

```
void snd_stream_set_userdata (
             snd_stream_hnd_t hnd,
             void * d )
```

Set the user data for a given stream.

This function sets the user data pointer for the given stream, overwriting any existing one that may have been in place. This is designed to allow the user the ability to associate a piece of data with the stream for instance to assist in identifying what sound is playing on a stream. The driver does not attempt to use this data in any way.

Parameters

hnd	The stream handle to look up.
d	A pointer to the user data.

snd_stream_shutdown()

```
void snd_stream_shutdown (
            void  )
```

Shut down the stream system.

This function shuts down the stream system and frees the memory associated with it. This does not call snd_shutdown().

snd_stream_start()

```
void snd_stream_start (
            snd_stream_hnd_t hnd,
            uint32 freq,
            int st )
```

Start a 16-bit PCM stream.

This function starts processing the given stream, prefilling the buffers as necessary. In queueing mode, this will not start playback.

Parameters

hnd	The stream to start.
freq	The frequency of the sound.
st	1 if the sound is stereo, 0 if mono.

snd_stream_start_adpcm()

```
void snd_stream_start_adpcm (
            snd_stream_hnd_t hnd,
            uint32 freq,
            int st )
```

Start a 4-bit ADPCM stream.

This function starts processing the given stream, prefilling the buffers as necessary. In queueing mode, this will not start playback.

Parameters

hnd	The stream to start.
freq	The frequency of the sound.
st	1 if the sound is stereo, 0 if mono.

snd_stream_start_pcm8()

```
void snd_stream_start_pcm8 (
             snd_stream_hnd_t hnd,
             uint32 freq,
             int st )
```

Start a 8-bit PCM stream.

This function starts processing the given stream, prefilling the buffers as necessary. In queueing mode, this will not start playback.

Parameters

hnd	The stream to start.
freq	The frequency of the sound.
st	1 if the sound is stereo, 0 if mono.

snd_stream_stop()

```
void snd_stream_stop (
             snd_stream_hnd_t hnd )
```

Stop a stream.

This function stops a stream, stopping any sound playing from it. This will happen immediately, regardless of whether queueing is enabled or not.

Parameters

hnd	The stream to stop.

snd_stream_volume()

```
void snd_stream_volume (
             snd_stream_hnd_t hnd,
             int vol )
```

Set the volume on the stream.

This function sets the volume of the specified stream.

Parameters

hnd	The stream to set volume on.
vol	The volume to set. Valid values are 0-255.

9.239 stream.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/sound/stream.h
00004    Copyright (C) 2002, 2004 Megan Potter
00005    Copyright (C) 2020 Lawrence Sebald
00006    Copyright (C) 2023 Ruslan Rostovtsev
00007
00008 */
00009
00010 /** \file   dc/sound/stream.h
00011     \brief  Sound streaming support.
00012
00013     This file contains declarations for doing streams of sound. This underlies
00014     pretty much any decoded sounds you might use, including the Ogg Vorbis
00015     libraries. Note that this does not actually handle decoding, so you'll have
00016     to worry about that yourself (or use something in kos-ports).
00017
00018     \author Megan Potter
00019     \author Florian Schulze
00020     \author Lawrence Sebald
00021     \author Ruslan Rostovtsev
00022 */
00023
00024 #ifndef __DC_SOUND_STREAM_H
00025 #define __DC_SOUND_STREAM_H
00026
00027 #include <sys/cdefs.h>
00028 __BEGIN_DECLS
00029
00030 #include <arch/types.h>
00031
00032 /** \brief  The maximum number of streams that can be allocated at once. */
00033 #define SND_STREAM_MAX 4
00034
00035 /** \brief  The maximum buffer size for a stream. */
00036 #define SND_STREAM_BUFFER_MAX 0x10000
00037
00038 /** \brief  Stream handle type.
00039
00040     Each stream will be assigned a handle, which will be of this type. Further
00041     operations on the stream will use the handle to identify which stream is
00042     being referred to.
00043 */
00044 typedef int snd_stream_hnd_t;
00045
00046 /** \brief  Invalid stream handle.
00047
00048     If a stream cannot be allocated, this will be returned.
00049 */
00050 #define SND_STREAM_INVALID -1
00051
00052 /** \brief  Stream get data callback type.
00053
00054     Functions for providing stream data will be of this type, and can be
00055     registered with snd_stream_set_callback().
00056
00057     \param  hnd             The stream handle being referred to.
00058     \param  smp_req         The number of samples requested.
00059     \param  smp_recv        Used to return the number of samples available.
00060     \return                 A pointer to the buffer of samples. If stereo, the
00061                             samples should be interleaved. For best perfomance
00062                             use 32-byte aligned pointer.
00063 */
00064 typedef void *(*snd_stream_callback_t)(snd_stream_hnd_t hnd, int smp_req,
00065                                        int *smp_recv);
00066
00067 /** \brief  Set the callback for a given stream.
00068
00069     This function sets the get data callback function for a given stream,
00070     overwriting any old callback that may have been in place.
00071
00072     \param  hnd             The stream handle for the callback.
00073     \param  cb              A pointer to the callback function.
00074 */
00075 void snd_stream_set_callback(snd_stream_hnd_t hnd, snd_stream_callback_t cb);
00076
```

```
00077 /** \brief  Set the user data for a given stream.
00078
00079     This function sets the user data pointer for the given stream, overwriting
00080     any existing one that may have been in place. This is designed to allow the
00081     user the ability to associate a piece of data with the stream for instance
00082     to assist in identifying what sound is playing on a stream. The driver does
00083     not attempt to use this data in any way.
00084
00085     \param  hnd             The stream handle to look up.
00086     \param  d               A pointer to the user data.
00087 */
00088 void snd_stream_set_userdata(snd_stream_hnd_t hnd, void *d);
00089
00090 /** \brief  Get the user data for a given stream.
00091
00092     This function retrieves the set user data pointer for a given stream.
00093
00094     \param  hnd             The stream handle to look up.
00095     \return                 The user data pointer set for this stream or NULL
00096                             if no data pointer has been set.
00097 */
00098 void *snd_stream_get_userdata(snd_stream_hnd_t hnd);
00099
00100 /* Add an effect filter to the sound stream chain. When the stream
00101    buffer filler needs more data, it starts out by calling the initial
00102    callback (set above). It then calls each function in the effect
00103    filter chain, which can modify the buffer and the amount of data
00104    available as well. Filters persist across multiple calls to _init()
00105    but will be emptied by _shutdown(). */
00106
00107 /** \brief  Stream filter callback type.
00108
00109     Functions providing filters over the stream data will be of this type, and
00110     can be set with snd_stream_filter_add().
00111
00112     \param  hnd             The stream being referred to.
00113     \param  obj             Filter user data.
00114     \param  hz              The frequency of the sound data.
00115     \param  channels        The number of channels in the sound data.
00116     \param  buffer          A pointer to the buffer to process. This is before
00117                             any stereo separation is done. Can be changed by the
00118                             filter, if appropriate.
00119     \param  samplecnt       A pointer to the number of samples. This can be
00120                             modified by the filter, if appropriate.
00121 */
00122 typedef void (*snd_stream_filter_t)(snd_stream_hnd_t hnd, void *obj, int hz,
00123                                     int channels, void **buffer,
00124                                     int *samplecnt);
00125
00126 /** \brief  Add a filter to the specified stream.
00127
00128     This function adds a filter to the specified stream. The filter will be
00129     called on each block of data input to the stream from then forward.
00130
00131     \param  hnd             The stream to add the filter to.
00132     \param  filtfunc        A pointer to the filter function.
00133     \param  obj             Filter function user data.
00134 */
00135 void snd_stream_filter_add(snd_stream_hnd_t hnd, snd_stream_filter_t filtfunc,
00136                            void *obj);
00137
00138 /** \brief  Remove a filter from the specified stream.
00139
00140     This function removes a filter that was previously added to the specified
00141     stream.
00142
00143     \param  hnd             The stream to remove the filter from.
00144     \param  filtfunc        A pointer to the filter function to remove.
00145     \param  obj             The filter function's user data. Must be the same as
00146                             what was passed as obj to snd_stream_filter_add().
00147 */
00148 void snd_stream_filter_remove(snd_stream_hnd_t hnd,
00149                               snd_stream_filter_t filtfunc, void *obj);
00150
00151 /** \brief  Prefill the stream buffers.
00152
00153     This function prefills the stream buffers before starting it. This is
00154     implicitly called by snd_stream_start(), so there's probably no good reason
00155     to call this yourself.
00156
00157     \param  hnd             The stream to prefill buffers on.
```

```
00158 */
00159 void snd_stream_prefill(snd_stream_hnd_t hnd);
00160
00161 /** \brief  Initialize the stream system.
00162
00163     This function initializes the sound stream system and allocates memory for
00164     it as needed. Note, this is not done by the default init, so if you're using
00165     the streaming support and not using something like the kos-ports Ogg Vorbis
00166     library, you'll need to call this yourself. This will implicitly call
00167     snd_init(), so it will potentially overwrite anything going on the AICA.
00168
00169     \retval -1              On failure.
00170     \retval 0               On success.
00171 */
00172 int snd_stream_init(void);
00173
00174 /** \brief  Shut down the stream system.
00175
00176     This function shuts down the stream system and frees the memory associated
00177     with it. This does not call snd_shutdown().
00178 */
00179 void snd_stream_shutdown(void);
00180
00181 /** \brief  Allocate a stream.
00182
00183     This function allocates a stream and sets its parameters.
00184
00185     \param  cb              The get data callback for the stream.
00186     \param  bufsize         The size of the buffer for the stream.
00187     \return                 A handle to the new stream on success,
00188                             SND_STREAM_INVALID on failure.
00189 */
00190 snd_stream_hnd_t snd_stream_alloc(snd_stream_callback_t cb, int bufsize);
00191
00192 /** \brief  Reinitialize a stream.
00193
00194     This function reinitializes a stream, resetting its callback function.
00195
00196     \param  hnd             The stream handle to reinit.
00197     \param  cb              The new get data callback for the stream.
00198     \return                 hnd
00199 */
00200 int snd_stream_reinit(snd_stream_hnd_t hnd, snd_stream_callback_t cb);
00201
00202 /** \brief  Destroy a stream.
00203
00204     This function destroys a previously created stream, freeing all memory
00205     associated with it.
00206
00207     \param  hnd             The stream to clean up.
00208 */
00209 void snd_stream_destroy(snd_stream_hnd_t hnd);
00210
00211 /** \brief  Enable queueing on a stream.
00212
00213     This function enables queueing on the specified stream. This will make it so
00214     that you must call snd_stream_queue_go() to actually start the stream, after
00215     scheduling the start. This is useful for getting something ready but not
00216     firing it right away.
00217
00218     \param  hnd             The stream to enable queueing on.
00219 */
00220 void snd_stream_queue_enable(snd_stream_hnd_t hnd);
00221
00222 /** \brief  Disable queueing on a stream.
00223
00224     This function disables queueing on the specified stream. This does not imply
00225     that a previously queued start on the stream will be fired if queueing was
00226     enabled before.
00227
00228     \param  hnd             The stream to disable queueing on.
00229 */
00230 void snd_stream_queue_disable(snd_stream_hnd_t hnd);
00231
00232 /** \brief  Start a stream after queueing the request.
00233
00234     This function makes the stream start once a start request has been queued,
00235     if queueing mode is enabled on the stream.
00236
00237     \param  hnd             The stream to start the queue on.
00238 */
```

```
00239 void snd_stream_queue_go(snd_stream_hnd_t hnd);
00240
00241 /** \brief  Start a 16-bit PCM stream.
00242
00243     This function starts processing the given stream, prefilling the buffers as
00244     necessary. In queueing mode, this will not start playback.
00245
00246     \param  hnd             The stream to start.
00247     \param  freq            The frequency of the sound.
00248     \param  st              1 if the sound is stereo, 0 if mono.
00249 */
00250 void snd_stream_start(snd_stream_hnd_t hnd, uint32 freq, int st);
00251
00252 /** \brief  Start a 8-bit PCM stream.
00253
00254     This function starts processing the given stream, prefilling the buffers as
00255     necessary. In queueing mode, this will not start playback.
00256
00257     \param  hnd             The stream to start.
00258     \param  freq            The frequency of the sound.
00259     \param  st              1 if the sound is stereo, 0 if mono.
00260 */
00261 void snd_stream_start_pcm8(snd_stream_hnd_t hnd, uint32 freq, int st);
00262
00263 /** \brief  Start a 4-bit ADPCM stream.
00264
00265     This function starts processing the given stream, prefilling the buffers as
00266     necessary. In queueing mode, this will not start playback.
00267
00268     \param  hnd             The stream to start.
00269     \param  freq            The frequency of the sound.
00270     \param  st              1 if the sound is stereo, 0 if mono.
00271 */
00272 void snd_stream_start_adpcm(snd_stream_hnd_t hnd, uint32 freq, int st);
00273
00274 /** \brief  Stop a stream.
00275
00276     This function stops a stream, stopping any sound playing from it. This will
00277     happen immediately, regardless of whether queueing is enabled or not.
00278
00279     \param  hnd             The stream to stop.
00280 */
00281 void snd_stream_stop(snd_stream_hnd_t hnd);
00282
00283 /** \brief  Poll a stream.
00284
00285     This function polls the specified stream to load more data if necessary. If
00286     using the streaming support, you must call this function periodically (most
00287     likely in a thread), or you won't get any sound output.
00288
00289     \param  hnd             The stream to poll.
00290     \retval -3              If NULL was returned from the callback.
00291     \retval -1              If no callback is set, or if the state has been
00292                             corrupted.
00293     \retval 0               On success.
00294 */
00295 int snd_stream_poll(snd_stream_hnd_t hnd);
00296
00297 /** \brief  Set the volume on the stream.
00298
00299     This function sets the volume of the specified stream.
00300
00301     \param  hnd             The stream to set volume on.
00302     \param  vol             The volume to set. Valid values are 0-255.
00303 */
00304 void snd_stream_volume(snd_stream_hnd_t hnd, int vol);
00305
00306 __END_DECLS
00307
00308 #endif  /* __DC_SOUND_STREAM_H */
```

9.240 kernel/arch/dreamcast/include/dc/spu.h File Reference

Functions related to sound.

```
#include <sys/cdefs.h>
#include <arch/types.h>
#include <arch/memory.h>
#include <dc/g2bus.h>
```

Macros

- #define SPU_RAM_BASE 0x00800000

  Sound ram address from the SH4 side.
- #define SPU_RAM_UNCACHED_BASE (MEM_AREA_P2_BASE | SPU_RAM_BASE)

Typedefs

- typedef g2_dma_callback_t spu_dma_callback_t

  SPU DMA callback type.

Functions

- void spu_memload (uintptr_t to, void *from, size_t length)

  Copy a block of data to sound RAM.
- void spu_memload_sq (uintptr_t to, void *from, size_t length)

  Copy a block of data to sound RAM.
- void spu_memread (void *to, uintptr_t from, size_t length)

  Copy a block of data from sound RAM.
- void spu_memset (uintptr_t to, uint32_t what, size_t length)

  Set a block of sound RAM to the specified value.
- int spu_dma_transfer (void *from, uintptr_t dest, size_t length, int block, spu_dma_callback_t callback, void *cbdata)

  Copy a block of data from SH4 RAM to sound RAM via DMA.
- void spu_enable (void)

  Enable the SPU.
- void spu_disable (void)

  Disable the SPU.
- void spu_cdda_volume (int left_volume, int right_volume)

  Set CDDA volume.
- void spu_cdda_pan (int left_pan, int right_pan)

  Set CDDA panning.
- void spu_master_mixer (int volume, int stereo)

  Set master mixer settings.
- int spu_init (void)

  Initialize the SPU.
- int spu_shutdown (void)

  Shutdown the SPU.
- void spu_reset_chans (void)

  Reset SPU channels.

9.240.1 Detailed Description

Functions related to sound.

This file deals with memory transfers and the like for the sound hardware.

Author

Megan Potter
Ruslan Rostovtsev

9.240.2 Macro Definition Documentation

SPU_RAM_BASE

```
#define SPU_RAM_BASE 0x00800000
```

Sound ram address from the SH4 side.

SPU_RAM_UNCACHED_BASE

```
#define SPU_RAM_UNCACHED_BASE (MEM_AREA_P2_BASE | SPU_RAM_BASE)
```

9.240.3 Typedef Documentation

spu_dma_callback_t

```
typedef g2_dma_callback_t spu_dma_callback_t
```

SPU DMA callback type.

9.240.4 Function Documentation

spu_cdda_pan()

```
void spu_cdda_pan (
            int left_pan,
            int right_pan )
```

Set CDDA panning.

Valid values are from 0-31. 16 is centered.

Parameters

left_pan	Pan of the left channel.
right_pan	Pan of the right channel.

spu_cdda_volume()

```
void spu_cdda_volume (
             int left_volume,
             int right_volume )
```

Set CDDA volume.

Valid volume values are 0-15.

Parameters

left_volume	Volume of the left channel.
right_volume	Volume of the right channel.

spu_disable()

```
void spu_disable (
             void  )
```

Disable the SPU.

This function resets all sound channels and puts the ARM in a reset state.

spu_dma_transfer()

```
int spu_dma_transfer (
             void * from,
             uintptr_t dest,
             size_t length,
             int block,
             spu_dma_callback_t callback,
             void * cbdata )
```

Copy a block of data from SH4 RAM to sound RAM via DMA.

This function sets up a DMA transfer from main RAM to the sound RAM with G2 DMA.

Parameters

from	A pointer in main RAM to transfer from. Must be 32-byte aligned.

Parameters

dest	Offset in sound RAM to transfer to. Do not include the 0xA0800000 part, its implied. Must be 32-byte aligned.
length	Number of bytes to copy. Must be a multiple of 32.
block	1 if you want to wait for the transfer to complete, 0 otherwise (use the callback for this case).
callback	Function to call when the DMA completes. Can be NULL if you don't want to have a callback. This will be called in an interrupt context, so keep that in mind when writing the function.
cbdata	Data to pass to the callback function.

Return values

-1	On failure. Sets errno as appropriate.
0	On success.

Error Conditions:

EINVAL - Invalid channel
EFAULT - from or dest is not aligned
EIO - I/O error

spu_enable()

```
void spu_enable (
              void  )
```

Enable the SPU.

This function resets all sound channels and lets the ARM out of reset.

spu_init()

```
int spu_init (
              void  )
```

Initialize the SPU.

This function will reset the SPU, clear the sound RAM, reinit the CDDA support and run an infinite loop on the ARM.

Return values

0	On success (no error conditions defined).

spu_master_mixer()

```
void spu_master_mixer (
            int volume,
            int stereo )
```

Set master mixer settings.

This function sets the master mixer volume and mono/stereo setting.

Parameters

volume	The volume to set (0-15).
stereo	1 for stereo output, 0 for mono.

spu_memload()

```
void spu_memload (
            uintptr_t to,
            void * from,
            size_t length )
```

Copy a block of data to sound RAM.

This function acts much like memcpy() but copies to the sound RAM area.

Parameters

to	The offset in sound RAM to copy to. Do not include the 0xA0800000 part, it is implied.
from	A pointer to copy from.
length	The number of bytes to copy. Automatically rounded up to be a multiple of 4.

spu_memload_sq()

```
void spu_memload_sq (
            uintptr_t to,
            void * from,
            size_t length )
```

Copy a block of data to sound RAM.

This function acts much like memcpy() but copies to the sound RAM area by using the store queues.

Parameters

to	The offset in sound RAM to copy to. Do not include the 0xA0800000 part, it is implied.
from	A pointer to copy from.
length	The number of bytes to copy. Automatically rounded up to be a multiple of 4.

spu_memread()

```
void spu_memread (
            void * to,
            uintptr_t from,
            size_t length )
```

Copy a block of data from sound RAM.

This function acts much like memcpy() but copies from the sound RAM area.

Parameters

to	A pointer to copy to.
from	The offset in sound RAM to copy from. Do not include the 0xA0800000 part, it is implied.
length	The number of bytes to copy. Automatically rounded up to be a multiple of 4.

spu_memset()

```
void spu_memset (
            uintptr_t to,
            uint32_t what,
            size_t length )
```

Set a block of sound RAM to the specified value.

This function acts like memset4(), setting the specified block of sound RAM to the given 32-bit value.

Parameters

to	The offset in sound RAM to set at. Do not include the 0xA0800000 part, it is implied.
what	The value to set.
length	The number of bytes to copy. Automatically rounded up to be a multiple of 4.

spu_reset_chans()

```
void spu_reset_chans (
            void  )
```

Reset SPU channels.

spu_shutdown()

```
int spu_shutdown (
            void  )
```

Shutdown the SPU.

This function disables the SPU and clears sound RAM.

Return values

0	On success (no error conditions defined).

9.241 spu.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/spu.h
00004    Copyright (C) 2000, 2001 Megan Potter
00005    Copyright (C) 2023 Ruslan Rostovtsev
00006
00007 */
00008
00009 /** \file   dc/spu.h
00010     \brief  Functions related to sound.
00011
00012     This file deals with memory transfers and the like for the sound hardware.
00013
00014     \author Megan Potter
00015     \author Ruslan Rostovtsev
00016 */
00017
00018 #ifndef __DC_SPU_H
00019 #define __DC_SPU_H
00020
00021 #include <sys/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 #include <arch/types.h>
00025 #include <arch/memory.h>
00026 #include <dc/g2bus.h>
00027
00028 /** \brief  Sound ram address from the SH4 side */
00029 #define SPU_RAM_BASE 0x00800000
00030 #define SPU_RAM_UNCACHED_BASE (MEM_AREA_P2_BASE | SPU_RAM_BASE)
00031
00032 /** \brief  Copy a block of data to sound RAM.
00033
00034     This function acts much like memcpy() but copies to the sound RAM area.
00035
00036     \param  to              The offset in sound RAM to copy to. Do not include
00037                             the 0xA0800000 part, it is implied.
00038     \param  from            A pointer to copy from.
00039     \param  length          The number of bytes to copy. Automatically rounded
00040                             up to be a multiple of 4.
00041 */
00042 void spu_memload(uintptr_t to, void *from, size_t length);
00043
00044
00045 /** \brief  Copy a block of data to sound RAM.
00046
00047     This function acts much like memcpy() but copies to the sound RAM area
00048     by using the store queues.
00049
00050     \param  to              The offset in sound RAM to copy to. Do not include
00051                             the 0xA0800000 part, it is implied.
00052     \param  from            A pointer to copy from.
00053     \param  length          The number of bytes to copy. Automatically rounded
00054                             up to be a multiple of 4.
00055 */
00056 void spu_memload_sq(uintptr_t to, void *from, size_t length);
00057
00058 /** \brief  Copy a block of data from sound RAM.
00059
00060     This function acts much like memcpy() but copies from the sound RAM area.
00061
00062     \param  to              A pointer to copy to.
00063     \param  from            The offset in sound RAM to copy from. Do not include
00064                             the 0xA0800000 part, it is implied.
00065     \param  length          The number of bytes to copy. Automatically rounded
00066                             up to be a multiple of 4.
```

```
00067 */
00068 void spu_memread(void *to, uintptr_t from, size_t length);
00069
00070 /** \brief  Set a block of sound RAM to the specified value.
00071
00072     This function acts like memset4(), setting the specified block of sound RAM
00073     to the given 32-bit value.
00074
00075     \param  to              The offset in sound RAM to set at. Do not include
00076                             the 0xA0800000 part, it is implied.
00077     \param  what            The value to set.
00078     \param  length          The number of bytes to copy. Automatically rounded
00079                             up to be a multiple of 4.
00080 */
00081 void spu_memset(uintptr_t to, uint32_t what, size_t length);
00082
00083 /* DMA copy from SH-4 RAM to SPU RAM; length must be a multiple of 32,
00084    and the source and destination addresses must be aligned on 32-byte
00085    boundaries. If block is non-zero, this function won't return until
00086    the transfer is complete. If callback is non-NULL, it will be called
00087    upon completion (in an interrupt context!). Returns <0 on error. */
00088
00089 /** \brief  SPU DMA callback type. */
00090 typedef g2_dma_callback_t spu_dma_callback_t;
00091
00092 /** \brief  Copy a block of data from SH4 RAM to sound RAM via DMA.
00093
00094     This function sets up a DMA transfer from main RAM to the sound RAM with G2
00095     DMA.
00096
00097     \param  from            A pointer in main RAM to transfer from. Must be
00098                             32-byte aligned.
00099     \param  dest            Offset in sound RAM to transfer to. Do not include
00100                             the 0xA0800000 part, its implied. Must be 32-byte
00101                             aligned.
00102     \param  length          Number of bytes to copy. Must be a multiple of 32.
00103     \param  block           1 if you want to wait for the transfer to complete,
00104                             0 otherwise (use the callback for this case).
00105     \param  callback        Function to call when the DMA completes. Can be NULL
00106                             if you don't want to have a callback. This will be
00107                             called in an interrupt context, so keep that in mind
00108                             when writing the function.
00109     \param  cbdata          Data to pass to the callback function.
00110     \retval -1              On failure. Sets errno as appropriate.
00111     \retval 0               On success.
00112
00113     \par    Error Conditions:
00114     \em     EINVAL - Invalid channel \n
00115     \em     EFAULT - from or dest is not aligned \n
00116     \em     EIO - I/O error
00117 */
00118 int spu_dma_transfer(void *from, uintptr_t dest, size_t length, int block,
00119                      spu_dma_callback_t callback, void *cbdata);
00120
00121 /** \brief  Enable the SPU.
00122
00123     This function resets all sound channels and lets the ARM out of reset.
00124 */
00125 void spu_enable(void);
00126
00127 /** \brief  Disable the SPU.
00128
00129     This function resets all sound channels and puts the ARM in a reset state.
00130 */
00131 void spu_disable(void);
00132
00133 /** \brief  Set CDDA volume.
00134
00135     Valid volume values are 0-15.
00136
00137     \param  left_volume     Volume of the left channel.
00138     \param  right_volume    Volume of the right channel.
00139 */
00140 void spu_cdda_volume(int left_volume, int right_volume);
00141
00142 /** \brief  Set CDDA panning.
00143
00144     Valid values are from 0-31. 16 is centered.
00145
00146     \param  left_pan        Pan of the left channel.
00147     \param  right_pan       Pan of the right channel.
```

```
00148 */
00149 void spu_cdda_pan(int left_pan, int right_pan);
00150
00151 /** \brief  Set master mixer settings.
00152
00153     This function sets the master mixer volume and mono/stereo setting.
00154
00155     \param  volume          The volume to set (0-15).
00156     \param  stereo          1 for stereo output, 0 for mono.
00157 */
00158 void spu_master_mixer(int volume, int stereo);
00159
00160 /** \brief  Initialize the SPU.
00161
00162     This function will reset the SPU, clear the sound RAM, reinit the CDDA
00163     support and run an infinite loop on the ARM.
00164
00165     \retval 0               On success (no error conditions defined).
00166 */
00167 int spu_init(void);
00168
00169 /** \brief  Shutdown the SPU.
00170
00171     This function disables the SPU and clears sound RAM.
00172
00173     \retval 0               On success (no error conditions defined).
00174 */
00175 int spu_shutdown(void);
00176
00177 /** \brief  Reset SPU channels. */
00178 void spu_reset_chans(void);
00179
00180 __END_DECLS
00181
00182 #endif  /* __DC_SPU_H */
00183
```

9.242 kernel/arch/dreamcast/include/dc/sq.h File Reference

Functions to access the SH4 Store Queues.

```
#include <sys/cdefs.h>
#include <stdint.h>
#include <arch/types.h>
#include <arch/memory.h>
```

Macros

- #define QACR0 (*(volatile uint32_t *)(void *)0xff000038)

  Store Queue 0 access register.
- #define QACR1 (*(volatile uint32_t *)(void *)0xff00003c)

  Store Queue 1 access register <>
- #define SET_QACR_REGS(dest0, dest1)

  Set Store Queue QACR registers.*
- #define SQ_MASK_DEST_ADDR(dest) (MEM_AREA_SQ_BASE | ((uintptr_t)(dest) & 0x03ffffe0))

  Mask dest to Store Queue area as address.
- #define SQ_MASK_DEST(dest) ((uint32_t *)(void *) SQ_MASK_DEST_ADDR(dest))

  Mask dest to Store Queue area as pointer.

Functions

- void sq_lock (void)

  Lock Store Queues.
- void sq_unlock (void)

  Unlock Store Queues.
- void * sq_cpy (void *dest, const void *src, size_t n)

  Copy a block of memory.
- void * sq_set (void *dest, uint32_t c, size_t n)

  Set a block of memory to an 8-bit value.
- void * sq_set16 (void *dest, uint32_t c, size_t n)

  Set a block of memory to a 16-bit value.
- void * sq_set32 (void *dest, uint32_t c, size_t n)

  Set a block of memory to a 32-bit value.
- void sq_clr (void *dest, size_t n)

  Clear a block of memory.
- void * sq_cpy_pvr (void *dest, const void *src, size_t n)

  Copy a block of memory to VRAM.
- void * sq_set_pvr (void *dest, uint32_t c, size_t n)

  Set a block of PVR memory to a 16-bit value.

9.242.1 Detailed Description

Functions to access the SH4 Store Queues.

Author

Andrew Kieschnick

9.243 sq.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kernel/arch/dreamcast/include/dc/sq.h
00004    Copyright (C) 2000-2001 Andrew Kieschnick
00005    Copyright (C) 2023 Falco Girgis
00006    Copyright (C) 2023 Andy Barajas
00007    Copyright (C) 2023 Ruslan Rostovtsev
00008 */
00009
00010 /** \file    dc/sq.h
00011     \ingroup store_queues
00012     \brief   Functions to access the SH4 Store Queues.
00013
00014     \author Andrew Kieschnick
00015 */
00016
00017 /** \defgroup  store_queues Store Queues
00018     \brief  SH4 CPU Peripheral for burst memory transactions.
00019
00020     The store queues are a way to do efficient burst transfers from the CPU to
00021     external memory. They can be used in a variety of ways, such as to transfer
00022     a texture to PVR memory. The transfers are in units of 32-bytes, and the
00023     destinations must be 32-byte aligned.
00024
```

```
00025     \note
00026     Mastery over knowing when and how to utilize the store queues is
00027     important when trying to push the limits of the Dreamcast, specifically
00028     when transferring chunks of data between regions of memory. It is often
00029     the case that the DMA is faster for transactions which are consistently
00030     large; however, the store queues tend to have better performance and
00031     have less configuration overhead when bursting smaller chunks of data.
00032 */
00033
00034 #ifndef __DC_SQ_H
00035 #define __DC_SQ_H
00036
00037 #include <sys/cdefs.h>
00038 __BEGIN_DECLS
00039
00040 #include <stdint.h>
00041 #include <arch/types.h>
00042 #include <arch/memory.h>
00043
00044 /** \brief   Store Queue 0 access register
00045     \ingroup store_queues
00046 */
00047 #define QACR0 (*(volatile uint32_t *)(void *)0xff000038)
00048
00049 /** \brief   Store Queue 1 access register
00050     \ingroup store_queues
00051 */
00052 #define QACR1 (*(volatile uint32_t *)(void *)0xff00003c)
00053
00054 /** \brief   Set Store Queue QACR* registers
00055     \ingroup store_queues
00056 */
00057 #define SET_QACR_REGS(dest0, dest1) \
00058     do { \
00059         QACR0 = ((uintptr_t)(dest0)) >> 24; \
00060         QACR1 = ((uintptr_t)(dest1)) >> 24; \
00061     } while(0)
00062
00063 /** \brief   Mask dest to Store Queue area as address
00064     \ingroup store_queues
00065 */
00066 #define SQ_MASK_DEST_ADDR(dest) \
00067     (MEM_AREA_SQ_BASE | ((uintptr_t)(dest) & 0x03ffffe0))
00068
00069 /** \brief   Mask dest to Store Queue area as pointer
00070     \ingroup store_queues
00071 */
00072 #define SQ_MASK_DEST(dest) \
00073     ((uint32_t *)(void *) SQ_MASK_DEST_ADDR(dest))
00074
00075 /** \brief  Lock Store Queues
00076     \ingroup store_queues
00077
00078     Locks the store queues so that they cannot be used from another thread
00079     until unlocked.
00080
00081     \warning
00082     This function is called automatically by the store queue API provided by KOS;
00083     however, it must be called manually when driving the SQs directly from outside
00084     of this API.
00085
00086     \sa sq_unlock()
00087 */
00088 void sq_lock(void);
00089
00090 /** \brief  Unlock Store Queues
00091     \ingroup store_queues
00092
00093     Unlocks the store queues so that they can be used from any thread.
00094
00095     \note
00096     sq_lock() should've already been called previously.
00097
00098     \warning
00099     sq_lock() and sq_unlock() are called automatically by the store queue API provided
00100     by KOS; however, they must be called manually when driving the SQs directly from
00101     outside this API.
00102
00103     \sa sq_lock()
00104 */
00105 void sq_unlock(void);
```

```
00106
00107 /** \brief   Copy a block of memory.
00108     \ingroup store_queues
00109
00110     This function is similar to memcpy4(), but uses the store queues to do its
00111     work.
00112
00113     \warning
00114     The dest pointer must be at least 32-byte aligned, the src pointer
00115     must be at least 4-byte aligned (8-byte aligned uses fast path),
00116     and n must be a multiple of 32!
00117
00118     \param  dest            The address to copy to (32-byte aligned).
00119     \param  src             The address to copy from (32-bit (4/8-byte) aligned).
00120     \param  n               The number of bytes to copy (multiple of 32).
00121     \return                 The original value of dest.
00122
00123     \sa sq_cpy_pvr()
00124 */
00125 void * sq_cpy(void *dest, const void *src, size_t n);
00126
00127 /** \brief   Set a block of memory to an 8-bit value.
00128     \ingroup store_queues
00129
00130     This function is similar to calling memset(), but uses the store queues to
00131     do its work.
00132
00133     \warning
00134     The dest pointer must be a 32-byte aligned with n being a multiple of 32,
00135     and only the low 8-bits are used from c.
00136
00137     \param  dest            The address to begin setting at (32-byte aligned).
00138     \param  c               The value to set (in the low 8-bits).
00139     \param  n               The number of bytes to set (multiple of 32).
00140     \return                 The original value of dest.
00141
00142     \sa sq_set16(), sq_set32(), sq_set_pvr()
00143 */
00144 void * sq_set(void *dest, uint32_t c, size_t n);
00145
00146 /** \brief   Set a block of memory to a 16-bit value.
00147     \ingroup store_queues
00148
00149     This function is similar to calling memset2(), but uses the store queues to
00150     do its work.
00151
00152     \warning
00153     The dest pointer must be a 32-byte aligned with n being a multiple of 32,
00154     and only the low 16-bits are used from c.
00155
00156     \param  dest            The address to begin setting at (32-byte aligned).
00157     \param  c               The value to set (in the low 16-bits).
00158     \param  n               The number of bytes to set (multiple of 32).
00159     \return                 The original value of dest.
00160
00161     \sa sq_set(), sq_set32(), sq_set_pvr()
00162 */
00163 void * sq_set16(void *dest, uint32_t c, size_t n);
00164
00165 /** \brief   Set a block of memory to a 32-bit value.
00166     \ingroup store_queues
00167
00168     This function is similar to calling memset4(), but uses the store queues to
00169     do its work.
00170
00171     \warning
00172     The dest pointer must be a 32-byte aligned with n being a multiple of 32!
00173
00174     \param  dest            The address to begin setting at (32-byte aligned).
00175     \param  c               The value to set (all 32-bits).
00176     \param  n               The number of bytes to set (multiple of 32).
00177     \return                 The original value of dest.
00178
00179     \sa sq_set(), sq_set16(), sq_set_pvr()
00180 */
00181 void * sq_set32(void *dest, uint32_t c, size_t n);
00182
00183 /** \brief   Clear a block of memory.
00184     \ingroup store_queues
00185
00186     This function is similar to calling memset() with a value to set of 0, but
```

```
00187     uses the store queues to do its work.
00188
00189     \warning
00190     The dest pointer must be a 32-byte aligned with n being a multiple of 32!
00191
00192     \param  dest            The address to begin clearing at (32-byte aligned).
00193     \param  n               The number of bytes to clear (multiple of 32).
00194 */
00195 void sq_clr(void *dest, size_t n);
00196
00197 /** \brief   Copy a block of memory to VRAM
00198     \ingroup store_queues
00199     \author  TapamN
00200
00201     This function is similar to sq_cpy(), but it has been
00202     optimized for writing to a destination residing within VRAM.
00203
00204     \note
00205     TapamN has reported over a 2x speedup versus the regular
00206     sq_cpy() when using this function to write to VRAM.
00207
00208     \warning
00209     This function cannot be used at the same time as a PVR DMA transfer.
00210
00211     The dest pointer must be at least 32-byte aligned and reside
00212     in video memory, the src pointer must be at least 8-byte aligned,
00213     and n must be a multiple of 32.
00214
00215     \param  dest            The address to copy to (32-byte aligned).
00216     \param  src             The address to copy from (32-bit (8-byte) aligned).
00217     \param  n               The number of bytes to copy (multiple of 32).
00218     \return                 The original value of dest.
00219
00220     \sa sq_cpy()
00221 */
00222 void * sq_cpy_pvr(void *dest, const void *src, size_t n);
00223
00224 /** \brief   Set a block of PVR memory to a 16-bit value.
00225     \ingroup store_queues
00226
00227     This function is similar to sq_set16(), but it has been
00228     optimized for writing to a destination residing within VRAM.
00229
00230     \warning
00231     This function cannot be used at the same time as a PVR DMA transfer.
00232
00233     The dest pointer must be at least 32-byte aligned and reside in video
00234     memory, n must be a multiple of 32 and only the low 16-bits are used
00235     from c.
00236
00237     \param  dest            The address to begin setting at (32-byte aligned).
00238     \param  c               The value to set (in the low 16-bits).
00239     \param  n               The number of bytes to set (multiple of 32).
00240     \return                 The original value of dest.
00241
00242     \sa sq_set(), sq_set16(), sq_set32()
00243 */
00244 void * sq_set_pvr(void *dest, uint32_t c, size_t n);
00245
00246 __END_DECLS
00247
00248 #endif
```

9.244 kernel/arch/dreamcast/include/dc/ubc.h File Reference

User-break controller support.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Macros

- #define BARA (*((vuint32*)0xFF200000))
  BARA register.
- #define BASRA (*((vuint8*)0xFF000014))
  BASRA register.
- #define BAMRA (*((vuint8*)0xFF200004))
  BAMRA register.
- #define BBRA (*((vuint16*)0xFF200008))
  BBRA register.
- #define BARB (*((vuint32*)0xFF20000C))
  BARB register.
- #define BASRB (*((vuint8*)0xFF000018))
  BASRB register.
- #define BAMRB (*((vuint8*)0xFF200010))
  BAMRB register.
- #define BBRB (*((vuint16*)0xFF200014))
  BBRB register.
- #define BRCR (*((vuint16*)0xFF200020))
  BRCR register.

Functions

- static void ubc_pause (void)
  Pause after setting UBC parameters.
- static void ubc_disable_all (void)
  Disable all UBC breakpoints.
- static void ubc_break_data_write (uint32 address)
  Set a UBC data-write breakpoint at the given address.
- static void ubc_break_inst (uint32 address, int use_dbr)
  Set a UBC instruction access breakpoint at the given address.

9.244.1 Detailed Description

User-break controller support.

This file defines some functionality for using the SH4 UBC.

Author

Megan Potter

9.244.2 Function Documentation

ubc_break_data_write()

```
static void ubc_break_data_write (
            uint32 address )  [inline], [static]
```

Set a UBC data-write breakpoint at the given address.

Parameters

address	The address to set the breakpoint at

References BAMRA, BARA, BASRA, BBRA, BRCR, and ubc_pause().

ubc_break_inst()

```
static void ubc_break_inst (
            uint32 address,
            int use_dbr )  [inline], [static]
```

Set a UBC instruction access breakpoint at the given address.

Parameters

address	The address to set the breakpoint at.
use_dbr	Use the DBR register as the base for the exception.

References BAMRA, BARA, BASRA, BBRA, BRCR, and ubc_pause().

ubc_disable_all()

```
static void ubc_disable_all (
            void  )  [inline], [static]
```

Disable all UBC breakpoints.

References BBRA, BBRB, and ubc_pause().

ubc_pause()

```
static void ubc_pause (
            void  )  [inline], [static]
```

Pause after setting UBC parameters.

Referenced by ubc_break_data_write(), ubc_break_inst(), and ubc_disable_all().

9.245 ubc.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kernel/arch/dreamcast/include/dc/ubc.h
00004    (C)2002 Megan Potter
00005
00006 */
00007
00008 /** \file   dc/ubc.h
00009     \brief  User-break controller support.
00010
00011     This file defines some functionality for using the SH4 UBC.
00012
00013     \author Megan Potter
00014 */
00015
00016 #ifndef __DC_UBC_H
00017 #define __DC_UBC_H
00018
00019 #include <sys/cdefs.h>
00020 __BEGIN_DECLS
00021
00022 #include <arch/types.h>
00023
00024 /* From the SH-4 PDF */
00025 /** \defgroup   ubc_regs    UBC Registers
00026
00027     These registers are as documented in the SH4 manual. Consult it for more
00028     information.
00029
00030     @{
00031 */
00032 #define BARA (*((vuint32*)0xFF200000))  /**< \brief BARA register. */
00033 #define BASRA (*((vuint8*)0xFF000014))  /**< \brief BASRA register. */
00034 #define BAMRA (*((vuint8*)0xFF200004))  /**< \brief BAMRA register. */
00035 #define BBRA (*((vuint16*)0xFF200008))  /**< \brief BBRA register. */
00036 #define BARB (*((vuint32*)0xFF20000C))  /**< \brief BARB register. */
00037 #define BASRB (*((vuint8*)0xFF000018))  /**< \brief BASRB register. */
00038 #define BAMRB (*((vuint8*)0xFF200010))  /**< \brief BAMRB register. */
00039 #define BBRB (*((vuint16*)0xFF200014))  /**< \brief BBRB register. */
00040 #define BRCR (*((vuint16*)0xFF200020))  /**< \brief BRCR register. */
00041 /** @} */
00042
00043 /* These are inlined to avoid complications with using them */
00044
00045 /** \brief  Pause after setting UBC parameters. */
00046 static inline void ubc_pause(void) {
00047     __asm__ __volatile__("nop\n"
00048                          "nop\n"
00049                          "nop\n"
00050                          "nop\n"
00051                          "nop\n"
00052                          "nop\n"
00053                          "nop\n"
00054                          "nop\n"
00055                          "nop\n"
00056                          "nop\n"
00057                          "nop\n");
00058 }
00059
00060 /** \brief Disable all UBC breakpoints. */
00061 static inline void ubc_disable_all(void) {
00062     BBRA = 0;
00063     BBRB = 0;
00064     ubc_pause();
00065 }
00066
00067 /** \brief  Set a UBC data-write breakpoint at the given address.
00068     \param  address         The address to set the breakpoint at
00069 */
00070 static inline void ubc_break_data_write(uint32 address) {
00071     BASRA = 0;      /* ASID = 0 */
00072     BARA = address;     /* Break address */
00073     BAMRA = 4;      /* Mask the ASID */
00074     BRCR = 0;       /* Nothing special, clear all flags */
00075     BBRA = 0x28;        /* Operand write cycle, no size constraint */
00076     ubc_pause();
```

```
00077 }
00078
00079 /** \brief  Set a UBC instruction access breakpoint at the given address.
00080     \param  address         The address to set the breakpoint at.
00081     \param  use_dbr         Use the DBR register as the base for the exception.
00082 */
00083 static inline void ubc_break_inst(uint32 address, int use_dbr) {
00084     BASRA = 0;      /* ASID = 0 */
00085     BARA = address; /* Break address */
00086     BAMRA = 4;      /* Mask the ASID */
00087
00088     if(use_dbr) {
00089         BRCR = 1;   /* Use the DBR as the base for the IRQ */
00090     }
00091     else {
00092         BRCR = 0;
00093     }
00094
00095     BBRA = 0x1C;    /* Instruction cycle, no size constraint */
00096     ubc_pause();
00097 }
00098
00099 /* More to come.... */
00100
00101 __END_DECLS
00102
00103 #endif  /* __DC_UBC_H */
00104
```

9.246 kernel/arch/dreamcast/include/dc/vblank.h File Reference

VBlank handler registration.

```
#include <sys/cdefs.h>
#include <dc/asic.h>
```

Functions

- int vblank_handler_add (asic_evt_handler hnd)

  Add a vblank handler.
- int vblank_handler_remove (int handle)

  Remove a vblank handler.

9.246.1 Detailed Description

VBlank handler registration.

This file allows functions to be registered to be called on each vblank interrupt that occurs. This gives a way to schedule small functions that must occur regularly, without using threads.

Author

Megan Potter

9.246.2 Function Documentation

vblank_handler_add()

```
int vblank_handler_add (
             asic_evt_handler hnd )
```

Add a vblank handler.

This function adds a handler to the vblank handler list. The function will be called at the start of every vblank period with the same parameters that were passed to the IRQ handler for vblanks.

Parameters

hnd	The handler to add.

Returns

The handle id on success, or <0 on failure.

vblank_handler_remove()

```
int vblank_handler_remove (
             int handle )
```

Remove a vblank handler.

This function removes the specified handler from the vblank handler list.

Parameters

handle	The handle id to remove (returned by vblank_handler_add() when the handler was added).

Return values

0	On success.
-1	On failure.

9.247 vblank.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/vblank.h
00004    Copyright (C)2003 Megan Potter
```

```
00005
00006 */
00007
00008 /** \file   dc/vblank.h
00009     \brief  VBlank handler registration.
00010
00011     This file allows functions to be registered to be called on each vblank
00012     interrupt that occurs. This gives a way to schedule small functions that
00013     must occur regularly, without using threads.
00014
00015     \author Megan Potter
00016 */
00017
00018 #ifndef __DC_VBLANK_H
00019 #define __DC_VBLANK_H
00020
00021 #include <sys/cdefs.h>
00022 __BEGIN_DECLS
00023
00024 #include <dc/asic.h>
00025
00026 /** \brief  Add a vblank handler.
00027
00028     This function adds a handler to the vblank handler list. The function will
00029     be called at the start of every vblank period with the same parameters that
00030     were passed to the IRQ handler for vblanks.
00031
00032     \param  hnd             The handler to add.
00033     \return                 The handle id on success, or <0 on failure.
00034 */
00035 int vblank_handler_add(asic_evt_handler hnd);
00036
00037 /** \brief  Remove a vblank handler.
00038
00039     This function removes the specified handler from the vblank handler list.
00040
00041     \param  handle          The handle id to remove (returned by
00042                             vblank_handler_add() when the handler was added).
00043     \retval 0               On success.
00044     \retval -1              On failure.
00045 */
00046 int vblank_handler_remove(int handle);
00047
00048 /* \cond */
00049 /** Initialize the vblank handler. This must be called after the asic module
00050     is initialized. */
00051 int vblank_init(void);
00052
00053 /** Shut down the vblank handler. */
00054 int vblank_shutdown(void);
00055 /* \endcond */
00056
00057 __END_DECLS
00058
00059 #endif  /* __DC_VBLANK_H */
00060
```

9.248 kernel/arch/dreamcast/include/dc/vec3f.h File Reference

Basic matrix operations.

`#include <sys/cdefs.h>`

Data Structures

- struct vec3f_t

Macros

- #define R_DEG 182.044444436233495419095237937 43
- #define R_RAD 10430.37835
- #define vec3f_dot(x1, y1, z1, x2, y2, z2, w)

  Macro to return the scalar dot product of two 3d vectors.
- #define vec3f_length(x, y, z, w)

  Macro to return scalar Euclidean length of a 3d vector.
- #define vec3f_distance(x1, y1, z1, x2, y2, z2, w)

  Macro to return the Euclidean distance between two 3d vectors.
- #define vec3f_normalize(x, y, z)

  Macro to return the normalized version of a vector.
- #define vec3f_sub_normalize(x1, y1, z1, x2, y2, z2, x3, y3, z3)

  Macro to return the normalized version of a vector minus another vector.
- #define vec3f_rotr_xy(px, py, pz, cx, cy, cz, r)

  Macro to rotate a vector about its origin on the x, y plane.
- #define vec3f_rotr_xz(px, py, pz, cx, cy, cz, r)

  Macro to rotate a vector about its origin on the x, z plane.
- #define vec3f_rotr_yz(px, py, pz, cx, cy, cz, r)

  Macro to rotate a vector about its origin on the y, z plane.
- #define vec3f_rotd_xy(px, py, pz, cx, cy, cz, r)

  Macro to rotate a vector about its origin on the x, y plane.
- #define vec3f_rotd_xz(px, py, pz, cx, cy, cz, r)

  Macro to rotate a vector about its origin on the x, z plane.
- #define vec3f_rotd_yz(px, py, pz, cx, cy, cz, r)

  Macro to rotate a vector about its origin on the y, z plane.

9.248.1 Detailed Description

Basic matrix operations.

This file contains various basic vector math functionality for using the SH4's vector instructions. Higher level functionality in KGL is built off of these.

Author

Josh "PH3NOM" Pearson

See also

dc/matrix.h

9.248.2 Macro Definition Documentation

R_DEG

```
#define R_DEG 182.044444436233495419095237937 43
```

R_RAD

```
#define R_RAD 10430.37835
```

vec3f_distance

```
#define vec3f_distance(
              x1,
              y1,
              z1,
              x2,
              y2,
              z2,
              w )
```

Value:

```
        { \
        register float __x  __asm__("fr0") = (x2-x1); \
        register float __y  __asm__("fr1") = (y2-y1); \
        register float __z  __asm__("fr2") = (z2-z1); \
        register float __w  __asm__("fr3"); \
        __asm__ __volatile__( \
                              "fldi0 fr3\n" \
                              "fipr  fv0,fv0\n" \
                              "fsqrt fr3\n" \
                              : "+f" (__w) \
                              : "f" (__x), "f" (__y), "f" (__z), "f" (__w) \
                             ); \
        w = __w; \
    }
```

Macro to return the Euclidean distance between two 3d vectors.

This macro is an inline assembly operation using the SH4's fast (approximate) math instructions, and returns a single-precision floating-point value.

Parameters

x1	The X coordinate of first vector.
y1	The Y coordinate of first vector.
z1	The Z coordinate of first vector.
x2	The X coordinate of second vector.
y2	The Y coordinate of second vector.
z2	The Z coordinate of second vector.
w	The result of the calculation.

vec3f_dot

```
#define vec3f_dot(
              x1,
              y1,
              z1,
```

```
        x2,
        y2,
        z2,
        w )
```

Value:

```
    { \
    register float __x __asm__("fr0") = (x1); \
    register float __y __asm__("fr1") = (y1); \
    register float __z __asm__("fr2") = (z1); \
    register float __w __asm__("fr3"); \
    register float __a __asm__("fr4") = (x2); \
    register float __b __asm__("fr5") = (y2); \
    register float __c __asm__("fr6") = (z2); \
    register float __d __asm__("fr7"); \
    __asm__ __volatile__( \
                          "fldi0 fr3\n" \
                          "fldi0 fr7\n" \
                          "fipr    fv4,fv0" \
                          : "+f" (__w) \
                          : "f" (__x), "f" (__y), "f" (__z), "f" (__w), \
                          "f" (__a), "f" (__b), "f" (__c), "f" (__d) \
                         ); \
    w = __w; \
}
```

Macro to return the scalar dot product of two 3d vectors.

This macro is an inline assembly operation using the SH4's fast (approximate) math instructions, and returns a single-precision floating-point value.

Parameters

x1	The X coordinate of first vector.
y1	The Y coordinate of first vector.
z1	The Z coordinate of first vector.
x2	The X coordinate of second vector.
y2	The Y coordinate of second vector.
z2	The Z coordinate of second vector.
w	The result of the calculation.

vec3f_length

```
#define vec3f_length(
        x,
        y,
        z,
        w )
```

Value:

```
    { \
    register float __x __asm__("fr0") = (x); \
    register float __y __asm__("fr1") = (y); \
    register float __z __asm__("fr2") = (z); \
    register float __w __asm__("fr3"); \
    __asm__ __volatile__( \
                          "fldi0 fr3\n" \
                          "fipr  fv0,fv0\n" \
                          "fsqrt fr3\n" \
                          : "+f" (__w) \
```

```
                              : "f" (__x), "f" (__y), "f" (__z), "f" (__w) \
                            ); \
        w = __w; \
    }
```

Macro to return scalar Euclidean length of a 3d vector.

This macro is an inline assembly operation using the SH4's fast (approximate) math instructions, and returns a single-precision floating-point value.

Parameters

x	The X coordinate of vector.
y	The Y coordinate of vector.
z	The Z coordinate of vector.
w	The result of the calculation.

vec3f_normalize

```
#define vec3f_normalize(
               x,
               y,
               z )
```

Value:

```
        { \
        register float __x __asm__("fr0") = x; \
        register float __y __asm__("fr1") = y; \
        register float __z __asm__("fr2") = z; \
        __asm__ __volatile__( \
                              "fldi0 fr3\n" \
                              "fipr  fv0,fv0\n" \
                              "fsrra fr3\n" \
                              "fmul  fr3, fr0\n" \
                              "fmul  fr3, fr1\n" \
                              "fmul  fr3, fr2\n" \
                              : "=f" (__x), "=f" (__y), "=f" (__z) \
                              : "0" (__x), "1" (__y), "2" (__z) \
                              : "fr3" ); \
        x = __x; y = __y; z = __z; \
    }
```

Macro to return the normalized version of a vector.

This macro is an inline assembly operation using the SH4's fast (approximate) math instructions to calculate a vector that is in the same direction as the input vector but with a Euclidean length of one. The input vector is modified by the operation as the resulting values.

Parameters

x	The X coordinate of vector.
y	The Y coordinate of vector.
z	The Z coordinate of vector.

vec3f_rotd_xy

```
#define vec3f_rotd_xy(
              px,
              py,
              pz,
              cx,
              cy,
              cz,
              r )
```

Value:

```
        { \
        register float __px __asm__("fr0") = px; \
        register float __pz __asm__("fr1") = pz; \
        register float __cx __asm__("fr4") = cx; \
        register float __cz __asm__("fr5") = cz; \
        register float __r  __asm__("fr6") = r; \
        register float __s __asm__("fr7") = R_DEG; \
        __asm__ __volatile__( \
                              "fmul fr7, fr6\n" \
                              "ftrc fr6, fpul\n" \
                              "fsca fpul, dr6\n" \
                              "fsub fr4, fr0\n" \
                              "fsub fr5, fr1\n" \
                              "fmov fr0, fr2\n" \
                              "fmov fr1, fr3\n" \
                              "fmul fr7, fr0\n" \
                              "fmul fr6, fr1\n" \
                              "fmul fr6, fr2\n" \
                              "fmul fr7, fr3\n" \
                              "fadd fr0, fr4\n" \
                              "fsub fr1, fr4\n" \
                              "fadd fr2, fr5\n" \
                              "fadd fr3, fr5\n" \
                              : "+f" (__cx), "+f" (__cz) \
                              : "f" (__px), "f" (__pz), "f" (__r), "f" (__s) ); \
        px = __cx; pz = __cz; \
    }
```

Macro to rotate a vector about its origin on the x, y plane.

This macro is an inline assembly operation using the SH4's fast (approximate) math instructions. The return vector is stored into the first vertex parameter: x1, y1, and z1.

Parameters

px	The X coordinate of vector to rotate.
py	The Y coordinate of vector to rotate.
pz	The Z coordinate of vector to rotate.
cx	The X coordinate of origin vector.
cy	The Y coordinate of origin vector.
cz	The Z coordinate of origin vector.
r	The angle (in degrees) of rotation.

vec3f_rotd_xz

```
#define vec3f_rotd_xz(
              px,
```

```
            py,
            pz,
            cx,
            cy,
            cz,
            r )
```

Value:

```
    { \
    register float __px __asm__("fr0") = px; \
    register float __pz __asm__("fr1") = pz; \
    register float __cx __asm__("fr4") = cx; \
    register float __cz __asm__("fr5") = cz; \
    register float __r  __asm__("fr6") = r; \
    register float __s __asm__("fr7") = R_DEG; \
    __asm__ __volatile__( \
                        "fmul fr7, fr6\n" \
                        "ftrc fr6, fpul\n" \
                        "fsca fpul, dr6\n" \
                        "fsub fr4, fr0\n" \
                        "fsub fr5, fr1\n" \
                        "fmov fr0, fr2\n" \
                        "fmov fr1, fr3\n" \
                        "fmul fr7, fr0\n" \
                        "fmul fr6, fr1\n" \
                        "fmul fr6, fr2\n" \
                        "fmul fr7, fr3\n" \
                        "fadd fr0, fr4\n" \
                        "fsub fr1, fr4\n" \
                        "fadd fr2, fr5\n" \
                        "fadd fr3, fr5\n" \
                        : "+f" (__cx), "+f" (__cz) \
                        : "f" (__px), "f" (__pz), "f" (__r), "f" (__s) ); \
    px = __cx; pz = __cz; \
}
```

Macro to rotate a vector about its origin on the x, z plane.

This macro is an inline assembly operation using the SH4's fast (approximate) math instructions. The return vector is stored into the first vertex parameter: x1, y1, and z1.

Parameters

px	The X coordinate of vector to rotate.
py	The Y coordinate of vector to rotate.
pz	The Z coordinate of vector to rotate.
cx	The X coordinate of origin vector.
cy	The Y coordinate of origin vector.
cz	The Z coordinate of origin vector.
r	The angle (in degrees) of rotation.

vec3f_rotd_yz

```
#define vec3f_rotd_yz(
            px,
            py,
            pz,
            cx,
            cy,
```

```
            cz,
            r )
```

Value:

```
    { \
    register float __py __asm__("fr0") = py; \
    register float __pz __asm__("fr1") = pz; \
    register float __cy __asm__("fr4") = cy; \
    register float __cz __asm__("fr5") = cz; \
    register float __r  __asm__("fr6") = r; \
    register float __s __asm__("fr7") = R_DEG; \
    __asm__ __volatile__( \
                        "fmul fr7, fr6\n" \
                        "ftrc fr6, fpul\n" \
                        "fsca fpul, dr6\n" \
                        "fsub fr4, fr0\n" \
                        "fsub fr5, fr1\n" \
                        "fmov fr0, fr2\n" \
                        "fmov fr1, fr3\n" \
                        "fmul fr7, fr0\n" \
                        "fmul fr6, fr1\n" \
                        "fmul fr6, fr2\n" \
                        "fmul fr7, fr3\n" \
                        "fadd fr0, fr4\n" \
                        "fsub fr1, fr4\n" \
                        "fadd fr2, fr5\n" \
                        "fadd fr3, fr5\n" \
                        : "+f" (__cy), "+f" (__cz) \
                        : "f" (__py), "f" (__pz), "f" (__r), "f" (__s) ); \
    py = __cy; pz = __cz; \
}
```

Macro to rotate a vector about its origin on the y, z plane.

This macro is an inline assembly operation using the SH4's fast (approximate) math instructions. The return vector is stored into the first vertex parameter: x1, y1, and z1.

Parameters

px	The X coordinate of vector to rotate.
py	The Y coordinate of vector to rotate.
pz	The Z coordinate of vector to rotate.
cx	The X coordinate of origin vector.
cy	The Y coordinate of origin vector.
cz	The Z coordinate of origin vector.
r	The angle (in degrees) of rotation.

vec3f_rotr_xy

```
#define vec3f_rotr_xy(
            px,
            py,
            pz,
            cx,
            cy,
            cz,
            r )
```

Value:

```
        { \
        register float __px __asm__("fr0") = px; \
        register float __py __asm__("fr1") = py; \
        register float __cx __asm__("fr4") = cx; \
        register float __cy __asm__("fr5") = cy; \
        register float __r  __asm__("fr6") = r; \
        register float __s __asm__("fr7") = R_RAD; \
        __asm__ __volatile__( \
                              "fmul fr7, fr6\n" \
                              "ftrc fr6, fpul\n" \
                              "fsca fpul, dr6\n" \
                              "fsub fr4, fr0\n" \
                              "fsub fr5, fr1\n" \
                              "fmov fr0, fr2\n" \
                              "fmov fr1, fr3\n" \
                              "fmul fr7, fr0\n" \
                              "fmul fr6, fr1\n" \
                              "fmul fr6, fr2\n" \
                              "fmul fr7, fr3\n" \
                              "fadd fr0, fr4\n" \
                              "fsub fr1, fr4\n" \
                              "fadd fr2, fr5\n" \
                              "fadd fr3, fr5\n" \
                              : "+f" (__cx), "+f" (__cy) \
                              : "f" (__px), "f" (__py), "f" (__r), "f" (__s) ); \
        px = __cx; py = __cy; \
    }
```

Macro to rotate a vector about its origin on the x, y plane.

This macro is an inline assembly operation using the SH4's fast (approximate) math instructions. The return vector is stored into the first vertex parameter: x1, y1, and z1.

Parameters

px	The X coordinate of vector to rotate.
py	The Y coordinate of vector to rotate.
pz	The Z coordinate of vector to rotate.
cx	The X coordinate of origin vector.
cy	The Y coordinate of origin vector.
cz	The Z coordinate of origin vector.
r	The angle (in radians) of rotation.

vec3f_rotr_xz

```
#define vec3f_rotr_xz(
              px,
              py,
              pz,
              cx,
              cy,
              cz,
              r )
```

Value:

```
        { \
        register float __px __asm__("fr0") = px; \
        register float __pz __asm__("fr1") = pz; \
        register float __cx __asm__("fr4") = cx; \
        register float __cz __asm__("fr5") = cz; \
```

```
        register float __r  __asm__("fr6") = r; \
        register float __s __asm__("fr7") = R_RAD; \
        __asm__ __volatile__( \
                              "fmul fr7, fr6\n" \
                              "ftrc fr6, fpul\n" \
                              "fsca fpul, dr6\n" \
                              "fsub fr4, fr0\n" \
                              "fsub fr5, fr1\n" \
                              "fmov fr0, fr2\n" \
                              "fmov fr1, fr3\n" \
                              "fmul fr7, fr0\n" \
                              "fmul fr6, fr1\n" \
                              "fmul fr6, fr2\n" \
                              "fmul fr7, fr3\n" \
                              "fadd fr0, fr4\n" \
                              "fsub fr1, fr4\n" \
                              "fadd fr2, fr5\n" \
                              "fadd fr3, fr5\n" \
                              : "+f" (__cx), "+f" (__cz) \
                              : "f" (__px), "f" (__pz), "f" (__r), "f" (__s) ); \
        px = __cx; pz = __cz; \
    }
```

Macro to rotate a vector about its origin on the x, z plane.

This macro is an inline assembly operation using the SH4's fast (approximate) math instructions. The return vector is stored into the first vertex parameter: x1, y1, and z1.

Parameters

px	The X coordinate of vector to rotate.
py	The Y coordinate of vector to rotate.
pz	The Z coordinate of vector to rotate.
cx	The X coordinate of origin vector.
cy	The Y coordinate of origin vector.
cz	The Z coordinate of origin vector.
r	The angle (in radians) of rotation.

vec3f_rotr_yz

```
#define vec3f_rotr_yz(
              px,
              py,
              pz,
              cx,
              cy,
              cz,
              r )
```

Value:

```
        { \
        register float __py __asm__("fr0") = py; \
        register float __pz __asm__("fr1") = pz; \
        register float __cy __asm__("fr4") = cy; \
        register float __cz __asm__("fr5") = cz; \
        register float __r  __asm__("fr6") = r; \
        register float __s __asm__("fr7") = R_RAD; \
        __asm__ __volatile__( \
                              "fmul fr7, fr6\n" \
                              "ftrc fr6, fpul\n" \
```

```
                            "fsca fpul, dr6\n" \
                            "fsub fr4, fr0\n" \
                            "fsub fr5, fr1\n" \
                            "fmov fr0, fr2\n" \
                            "fmov fr1, fr3\n" \
                            "fmul fr7, fr0\n" \
                            "fmul fr6, fr1\n" \
                            "fmul fr6, fr2\n" \
                            "fmul fr7, fr3\n" \
                            "fadd fr0, fr4\n" \
                            "fsub fr1, fr4\n" \
                            "fadd fr2, fr5\n" \
                            "fadd fr3, fr5\n" \
                            : "+f" (__cy), "+f" (__cz) \
                            : "f" (__py), "f" (__pz), "f" (__r), "f" (__s) ); \
        py = __cy; pz = __cz; \
    }
```

Macro to rotate a vector about its origin on the y, z plane.

This macro is an inline assembly operation using the SH4's fast (approximate) math instructions. The return vector is stored into the first vertex parameter: x1, y1, and z1.

Parameters

px	The X coordinate of vector to rotate.
py	The Y coordinate of vector to rotate.
pz	The Z coordinate of vector to rotate.
cx	The X coordinate of origin vector.
cy	The Y coordinate of origin vector.
cz	The Z coordinate of origin vector.
r	The angle (in radians) of rotation.

vec3f_sub_normalize

```
#define vec3f_sub_normalize(
              x1,
              y1,
              z1,
              x2,
              y2,
              z2,
              x3,
              y3,
              z3 )
```

Value:

```
        { \
        register float __x __asm__("fr0") = x1 - x2; \
        register float __y __asm__("fr1") = y1 - y2; \
        register float __z __asm__("fr2") = z1 - z2; \
        __asm__ __volatile__( \
                            "fldi0 fr3\n" \
                            "fipr  fv0,fv0\n" \
                            "fsrra fr3\n" \
                            "fmul  fr3, fr0\n" \
                            "fmul  fr3, fr1\n" \
                            "fmul  fr3, fr2\n" \
                            : "=f" (__x), "=f" (__y), "=f" (__z) \
```

```
                                : "0" (__x), "1" (__y), "2" (__z) \
                                : "fr3" ); \
        x3 = __x; y3 = __y; z3 = __z; \
    }
```

Macro to return the normalized version of a vector minus another vector.

This macro is an inline assembly operation using the SH4's fast (approximate) math instructions. The return vector is stored into the third vertex parameter: x3, y3, and z3.

Parameters

x1	The X coordinate of first vector.
y1	The Y coordinate of first vector.
z1	The Z coordinate of first vector.
x2	The X coordinate of second vector.
y2	The Y coordinate of second vector.
z2	The Z coordinate of second vector.
x3	The X coordinate of output vector.
y3	The Y coordinate of output vector.
z3	The Z coordinate of output vector.

9.249 vec3f.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/vec3f.h
00004    Copyright (C) 2013, 2014 Josh "PH3NOM" Pearson
00005
00006 */
00007
00008 /** \file   dc/vec3f.h
00009     \brief  Basic matrix operations.
00010
00011     This file contains various basic vector math functionality for using the
00012     SH4's vector instructions. Higher level functionality in KGL is built off
00013     of these.
00014
00015     \author Josh "PH3NOM" Pearson
00016     \see    dc/matrix.h
00017 */
00018
00019 #ifndef __DC_VEC3F_H
00020 #define __DC_VEC3F_H
00021
00022 #include <sys/cdefs.h>
00023 __BEGIN_DECLS
00024
00025 typedef struct vec3f {
00026     float x, y, z;
00027 } vec3f_t;
00028
00029 #define R_DEG 182.044444436233495419095237937 43
00030 #define R_RAD 10430.37835
00031
00032 /** \brief  Macro to return the scalar dot product of two 3d vectors.
00033
00034     This macro is an inline assembly operation using the SH4's fast
00035     (approximate) math instructions, and returns a single-precision
00036     floating-point value.
00037
00038     \param  x1              The X coordinate of first vector.
```

```
00039     \param  y1              The Y coordinate of first vector.
00040     \param  z1              The Z coordinate of first vector.
00041     \param  x2              The X coordinate of second vector.
00042     \param  y2              The Y coordinate of second vector.
00043     \param  z2              The Z coordinate of second vector.
00044     \param  w               The result of the calculation.
00045 */
00046 #define vec3f_dot(x1, y1, z1, x2, y2, z2, w) { \
00047         register float __x __asm__("fr0") = (x1); \
00048         register float __y __asm__("fr1") = (y1); \
00049         register float __z __asm__("fr2") = (z1); \
00050         register float __w __asm__("fr3"); \
00051         register float __a __asm__("fr4") = (x2); \
00052         register float __b __asm__("fr5") = (y2); \
00053         register float __c __asm__("fr6") = (z2); \
00054         register float __d __asm__("fr7"); \
00055         __asm__ __volatile__( \
00056                                 "fldi0 fr3\n" \
00057                                 "fldi0 fr7\n" \
00058                                 "fipr    fv4,fv0" \
00059                                 : "+f" (__w) \
00060                                 : "f" (__x), "f" (__y), "f" (__z), "f" (__w), \
00061                                 "f" (__a), "f" (__b), "f" (__c), "f" (__d) \
00062                               ); \
00063         w = __w; \
00064     }
00065
00066 /** \brief  Macro to return scalar Euclidean length of a 3d vector.
00067
00068     This macro is an inline assembly operation using the SH4's fast
00069     (approximate) math instructions, and returns a single-precision
00070     floating-point value.
00071
00072     \param  x               The X coordinate of vector.
00073     \param  y               The Y coordinate of vector.
00074     \param  z               The Z coordinate of vector.
00075     \param  w               The result of the calculation.
00076 */
00077 #define vec3f_length(x, y, z, w) { \
00078         register float __x __asm__("fr0") = (x); \
00079         register float __y __asm__("fr1") = (y); \
00080         register float __z __asm__("fr2") = (z); \
00081         register float __w __asm__("fr3"); \
00082         __asm__ __volatile__( \
00083                                 "fldi0 fr3\n" \
00084                                 "fipr  fv0,fv0\n" \
00085                                 "fsqrt fr3\n" \
00086                                 : "+f" (__w) \
00087                                 : "f" (__x), "f" (__y), "f" (__z), "f" (__w) \
00088                               ); \
00089         w = __w; \
00090     }
00091
00092 /** \brief  Macro to return the Euclidean distance between two 3d vectors.
00093
00094     This macro is an inline assembly operation using the SH4's fast
00095     (approximate) math instructions, and returns a single-precision
00096     floating-point value.
00097
00098     \param  x1              The X coordinate of first vector.
00099     \param  y1              The Y coordinate of first vector.
00100     \param  z1              The Z coordinate of first vector.
00101     \param  x2              The X coordinate of second vector.
00102     \param  y2              The Y coordinate of second vector.
00103     \param  z2              The Z coordinate of second vector.
00104     \param  w               The result of the calculation.
00105 */
00106 #define vec3f_distance(x1, y1, z1, x2, y2, z2, w) { \
00107         register float __x  __asm__("fr0") = (x2-x1); \
00108         register float __y  __asm__("fr1") = (y2-y1); \
00109         register float __z  __asm__("fr2") = (z2-z1); \
00110         register float __w  __asm__("fr3"); \
00111         __asm__ __volatile__( \
00112                                 "fldi0 fr3\n" \
00113                                 "fipr  fv0,fv0\n" \
00114                                 "fsqrt fr3\n" \
00115                                 : "+f" (__w) \
00116                                 : "f" (__x), "f" (__y), "f" (__z), "f" (__w) \
00117                               ); \
00118         w = __w; \
00119     }
```

```
00120
00121 /** \brief  Macro to return the normalized version of a vector.
00122
00123     This macro is an inline assembly operation using the SH4's fast
00124     (approximate) math instructions to calculate a vector that is in the same
00125     direction as the input vector but with a Euclidean length of one. The input
00126     vector is modified by the operation as the resulting values.
00127
00128     \param  x               The X coordinate of vector.
00129     \param  y               The Y coordinate of vector.
00130     \param  z               The Z coordinate of vector.
00131 */
00132 #define vec3f_normalize(x, y, z) { \
00133         register float __x __asm__("fr0") = x; \
00134         register float __y __asm__("fr1") = y; \
00135         register float __z __asm__("fr2") = z; \
00136         __asm__ __volatile__( \
00137                                 "fldi0 fr3\n" \
00138                                 "fipr  fv0,fv0\n" \
00139                                 "fsrra fr3\n" \
00140                                 "fmul  fr3, fr0\n" \
00141                                 "fmul  fr3, fr1\n" \
00142                                 "fmul  fr3, fr2\n" \
00143                                 : "=f" (__x), "=f" (__y), "=f" (__z) \
00144                                 : "0" (__x), "1" (__y), "2" (__z) \
00145                                 : "fr3" ); \
00146         x = __x; y = __y; z = __z; \
00147     }
00148
00149 /** \brief  Macro to return the normalized version of a vector minus another
00150             vector.
00151
00152     This macro is an inline assembly operation using the SH4's fast
00153     (approximate) math instructions. The return vector is stored into the third
00154     vertex parameter: x3, y3, and z3.
00155
00156     \param  x1              The X coordinate of first vector.
00157     \param  y1              The Y coordinate of first vector.
00158     \param  z1              The Z coordinate of first vector.
00159     \param  x2              The X coordinate of second vector.
00160     \param  y2              The Y coordinate of second vector.
00161     \param  z2              The Z coordinate of second vector.
00162     \param  x3              The X coordinate of output vector.
00163     \param  y3              The Y coordinate of output vector.
00164     \param  z3              The Z coordinate of output vector.
00165 */
00166 #define vec3f_sub_normalize(x1, y1, z1, x2, y2, z2, x3, y3, z3) { \
00167         register float __x __asm__("fr0") = x1 - x2; \
00168         register float __y __asm__("fr1") = y1 - y2; \
00169         register float __z __asm__("fr2") = z1 - z2; \
00170         __asm__ __volatile__( \
00171                                 "fldi0 fr3\n" \
00172                                 "fipr  fv0,fv0\n" \
00173                                 "fsrra fr3\n" \
00174                                 "fmul  fr3, fr0\n" \
00175                                 "fmul  fr3, fr1\n" \
00176                                 "fmul  fr3, fr2\n" \
00177                                 : "=f" (__x), "=f" (__y), "=f" (__z) \
00178                                 : "0" (__x), "1" (__y), "2" (__z) \
00179                                 : "fr3" ); \
00180         x3 = __x; y3 = __y; z3 = __z; \
00181     }
00182
00183 /** \brief  Macro to rotate a vector about its origin on the x, y plane.
00184
00185     This macro is an inline assembly operation using the SH4's fast
00186     (approximate) math instructions. The return vector is stored into the first
00187     vertex parameter: x1, y1, and z1.
00188
00189     \param  px              The X coordinate of vector to rotate.
00190     \param  py              The Y coordinate of vector to rotate.
00191     \param  pz              The Z coordinate of vector to rotate.
00192     \param  cx              The X coordinate of origin vector.
00193     \param  cy              The Y coordinate of origin vector.
00194     \param  cz              The Z coordinate of origin vector.
00195     \param  r               The angle (in radians) of rotation.
00196 */
00197 #define vec3f_rotr_xy(px, py, pz, cx, cy, cz, r) { \
00198         register float __px __asm__("fr0") = px; \
00199         register float __py __asm__("fr1") = py; \
00200         register float __cx __asm__("fr4") = cx; \
```

```
00201         register float __cy __asm__("fr5") = cy; \
00202         register float __r  __asm__("fr6") = r; \
00203         register float __s __asm__("fr7") = R_RAD; \
00204         __asm__ __volatile__( \
00205                                 "fmul fr7, fr6\n" \
00206                                 "ftrc fr6, fpul\n" \
00207                                 "fsca fpul, dr6\n" \
00208                                 "fsub fr4, fr0\n" \
00209                                 "fsub fr5, fr1\n" \
00210                                 "fmov fr0, fr2\n" \
00211                                 "fmov fr1, fr3\n" \
00212                                 "fmul fr7, fr0\n" \
00213                                 "fmul fr6, fr1\n" \
00214                                 "fmul fr6, fr2\n" \
00215                                 "fmul fr7, fr3\n" \
00216                                 "fadd fr0, fr4\n" \
00217                                 "fsub fr1, fr4\n" \
00218                                 "fadd fr2, fr5\n" \
00219                                 "fadd fr3, fr5\n" \
00220                                 : "+f" (__cx), "+f" (__cy) \
00221                                 : "f" (__px), "f" (__py), "f" (__r), "f" (__s) ); \
00222         px = __cx; py = __cy; \
00223     }
00224
00225 /** \brief  Macro to rotate a vector about its origin on the x, z plane.
00226
00227     This macro is an inline assembly operation using the SH4's fast
00228     (approximate) math instructions. The return vector is stored into the first
00229     vertex parameter: x1, y1, and z1.
00230
00231     \param  px              The X coordinate of vector to rotate.
00232     \param  py              The Y coordinate of vector to rotate.
00233     \param  pz              The Z coordinate of vector to rotate.
00234     \param  cx              The X coordinate of origin vector.
00235     \param  cy              The Y coordinate of origin vector.
00236     \param  cz              The Z coordinate of origin vector.
00237     \param  r               The angle (in radians) of rotation.
00238 */
00239 #define vec3f_rotr_xz(px, py, pz, cx, cy, cz, r) { \
00240         register float __px __asm__("fr0") = px; \
00241         register float __pz __asm__("fr1") = pz; \
00242         register float __cx __asm__("fr4") = cx; \
00243         register float __cz __asm__("fr5") = cz; \
00244         register float __r  __asm__("fr6") = r; \
00245         register float __s __asm__("fr7") = R_RAD; \
00246         __asm__ __volatile__( \
00247                                 "fmul fr7, fr6\n" \
00248                                 "ftrc fr6, fpul\n" \
00249                                 "fsca fpul, dr6\n" \
00250                                 "fsub fr4, fr0\n" \
00251                                 "fsub fr5, fr1\n" \
00252                                 "fmov fr0, fr2\n" \
00253                                 "fmov fr1, fr3\n" \
00254                                 "fmul fr7, fr0\n" \
00255                                 "fmul fr6, fr1\n" \
00256                                 "fmul fr6, fr2\n" \
00257                                 "fmul fr7, fr3\n" \
00258                                 "fadd fr0, fr4\n" \
00259                                 "fsub fr1, fr4\n" \
00260                                 "fadd fr2, fr5\n" \
00261                                 "fadd fr3, fr5\n" \
00262                                 : "+f" (__cx), "+f" (__cz) \
00263                                 : "f" (__px), "f" (__pz), "f" (__r), "f" (__s) ); \
00264         px = __cx; pz = __cz; \
00265     }
00266
00267 /** \brief  Macro to rotate a vector about its origin on the y, z plane.
00268
00269     This macro is an inline assembly operation using the SH4's fast
00270     (approximate) math instructions. The return vector is stored into the first
00271     vertex parameter: x1, y1, and z1.
00272
00273     \param  px              The X coordinate of vector to rotate.
00274     \param  py              The Y coordinate of vector to rotate.
00275     \param  pz              The Z coordinate of vector to rotate.
00276     \param  cx              The X coordinate of origin vector.
00277     \param  cy              The Y coordinate of origin vector.
00278     \param  cz              The Z coordinate of origin vector.
00279     \param  r               The angle (in radians) of rotation.
00280 */
00281 #define vec3f_rotr_yz(px, py, pz, cx, cy, cz, r) { \
```

```
00282         register float __py __asm__("fr0") = py; \
00283         register float __pz __asm__("fr1") = pz; \
00284         register float __cy __asm__("fr4") = cy; \
00285         register float __cz __asm__("fr5") = cz; \
00286         register float __r  __asm__("fr6") = r; \
00287         register float __s __asm__("fr7") = R_RAD; \
00288         __asm__ __volatile__( \
00289                               "fmul fr7, fr6\n" \
00290                               "ftrc fr6, fpul\n" \
00291                               "fsca fpul, dr6\n" \
00292                               "fsub fr4, fr0\n" \
00293                               "fsub fr5, fr1\n" \
00294                               "fmov fr0, fr2\n" \
00295                               "fmov fr1, fr3\n" \
00296                               "fmul fr7, fr0\n" \
00297                               "fmul fr6, fr1\n" \
00298                               "fmul fr6, fr2\n" \
00299                               "fmul fr7, fr3\n" \
00300                               "fadd fr0, fr4\n" \
00301                               "fsub fr1, fr4\n" \
00302                               "fadd fr2, fr5\n" \
00303                               "fadd fr3, fr5\n" \
00304                               : "+f" (__cy), "+f" (__cz) \
00305                               : "f" (__py), "f" (__pz), "f" (__r), "f" (__s) ); \
00306         py = __cy; pz = __cz; \
00307     }
00308
00309 /** \brief  Macro to rotate a vector about its origin on the x, y plane.
00310
00311     This macro is an inline assembly operation using the SH4's fast
00312     (approximate) math instructions. The return vector is stored into the first
00313     vertex parameter: x1, y1, and z1.
00314
00315     \param  px              The X coordinate of vector to rotate.
00316     \param  py              The Y coordinate of vector to rotate.
00317     \param  pz              The Z coordinate of vector to rotate.
00318     \param  cx              The X coordinate of origin vector.
00319     \param  cy              The Y coordinate of origin vector.
00320     \param  cz              The Z coordinate of origin vector.
00321     \param  r               The angle (in degrees) of rotation.
00322 */
00323 #define vec3f_rotd_xy(px, py, pz, cx, cy, cz, r) { \
00324         register float __px __asm__("fr0") = px; \
00325         register float __pz __asm__("fr1") = pz; \
00326         register float __cx __asm__("fr4") = cx; \
00327         register float __cz __asm__("fr5") = cz; \
00328         register float __r  __asm__("fr6") = r; \
00329         register float __s __asm__("fr7") = R_DEG; \
00330         __asm__ __volatile__( \
00331                               "fmul fr7, fr6\n" \
00332                               "ftrc fr6, fpul\n" \
00333                               "fsca fpul, dr6\n" \
00334                               "fsub fr4, fr0\n" \
00335                               "fsub fr5, fr1\n" \
00336                               "fmov fr0, fr2\n" \
00337                               "fmov fr1, fr3\n" \
00338                               "fmul fr7, fr0\n" \
00339                               "fmul fr6, fr1\n" \
00340                               "fmul fr6, fr2\n" \
00341                               "fmul fr7, fr3\n" \
00342                               "fadd fr0, fr4\n" \
00343                               "fsub fr1, fr4\n" \
00344                               "fadd fr2, fr5\n" \
00345                               "fadd fr3, fr5\n" \
00346                               : "+f" (__cx), "+f" (__cz) \
00347                               : "f" (__px), "f" (__pz), "f" (__r), "f" (__s) ); \
00348         px = __cx; pz = __cz; \
00349     }
00350
00351 /** \brief  Macro to rotate a vector about its origin on the x, z plane.
00352
00353     This macro is an inline assembly operation using the SH4's fast
00354     (approximate) math instructions. The return vector is stored into the first
00355     vertex parameter: x1, y1, and z1.
00356
00357     \param  px              The X coordinate of vector to rotate.
00358     \param  py              The Y coordinate of vector to rotate.
00359     \param  pz              The Z coordinate of vector to rotate.
00360     \param  cx              The X coordinate of origin vector.
00361     \param  cy              The Y coordinate of origin vector.
00362     \param  cz              The Z coordinate of origin vector.
```

```
00363     \param  r               The angle (in degrees) of rotation.
00364 */
00365 #define vec3f_rotd_xz(px, py, pz, cx, cy, cz, r) { \
00366         register float __px __asm__("fr0") = px; \
00367         register float __pz __asm__("fr1") = pz; \
00368         register float __cx __asm__("fr4") = cx; \
00369         register float __cz __asm__("fr5") = cz; \
00370         register float __r  __asm__("fr6") = r; \
00371         register float __s __asm__("fr7") = R_DEG; \
00372         __asm__ __volatile__( \
00373                                 "fmul fr7, fr6\n" \
00374                                 "ftrc fr6, fpul\n" \
00375                                 "fsca fpul, dr6\n" \
00376                                 "fsub fr4, fr0\n" \
00377                                 "fsub fr5, fr1\n" \
00378                                 "fmov fr0, fr2\n" \
00379                                 "fmov fr1, fr3\n" \
00380                                 "fmul fr7, fr0\n" \
00381                                 "fmul fr6, fr1\n" \
00382                                 "fmul fr6, fr2\n" \
00383                                 "fmul fr7, fr3\n" \
00384                                 "fadd fr0, fr4\n" \
00385                                 "fsub fr1, fr4\n" \
00386                                 "fadd fr2, fr5\n" \
00387                                 "fadd fr3, fr5\n" \
00388                                 : "+f" (__cx), "+f" (__cz) \
00389                                 : "f" (__px), "f" (__pz), "f" (__r), "f" (__s) ); \
00390         px = __cx; pz = __cz; \
00391     }
00392
00393 /** \brief  Macro to rotate a vector about its origin on the y, z plane.
00394
00395     This macro is an inline assembly operation using the SH4's fast
00396     (approximate) math instructions. The return vector is stored into the first
00397     vertex parameter: x1, y1, and z1.
00398
00399     \param  px              The X coordinate of vector to rotate.
00400     \param  py              The Y coordinate of vector to rotate.
00401     \param  pz              The Z coordinate of vector to rotate.
00402     \param  cx              The X coordinate of origin vector.
00403     \param  cy              The Y coordinate of origin vector.
00404     \param  cz              The Z coordinate of origin vector.
00405     \param  r               The angle (in degrees) of rotation.
00406 */
00407 #define vec3f_rotd_yz(px, py, pz, cx, cy, cz, r) { \
00408         register float __py __asm__("fr0") = py; \
00409         register float __pz __asm__("fr1") = pz; \
00410         register float __cy __asm__("fr4") = cy; \
00411         register float __cz __asm__("fr5") = cz; \
00412         register float __r  __asm__("fr6") = r; \
00413         register float __s __asm__("fr7") = R_DEG; \
00414         __asm__ __volatile__( \
00415                                 "fmul fr7, fr6\n" \
00416                                 "ftrc fr6, fpul\n" \
00417                                 "fsca fpul, dr6\n" \
00418                                 "fsub fr4, fr0\n" \
00419                                 "fsub fr5, fr1\n" \
00420                                 "fmov fr0, fr2\n" \
00421                                 "fmov fr1, fr3\n" \
00422                                 "fmul fr7, fr0\n" \
00423                                 "fmul fr6, fr1\n" \
00424                                 "fmul fr6, fr2\n" \
00425                                 "fmul fr7, fr3\n" \
00426                                 "fadd fr0, fr4\n" \
00427                                 "fsub fr1, fr4\n" \
00428                                 "fadd fr2, fr5\n" \
00429                                 "fadd fr3, fr5\n" \
00430                                 : "+f" (__cy), "+f" (__cz) \
00431                                 : "f" (__py), "f" (__pz), "f" (__r), "f" (__s) ); \
00432         py = __cy; pz = __cz; \
00433     }
00434
00435 __END_DECLS
00436
00437 #endif /* !__DC_VEC3F_H */
```

9.250 addons/include/kos/vector.h File Reference

Deprecated alias for <dc/vector.h>.

```
#include <dc/vector.h>
```

9.250.1 Detailed Description

Deprecated alias for <dc/vector.h>.

This file is provided for backwards compatibility. New code should include <dc/vector.h> directly.

9.251 vector.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    kos/vector.h
00004    Copyright (C) 2002 Megan Potter
00005
00006 */
00007
00008 /** \file   kos/vector.h
00009     \brief  Deprecated alias for <dc/vector.h>.
00010
00011     This file is provided for backwards compatibility. New code should include
00012     <dc/vector.h> directly.
00013 */
00014
00015 #include <dc/vector.h>
00016
```

9.252 kernel/arch/dreamcast/include/dc/vector.h File Reference

Primitive matrix, vector, and point types.

```
#include <sys/cdefs.h>
```

Data Structures

- struct vector_t

  4-part vector type.

Typedefs

- typedef vector_t point_t

  4-part point type (alias to the vector_t type).

Functions

- typedef __attribute__ ((aligned(8))) float matrix_t[4][4]

  Basic 4x4 matrix type.

9.252.1 Detailed Description

Primitive matrix, vector, and point types.

This file provides a few primivite data types that are useful for 3D graphics.

Author

Megan Potter

9.252.2 Typedef Documentation

point_t

```
typedef vector_t point_t
```

4-part point type (alias to the vector_t type).

9.252.3 Function Documentation

__attribute__()

```
typedef __attribute__ (
            (aligned(8))  )
```

Basic 4x4 matrix type.

Warning

This type must always be allocated on 8-byte boundaries, or else the API operating on it will crash on unaligned accesses. Keep this in mind with heap allocation, where you must ensure alignment manually.

9.253 vector.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/vector.h
00004    Copyright (C) 2002 Megan Potter
00005
00006 */
00007
00008 #ifndef __DC_VECTOR_H
00009 #define __DC_VECTOR_H
00010
00011 /** \file   dc/vector.h
00012     \brief  Primitive matrix, vector, and point types.
00013
00014     This file provides a few primivite data types that are useful for 3D
00015     graphics.
00016
00017     \author Megan Potter
00018 */
00019
00020 #include <sys/cdefs.h>
00021 __BEGIN_DECLS
00022
00023 /** \brief  Basic 4x4 matrix type.
00024     \headerfile dc/vector.h
00025
00026     \warning
00027     This type must always be allocated on 8-byte boundaries,
00028     or else the API operating on it will crash on unaligned
00029     accesses. Keep this in mind with heap allocation, where
00030     you must ensure alignment manually.
00031 */
00032 typedef __attribute__ ((aligned (8))) float matrix_t[4][4];
00033
00034 /** \brief  4-part vector type.
00035     \headerfile dc/vector.h
00036 */
00037 typedef struct vectorstr {
00038     float x, y, z, w;
00039 } vector_t;
00040
00041 /** \brief  4-part point type (alias to the vector_t type).
00042     \headerfile dc/vector.h
00043 */
00044 typedef vector_t point_t;
00045
00046 __END_DECLS
00047
00048 #endif  /* __DC_VECTOR_H */
00049
```

9.254 kernel/arch/dreamcast/include/dc/video.h File Reference

Functions related to video output.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Data Structures

- struct vid_mode_t

  Video mode structure.

Macros

- #define CT_ANY -1

  Any cable type. Used only internally.
- #define CT_VGA 0

  VGA Box.
- #define CT_NONE 1

  Nothing connected.
- #define CT_RGB 2

  RGB/SCART cable.
- #define CT_COMPOSITE 3

  Composite cable or RF switch.
- #define PM_RGB555 0

  RGB555 pixel mode (15-bit)
- #define PM_RGB565 1

  RGB565 pixel mode (16-bit)
- #define PM_RGB888P 2

  RBG888 packed pixel mode (24-bit)
- #define PM_RGB0888 3

  RGB0888 pixel mode (32-bit)
- #define PM_RGB888 PM_RGB0888

  Backwards compatibility support.
- #define DM_MULTIBUFFER 0x2000

  Multi-buffered mode setting.
- #define VID_MAX_FB 4

  The maximum number of framebuffers available.
- #define VID_INTERLACE 0x00000001

  Interlaced display.
- #define VID_LINEDOUBLE 0x00000002

  Display each scanline twice.
- #define VID_PIXELDOUBLE 0x00000004

  Display each pixel twice.
- #define VID_PAL 0x00000008

  50Hz refresh rate, if not VGA

Enumerations

- enum {
  DM_GENERIC_FIRST = 0x1000 , DM_320x240 = 0x1000 , DM_640x480 , DM_800x608 ,
  DM_256x256 , DM_768x480 , DM_768x576 , DM_GENERIC_LAST = DM_768x576 }

  Generic display modes.
- enum {
  DM_INVALID = 0 , DM_320x240_VGA = 1 , DM_320x240_NTSC , DM_640x480_VGA ,
  DM_640x480_NTSC_IL , DM_800x608_VGA , DM_640x480_PAL_IL , DM_256x256_PAL_IL ,
  DM_768x480_NTSC_IL , DM_768x576_PAL_IL , DM_768x480_PAL_IL , DM_320x240_PAL ,
  DM_320x240_VGA_MB , DM_320x240_NTSC_MB , DM_640x480_VGA_MB , DM_640x480_NTSC_IL_MB ,
  DM_800x608_VGA_MB , DM_640x480_PAL_IL_MB , DM_256x256_PAL_IL_MB , DM_768x480_NTSC_IL_MB ,
  DM_768x576_PAL_IL_MB , DM_768x480_PAL_IL_MB , DM_320x240_PAL_MB , DM_SENTINEL ,
  DM_MODE_COUNT }

  Specific display modes.

Functions

- int vid_check_cable (void)

  Retrieve the connected video cable type.
- void vid_set_start (uint32 base)

  Set the VRAM base of the framebuffer.
- void vid_flip (int fb)

  Set the current framebuffer in a multibuffered setup.
- uint32 vid_border_color (int r, int g, int b)

  Set the border color of the display.
- void vid_clear (int r, int g, int b)

  Clear the display.
- void vid_empty (void)

  Clear VRAM.
- void vid_waitvbl (void)

  Wait for VBlank.
- void vid_set_mode (int dm, int pm)

  Set the video mode.
- void vid_set_mode_ex (vid_mode_t *mode)

  Set the video mode.
- void vid_init (int disp_mode, int pixel_mode)

  Initialize the video system.
- void vid_shutdown (void)

  Shut down the video system.
- int vid_screen_shot (const char *destfn)

  Take a screenshot.

Variables

- static const uint8 vid_pmode_bpp [4] = {2, 2, 3, 4}

  vid_pmode_bpp Video pixel mode depths
- vid_mode_t vid_builtin [DM_MODE_COUNT]

  The list of builtin video modes. Do not modify these!
- vid_mode_t * vid_mode

  The current video mode. Do not modify directly!
- uint16 * vram_s

  16-bit size pointer to the current drawing area.
- uint32 * vram_l

  32-bit size pointer to the current drawing area.

9.254.1 Detailed Description

Functions related to video output.

This file deals with the video output hardware in the Dreamcast. There are functions defined herein that deal with setting up the video hardware, defining the resolution of the display, dealing with the framebuffer, etc.

Author

Anders Clerwall

Megan Potter

9.254.2 Macro Definition Documentation

DM_MULTIBUFFER

```
#define DM_MULTIBUFFER 0x2000
```

Multi-buffered mode setting.

OR this with the generic mode to get four framebuffers instead of one.

VID_MAX_FB

```
#define VID_MAX_FB 4
```

The maximum number of framebuffers available.

9.254.3 Enumeration Type Documentation

anonymous enum

```
anonymous enum
```

Generic display modes.

Enumerator

DM_GENERIC_FIRST	First valid generic mode.
DM_320x240	320x240 resolution
DM_640x480	640x480 resolution
DM_800x608	800x608 resolution
DM_256x256	256x256 resolution
DM_768x480	768x480 resolution
DM_768x576	768x576 resolution
DM_GENERIC_LAST	Last valid generic mode.

anonymous enum

```
anonymous enum
```

Specific display modes.

Enumerator

DM_INVALID	Invalid display mode.

Enumerator

DM_320x240_VGA	320x240 VGA 60Hz
DM_320x240_NTSC	320x240 NTSC 60Hz
DM_640x480_VGA	640x480 VGA 60Hz
DM_640x480_NTSC_IL	640x480 NTSC Interlaced 60Hz
DM_800x608_VGA	800x608 VGA 60Hz
DM_640x480_PAL_IL	640x480 PAL Interlaced 50Hz
DM_256x256_PAL_IL	256x256 PAL Interlaced 50Hz
DM_768x480_NTSC_IL	768x480 NTSC Interlaced 60Hz
DM_768x576_PAL_IL	768x576 PAL Interlaced 50Hz
DM_768x480_PAL_IL	768x480 PAL Interlaced 50Hz
DM_320x240_PAL	320x240 PAL 50Hz
DM_320x240_VGA_MB	320x240 VGA 60Hz, 4FBs
DM_320x240_NTSC_MB	320x240 NTSC 60Hz, 4FBs
DM_640x480_VGA_MB	640x480 VGA 60Hz, 4FBs
DM_640x480_NTSC_IL_MB	640x480 NTSC IL 60Hz, 4FBs
DM_800x608_VGA_MB	800x608 VGA 60Hz, 4FBs
DM_640x480_PAL_IL_MB	640x480 PAL IL 50Hz, 4FBs
DM_256x256_PAL_IL_MB	256x256 PAL IL 50Hz, 4FBs
DM_768x480_NTSC_IL_MB	768x480 NTSC IL 60Hz, 4FBs
DM_768x576_PAL_IL_MB	768x576 PAL IL 50Hz, 4FBs
DM_768x480_PAL_IL_MB	768x480 PAL IL 50Hz, 4FBs
DM_320x240_PAL_MB	320x240 PAL 50Hz, 4FBs
DM_SENTINEL	Sentinel value, for counting.
DM_MODE_COUNT	Number of modes.

9.254.4 Function Documentation

vid_border_color()

```
uint32 vid_border_color (
            int r,
            int g,
            int b )
```

Set the border color of the display.

This sets the color of the border area of the display. On some screens, the border area may not be shown at all, whereas on some displays you may see the whole thing.

Parameters

r	The red value of the color (0-255).
g	The green value of the color (0-255).
b	The blue value of the color (0-255).

Returns

Old border color value (RGB888)

vid_check_cable()

```
int vid_check_cable (
            void  )
```

Retrieve the connected video cable type.

This function checks the video cable and reports what it finds.

Return values

CT_VGA	If a VGA Box or cable is connected.
CT_NONE	If nothing is connected.
CT_RGB	If a RGB/SCART cable is connected.
CT_COMPOSITE	If a composite cable or RF switch is connected.

vid_clear()

```
void vid_clear (
            int r,
            int g,
            int b )
```

Clear the display.

This function sets the whole display to the specified color. Internally, this uses the store queues to actually clear the display entirely.

Parameters

r	The red value of the color (0-255).
g	The green value of the color (0-255).
b	The blue value of the color (0-255).

vid_empty()

```
void vid_empty (
            void  )
```

Clear VRAM.

This function is essentially a memset() for the whole of VRAM that will clear it all to 0 bytes.

vid_flip()

```
void vid_flip (
            int fb )
```

Set the current framebuffer in a multibuffered setup.

This function sets the displayed framebuffer to the specified buffer and sets the vram_s and vram_l pointers to point at the next framebuffer, to allow for tearing-free framebuffer-direct drawing. The specified buffer is masked against (vid_mode->fb_count - 1) in order to loop around.

Parameters

fb	The framebuffer to display (or -1 for the next one).

vid_init()

```
void vid_init (
            int disp_mode,
            int pixel_mode )
```

Initialize the video system.

This function initializes the video display, setting the mode to the specified parameters, clearing vram, and setting the first framebuffer as active.

Parameters

disp_mode	The display mode to use. One of the DM_* values.
pixel_mode	The pixel mode to use. One of the PM_* values.

vid_screen_shot()

```
int vid_screen_shot (
            const char * destfn )
```

Take a screenshot.

This function takes the current framebuffer (vram_s/vram_l) and dumps it out to a PPM file.

Parameters

destfn	The filename to save to.

Returns

0 on success, <0 on failure.

vid_set_mode()

```
void vid_set_mode (
             int dm,
             int pm )
```

Set the video mode.

This function sets the current video mode to the one specified by the parameters.

Parameters

dm	The display mode to use. One of the DM_* values.
pm	The pixel mode to use. One of the PM_* values.

vid_set_mode_ex()

```
void vid_set_mode_ex (
             vid_mode_t * mode )
```

Set the video mode.

This function sets the current video mode to the mode structure passed in. You can use this to add support to your program for modes that KOS doesn't have support for built-in (of course, you should tell us the settings so we can add them into KOS if you do this).

Parameters

mode	A filled in vid_mode_t for the mode wanted.

vid_set_start()

```
void vid_set_start (
             uint32 base )
```

Set the VRAM base of the framebuffer.

This function sets the vram_s and vram_l pointsers to specified offset within VRAM and sets the start position of the framebuffer to the same offset.

Parameters

base	The offset within VRAM to set the base to.

vid_shutdown()

```
void vid_shutdown (
            void  )
```

Shut down the video system.

This function reinitializes the video system to what dcload and friends expect it to be.

vid_waitvbl()

```
void vid_waitvbl (
            void  )
```

Wait for VBlank.

This function busy loops until the vertical blanking period starts.

9.254.5 Variable Documentation

vid_builtin

```
vid_mode_t vid_builtin[DM_MODE_COUNT]  [extern]
```

The list of builtin video modes. Do not modify these!

vid_mode

```
vid_mode_t* vid_mode  [extern]
```

The current video mode. Do not modify directly!

vid_pmode_bpp

```
const uint8 vid_pmode_bpp[4] = {2, 2, 3, 4}  [static]
```

vid_pmode_bpp Video pixel mode depths

vram_l

```
uint32* vram_l  [extern]
```

32-bit size pointer to the current drawing area.

vram_s

```
uint16* vram_s  [extern]
```

16-bit size pointer to the current drawing area.

9.255 video.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003     dc/video.h
00004     Copyright (C) 2001 Anders Clerwall (scav)
00005
00006 */
00007
00008 /** \file   dc/video.h
00009     \brief  Functions related to video output.
00010
00011     This file deals with the video output hardware in the Dreamcast. There are
00012     functions defined herein that deal with setting up the video hardware,
00013     defining the resolution of the display, dealing with the framebuffer, etc.
00014
00015     \author Anders Clerwall
00016     \author Megan Potter
00017 */
00018
00019 #ifndef __DC_VIDEO_H
00020 #define __DC_VIDEO_H
00021
00022 #include <sys/cdefs.h>
00023 __BEGIN_DECLS
00024
00025 #include <arch/types.h>
00026
00027 /** \defgroup vid_ctype Video Cable types
00028
00029     The vid_check_cable() function will return one of this set of values to let
00030     you know what type of cable is connected to the Dreamcast. These are also
00031     used in the video mode settings to limit modes to certain cable types.
00032
00033     @{
00034 */
00035 #define CT_ANY          -1  /**< \brief Any cable type. Used only internally. */
00036 #define CT_VGA          0   /**< \brief VGA Box */
00037 #define CT_NONE         1   /**< \brief Nothing connected */
00038 #define CT_RGB          2   /**< \brief RGB/SCART cable */
00039 #define CT_COMPOSITE    3   /**< \brief Composite cable or RF switch */
00040 /** @} */
00041
00042 /** \defgroup vid_pmode Video pixel modes
00043
00044     This set of constants control the pixel mode that the framebuffer is set to.
00045
00046     @{
00047 */
00048 #define PM_RGB555   0       /**< \brief RGB555 pixel mode (15-bit) */
00049 #define PM_RGB565   1       /**< \brief RGB565 pixel mode (16-bit) */
00050 #define PM_RGB888P  2       /**< \brief RBG888 packed pixel mode (24-bit) */
00051 #define PM_RGB0888  3       /**< \brief RGB0888 pixel mode (32-bit) */
00052 #define PM_RGB888   PM_RGB0888 /**< \brief Backwards compatibility support */
00053 /** @} */
```

```
00054
00055 /** \brief vid_pmode_bpp Video pixel mode depths */
00056 static const uint8 vid_pmode_bpp[4] = {2, 2, 3, 4};
00057
00058 /** \brief  Generic display modes */
00059 enum {
00060     DM_GENERIC_FIRST = 0x1000,      /**< \brief First valid generic mode */
00061     DM_320x240 = 0x1000,            /**< \brief 320x240 resolution */
00062     DM_640x480,                     /**< \brief 640x480 resolution */
00063     DM_800x608,                     /**< \brief 800x608 resolution */
00064     DM_256x256,                     /**< \brief 256x256 resolution */
00065     DM_768x480,                     /**< \brief 768x480 resolution */
00066     DM_768x576,                     /**< \brief 768x576 resolution */
00067     DM_GENERIC_LAST = DM_768x576    /**< \brief Last valid generic mode */
00068 };
00069
00070 /** \brief  Multi-buffered mode setting.
00071
00072     OR this with the generic mode to get four framebuffers instead of one.
00073 */
00074 #define DM_MULTIBUFFER  0x2000
00075
00076 //------------------------------------------------------------------------------
00077 // More specific modes (and actual indeces into the mode table)
00078
00079 /** \brief  Specific display modes */
00080 enum {
00081     DM_INVALID = 0,                 /**< \brief Invalid display mode */
00082     // Valid modes below
00083     DM_320x240_VGA = 1,             /**< \brief 320x240 VGA 60Hz */
00084     DM_320x240_NTSC,                /**< \brief 320x240 NTSC 60Hz */
00085     DM_640x480_VGA,                 /**< \brief 640x480 VGA 60Hz */
00086     DM_640x480_NTSC_IL,             /**< \brief 640x480 NTSC Interlaced 60Hz */
00087     DM_800x608_VGA,                 /**< \brief 800x608 VGA 60Hz */
00088     DM_640x480_PAL_IL,              /**< \brief 640x480 PAL Interlaced 50Hz */
00089     DM_256x256_PAL_IL,              /**< \brief 256x256 PAL Interlaced 50Hz */
00090     DM_768x480_NTSC_IL,             /**< \brief 768x480 NTSC Interlaced 60Hz */
00091     DM_768x576_PAL_IL,              /**< \brief 768x576 PAL Interlaced 50Hz */
00092     DM_768x480_PAL_IL,              /**< \brief 768x480 PAL Interlaced 50Hz */
00093     DM_320x240_PAL,                 /**< \brief 320x240 PAL 50Hz */
00094     DM_320x240_VGA_MB,              /**< \brief 320x240 VGA 60Hz, 4FBs */
00095     DM_320x240_NTSC_MB,             /**< \brief 320x240 NTSC 60Hz, 4FBs */
00096     DM_640x480_VGA_MB,              /**< \brief 640x480 VGA 60Hz, 4FBs */
00097     DM_640x480_NTSC_IL_MB,          /**< \brief 640x480 NTSC IL 60Hz, 4FBs */
00098     DM_800x608_VGA_MB,              /**< \brief 800x608 VGA 60Hz, 4FBs */
00099     DM_640x480_PAL_IL_MB,           /**< \brief 640x480 PAL IL 50Hz, 4FBs */
00100     DM_256x256_PAL_IL_MB,           /**< \brief 256x256 PAL IL 50Hz, 4FBs */
00101     DM_768x480_NTSC_IL_MB,          /**< \brief 768x480 NTSC IL 60Hz, 4FBs */
00102     DM_768x576_PAL_IL_MB,           /**< \brief 768x576 PAL IL 50Hz, 4FBs */
00103     DM_768x480_PAL_IL_MB,           /**< \brief 768x480 PAL IL 50Hz, 4FBs */
00104     DM_320x240_PAL_MB,              /**< \brief 320x240 PAL 50Hz, 4FBs */
00105     // The below is only for counting..
00106     DM_SENTINEL,                    /**< \brief Sentinel value, for counting */
00107     DM_MODE_COUNT                   /**< \brief Number of modes */
00108 };
00109
00110 /** \brief  The maximum number of framebuffers available. */
00111 #define VID_MAX_FB  4   // <-- This should be enough
00112
00113 // These are for the "flags" field of "vid_mode_t"
00114 /** \defgroup vid_flags Flags for the field in vid_mode_t.
00115
00116     These flags indicate various things related to the modes for a vid_mode_t.
00117
00118     @{
00119 */
00120 #define VID_INTERLACE   0x00000001  /**< \brief Interlaced display */
00121 #define VID_LINEDOUBLE  0x00000002  /**< \brief Display each scanline twice */
00122 #define VID_PIXELDOUBLE 0x00000004  /**< \brief Display each pixel twice */
00123 #define VID_PAL         0x00000008  /**< \brief 50Hz refresh rate, if not VGA */
00124 /** @} */
00125
00126 /** \brief  Video mode structure.
00127
00128     KOS maintains a list of valid video modes internally that correspond to the
00129     specific display modes enumeration. Each of them is built of one of these.
00130
00131     \headerfile dc/video.h
00132 */
00133 typedef struct vid_mode {
00134     int     generic;    /**< \brief Generic mode type for vid_set_mode() */
```

```
00135     uint16  width;      /**< \brief Width of the display, in pixels */
00136     uint16  height;     /**< \brief Height of the display, in pixels */
00137     uint32  flags;      /**< \brief Combination of one or more VID_* flags */
00138
00139     int16   cable_type; /**< \brief Allowed cable type */
00140     uint16  pm;         /**< \brief Pixel mode */
00141
00142     uint16  scanlines;  /**< \brief Number of scanlines */
00143     uint16  clocks;     /**< \brief Clocks per scanline */
00144     uint16  bitmapx;    /**< \brief Bitmap window X position */
00145     uint16  bitmapy;    /**< \brief Bitmap window Y position (automatically
00146                              increased for PAL) */
00147     uint16  scanint1;   /**< \brief First scanline interrupt position */
00148     uint16  scanint2;   /**< \brief Second scanline interrupt position
00149                              (automatically doubled for VGA) */
00150     uint16  borderx1;   /**< \brief Border X starting position */
00151     uint16  borderx2;   /**< \brief Border X stop position */
00152     uint16  bordery1;   /**< \brief Border Y starting position */
00153     uint16  bordery2;   /**< \brief Border Y stop position */
00154
00155     uint16  fb_curr;    /**< \brief Current framebuffer */
00156     uint16  fb_count;   /**< \brief Number of framebuffers */
00157     uint32  fb_base[VID_MAX_FB];    /**< \brief Offset to framebuffers */
00158 } vid_mode_t;
00159
00160 /** \brief  The list of builtin video modes. Do not modify these! */
00161 extern vid_mode_t vid_builtin[DM_MODE_COUNT];
00162
00163 /** \brief  The current video mode. Do not modify directly! */
00164 extern vid_mode_t *vid_mode;
00165
00166 // These point to the current drawing area. If you're not using a multi-buffered
00167 // mode, that means they do what KOS always used to do (they'll point at the
00168 // start of VRAM). If you're using a multi-buffered mode, they'll point at the
00169 // next framebuffer to be displayed. You must use vid_flip for this to work
00170 // though (if you use vid_set_start, they'll point at the display base, for
00171 // compatibility's sake).
00172
00173 /** \brief  16-bit size pointer to the current drawing area. */
00174 extern uint16 *vram_s;
00175
00176 /** \brief  32-bit size pointer to the current drawing area. */
00177 extern uint32 *vram_l;
00178
00179
00180 /** \brief  Retrieve the connected video cable type.
00181
00182     This function checks the video cable and reports what it finds.
00183
00184     \retval CT_VGA          If a VGA Box or cable is connected.
00185     \retval CT_NONE         If nothing is connected.
00186     \retval CT_RGB          If a RGB/SCART cable is connected.
00187     \retval CT_COMPOSITE    If a composite cable or RF switch is connected.
00188 */
00189 int vid_check_cable(void);
00190
00191 /** \brief  Set the VRAM base of the framebuffer.
00192
00193     This function sets the vram_s and vram_l pointsers to specified offset
00194     within VRAM and sets the start position of the framebuffer to the same
00195     offset.
00196
00197     \param  base            The offset within VRAM to set the base to.
00198 */
00199 void vid_set_start(uint32 base);
00200
00201 /** \brief  Set the current framebuffer in a multibuffered setup.
00202
00203     This function sets the displayed framebuffer to the specified buffer and
00204     sets the vram_s and vram_l pointers to point at the next framebuffer, to
00205     allow for tearing-free framebuffer-direct drawing. The specified buffer
00206     is masked against (vid_mode->fb_count - 1) in order to loop around.
00207
00208     \param  fb              The framebuffer to display (or -1 for the next one).
00209 */
00210 void vid_flip(int fb);
00211
00212 /** \brief  Set the border color of the display.
00213
00214     This sets the color of the border area of the display. On some screens, the
00215     border area may not be shown at all, whereas on some displays you may see
```

```
00216     the whole thing.
00217
00218     \param  r               The red value of the color (0-255).
00219     \param  g               The green value of the color (0-255).
00220     \param  b               The blue value of the color (0-255).
00221     \return                 Old border color value (RGB888)
00222 */
00223 uint32 vid_border_color(int r, int g, int b);
00224
00225 /** \brief  Clear the display.
00226
00227     This function sets the whole display to the specified color. Internally,
00228     this uses the store queues to actually clear the display entirely.
00229
00230     \param  r               The red value of the color (0-255).
00231     \param  g               The green value of the color (0-255).
00232     \param  b               The blue value of the color (0-255).
00233 */
00234 void vid_clear(int r, int g, int b);
00235
00236 /** \brief  Clear VRAM.
00237
00238     This function is essentially a memset() for the whole of VRAM that will
00239     clear it all to 0 bytes.
00240 */
00241 void vid_empty(void);
00242
00243 /** \brief  Wait for VBlank.
00244
00245     This function busy loops until the vertical blanking period starts.
00246 */
00247 void vid_waitvbl(void);
00248
00249 /** \brief  Set the video mode.
00250
00251     This function sets the current video mode to the one specified by the
00252     parameters.
00253
00254     \param  dm              The display mode to use. One of the DM_* values.
00255     \param  pm              The pixel mode to use. One of the PM_* values.
00256 */
00257 void vid_set_mode(int dm, int pm);
00258
00259 /** \brief  Set the video mode.
00260
00261     This function sets the current video mode to the mode structure passed in.
00262     You can use this to add support to your program for modes that KOS doesn't
00263     have support for built-in (of course, you should tell us the settings so we
00264     can add them into KOS if you do this).
00265
00266     \param  mode            A filled in vid_mode_t for the mode wanted.
00267 */
00268 void vid_set_mode_ex(vid_mode_t *mode);
00269
00270 /** \brief  Initialize the video system.
00271
00272     This function initializes the video display, setting the mode to the
00273     specified parameters, clearing vram, and setting the first framebuffer as
00274     active.
00275
00276     \param  disp_mode       The display mode to use. One of the DM_* values.
00277     \param  pixel_mode      The pixel mode to use. One of the PM_* values.
00278 */
00279 void vid_init(int disp_mode, int pixel_mode);
00280
00281 /** \brief  Shut down the video system.
00282
00283     This function reinitializes the video system to what dcload and friends
00284     expect it to be.
00285 */
00286 void vid_shutdown(void);
00287
00288 /** \brief  Take a screenshot.
00289
00290     This function takes the current framebuffer (vram_s/vram_l) and dumps it out
00291     to a PPM file.
00292
00293     \param  destfn          The filename to save to.
00294     \return                 0 on success, <0 on failure.
00295 */
00296 int vid_screen_shot(const char * destfn);
```

```
00297
00298 __END_DECLS
00299
00300 #endif  // __DC_VIDEO_H
00301
```

9.256 kernel/arch/dreamcast/include/dc/vmu_fb.h File Reference

VMU framebuffer.

```
#include <dc/maple.h>
#include <stdint.h>
```

Data Structures

- struct vmufb_t

  VMU framebuffer.
- struct vmufb_font_t

  VMU framebuffer font meta-data.

Functions

- void vmufb_paint_area (vmufb_t *fb, unsigned int x, unsigned int y, unsigned int w, unsigned int h, const char *data)

  Render into the VMU framebuffer.
- void vmufb_clear_area (vmufb_t *fb, unsigned int x, unsigned int y, unsigned int w, unsigned int h)

  Clear a specific area of the VMU framebuffer.
- void vmufb_clear (vmufb_t *fb)

  Clear the VMU framebuffer.
- void vmufb_present (const vmufb_t *fb, maple_device_t *dev)

  Present the VMU framebuffer to a VMU.
- void vmufb_print_string_into (vmufb_t *fb, const vmufb_font_t *font, unsigned int x, unsigned int y, unsigned int w, unsigned int h, unsigned int line_spacing, const char *str)

  Render a string into the VMU framebuffer.
- static __inline__ void vmufb_print_string (vmufb_t *fb, const vmufb_font_t *font, const char *str)

  Render a string into the VMU framebuffer.

9.256.1 Detailed Description

VMU framebuffer.

This file provides an API that can be used to compose a 48x32 image that can then be displayed on the VMUs connected to the system.

9.256.2 Function Documentation

vmufb_clear()

```
void vmufb_clear (
            vmufb_t * fb )
```

Clear the VMU framebuffer.

This function clears the whole VMU framebuffer.

Parameters

fb	A pointer to the vmufb_t to paint to.

vmufb_clear_area()

```
void vmufb_clear_area (
            vmufb_t * fb,
            unsigned int x,
            unsigned int y,
            unsigned int w,
            unsigned int h )
```

Clear a specific area of the VMU framebuffer.

This function clears the area of the VMU framebuffer designated by the x, y, w and h values.

Parameters

fb	A pointer to the vmufb_t to paint to.
x	The horizontal position of the top-left corner of the drawing area, in pixels
y	The vertical position of the top-left corner of the drawing area, in pixels
w	The width of the drawing area, in pixels
h	The height of the drawing area, in pixels

vmufb_paint_area()

```
void vmufb_paint_area (
            vmufb_t * fb,
            unsigned int x,
            unsigned int y,
            unsigned int w,
            unsigned int h,
            const char * data )
```

Render into the VMU framebuffer.

This function will paint the provided pixel data into the VMU framebuffer, into the rectangle provided by the x, y, w and h values.

Parameters

fb	A pointer to the vmufb_t to paint to.
x	The horizontal position of the top-left corner of the drawing area, in pixels
y	The vertical position of the top-left corner of the drawing area, in pixels
w	The width of the drawing area, in pixels
h	The height of the drawing area, in pixels
data	A pointer to the pixel data that will be painted into the drawing area.

vmufb_present()

```
void vmufb_present (
            const vmufb_t * fb,
            maple_device_t * dev )
```

Present the VMU framebuffer to a VMU.

This function presents the previously rendered VMU framebuffer to the VMU identified by the dev argument.

Parameters

fb	A pointer to the vmufb_t to paint to.
dev	The maple device of the VMU to present to

vmufb_print_string()

```
static __inline__ void vmufb_print_string (
            vmufb_t * fb,
            const vmufb_font_t * font,
            const char * str )  [static]
```

Render a string into the VMU framebuffer.

Simplified version of vmufb_print_string_into(). This is the same as calling vmufb_print_string_into with x=0, y=0, w=48, h=32, line_spacing=0.

Parameters

fb	A pointer to the vmufb_t to paint to.
font	A pointer to the vmufb_font_t that will be used for painting the text
str	The text to render

References vmufb_print_string_into().

vmufb_print_string_into()

```
void vmufb_print_string_into (
            vmufb_t * fb,
            const vmufb_font_t * font,
            unsigned int x,
            unsigned int y,
            unsigned int w,
            unsigned int h,
            unsigned int line_spacing,
            const char * str )
```

Render a string into the VMU framebuffer.

This function uses the provided font to render text into the VMU framebuffer.

Parameters

fb	A pointer to the vmufb_t to paint to.
font	A pointer to the vmufb_font_t that will be used for painting the text
x	The horizontal position of the top-left corner of the drawing area, in pixels
y	The vertical position of the top-left corner of the drawing area, in pixels
w	The width of the drawing area, in pixels
h	The height of the drawing area, in pixels
line_spacing	Specify the number of empty lines that should separate two lines of text
str	The text to render

Referenced by vmufb_print_string().

9.257 vmu_fb.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/vmu_fb.h
00004    Copyright (C) 2023 Paul Cercueil
00005
00006 */
00007
00008 /** \file   dc/vmu_fb.h
00009     \brief  VMU framebuffer.
00010
00011     This file provides an API that can be used to compose a 48x32 image that can
00012     then be displayed on the VMUs connected to the system.
00013 */
00014
00015 #include <dc/maple.h>
00016 #include <stdint.h>
00017
00018 /** \brief  VMU framebuffer.
00019
00020     This object contains a 48x32 monochrome framebuffer. It can be painted to,
00021     or displayed one the VMUs connected to the system, using the API below.
00022
00023     \headerfile dc/vmu_fb.h
00024  */
00025 typedef struct {
00026     uint32_t data[48];
00027 } vmufb_t;
00028
00029 /** \brief  VMU framebuffer font meta-data.
00030
00031     \headerfile dc/vmu_fb.h
00032  */
00033 typedef struct {
00034     unsigned int w;         /**< \brief Character width in pixels */
00035     unsigned int h;         /**< \brief Character height in pixels */
00036     unsigned int stride;    /**< \brief Size of one character in bytes */
00037     const char *fontdata;   /**< \brief Pointer to the font data */
00038 } vmufb_font_t;
00039
00040 /** \brief  Render into the VMU framebuffer
00041
00042     This function will paint the provided pixel data into the VMU framebuffer,
00043     into the rectangle provided by the x, y, w and h values.
00044
00045     \param  fb              A pointer to the vmufb_t to paint to.
00046     \param  x               The horizontal position of the top-left corner of
00047                             the drawing area, in pixels
00048     \param  y               The vertical position of the top-left corner of the
00049                             drawing area, in pixels
00050     \param  w               The width of the drawing area, in pixels
```

```
00051     \param  h               The height of the drawing area, in pixels
00052     \param  data            A pointer to the pixel data that will be painted
00053                             into the drawing area.
00054  */
00055 void vmufb_paint_area(vmufb_t *fb,
00056                       unsigned int x, unsigned int y,
00057                       unsigned int w, unsigned int h,
00058                       const char *data);
00059
00060 /** \brief  Clear a specific area of the VMU framebuffer
00061
00062     This function clears the area of the VMU framebuffer designated by the
00063     x, y, w and h values.
00064
00065     \param  fb              A pointer to the vmufb_t to paint to.
00066     \param  x               The horizontal position of the top-left corner of
00067                             the drawing area, in pixels
00068     \param  y               The vertical position of the top-left corner of the
00069                             drawing area, in pixels
00070     \param  w               The width of the drawing area, in pixels
00071     \param  h               The height of the drawing area, in pixels
00072  */
00073 void vmufb_clear_area(vmufb_t *fb,
00074                       unsigned int x, unsigned int y,
00075                       unsigned int w, unsigned int h);
00076
00077 /** \brief  Clear the VMU framebuffer
00078
00079     This function clears the whole VMU framebuffer.
00080
00081     \param  fb              A pointer to the vmufb_t to paint to.
00082  */
00083 void vmufb_clear(vmufb_t *fb);
00084
00085 /** \brief  Present the VMU framebuffer to a VMU
00086
00087     This function presents the previously rendered VMU framebuffer to the
00088     VMU identified by the dev argument.
00089
00090     \param  fb              A pointer to the vmufb_t to paint to.
00091     \param  dev             The maple device of the VMU to present to
00092  */
00093 void vmufb_present(const vmufb_t *fb, maple_device_t *dev);
00094
00095 /** \brief  Render a string into the VMU framebuffer
00096
00097     This function uses the provided font to render text into the VMU
00098     framebuffer.
00099
00100     \param  fb              A pointer to the vmufb_t to paint to.
00101     \param  font            A pointer to the vmufb_font_t that will be used for
00102                             painting the text
00103     \param  x               The horizontal position of the top-left corner of
00104                             the drawing area, in pixels
00105     \param  y               The vertical position of the top-left corner of the
00106                             drawing area, in pixels
00107     \param  w               The width of the drawing area, in pixels
00108     \param  h               The height of the drawing area, in pixels
00109     \param  line_spacing    Specify the number of empty lines that should
00110                             separate two lines of text
00111     \param  str             The text to render
00112  */
00113 void vmufb_print_string_into(vmufb_t *fb,
00114                              const vmufb_font_t *font,
00115                              unsigned int x, unsigned int y,
00116                              unsigned int w, unsigned int h,
00117                              unsigned int line_spacing,
00118                              const char *str);
00119
00120 /** \brief  Render a string into the VMU framebuffer
00121
00122     Simplified version of vmufb_print_string_into(). This is the same as calling
00123     vmufb_print_string_into with x=0, y=0, w=48, h=32, line_spacing=0.
00124
00125     \param  fb              A pointer to the vmufb_t to paint to.
00126     \param  font            A pointer to the vmufb_font_t that will be used for
00127                             painting the text
00128     \param  str             The text to render
00129  */
00130 static __inline__ void
00131 vmufb_print_string(vmufb_t *fb, const vmufb_font_t *font, const char *str) {
```

```
00132     vmufb_print_string_into(fb, font, 0, 0, 48, 32, 0, str);
00133 }
```

9.258 kernel/arch/dreamcast/include/dc/vmu_pkg.h File Reference

VMU Packaging functionality.

```
#include <sys/cdefs.h>
#include <arch/types.h>
```

Data Structures

- struct vmu_pkg_t
  VMU Package type.
- struct vmu_hdr_t
  Final VMU package type.

Macros

- #define VMUPKG_EC_NONE 0
  No eyecatch.
- #define VMUPKG_EC_16BIT 1
  16-bit ARGB4444
- #define VMUPKG_EC_256COL 2
  256-color palette
- #define VMUPKG_EC_16COL 3
  16-color palette

Functions

- int vmu_pkg_build (vmu_pkg_t *src, uint8 **dst, int *dst_size)
  Convert a vmu_pkg_t into an array of uint8s.
- int vmu_pkg_parse (uint8 *data, vmu_pkg_t *pkg)
  Parse an array of uint8s into a vmu_pkg_t.

9.258.1 Detailed Description

VMU Packaging functionality.

This file provides declarations for managing the headers that must be attached to VMU files for the BIOS to pay attention to them. This does not handle reading/writing files directly.

Author

Megan Potter

See also

dc/fs_vmu.h

9.258.2 Function Documentation

vmu_pkg_build()

```
int vmu_pkg_build (
            vmu_pkg_t * src,
            uint8 ** dst,
            int * dst_size )
```

Convert a vmu_pkg_t into an array of uint8s.

This function converts a vmu_pkg_t structure into an array of uint8's which may be written to a VMU file via fs_vmu, or whatever.

Parameters

src	The vmu_pkg_t to convert.
dst	The buffer (will be allocated for you).
dst_size	The size of the output.

Returns

0 on success, <0 on failure.

vmu_pkg_parse()

```
int vmu_pkg_parse (
            uint8 * data,
            vmu_pkg_t * pkg )
```

Parse an array of uint8s into a vmu_pkg_t.

This function does the opposite of vmu_pkg_build and is used to parse VMU files read in.

Parameters

data	The buffer to parse.
pkg	Where to store the vmu_pkg_t.

Return values

-1	On invalid CRC in the data.
0	On success.

9.259 vmu_pkg.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
00002
00003    dc/vmu_pkg.h
00004    Copyright (C) 2002 Megan Potter
00005
00006 */
00007
00008 /** \file   dc/vmu_pkg.h
00009     \brief  VMU Packaging functionality.
00010
00011     This file provides declarations for managing the headers that must be
00012     attached to VMU files for the BIOS to pay attention to them. This does not
00013     handle reading/writing files directly.
00014
00015     \author Megan Potter
00016     \see    dc/fs_vmu.h
00017 */
00018
00019 #ifndef __DC_VMU_PKG_H
00020 #define __DC_VMU_PKG_H
00021
00022 #include <sys/cdefs.h>
00023 __BEGIN_DECLS
00024
00025 #include <arch/types.h>
00026
00027 /** \brief  VMU Package type.
00028
00029     Anyone wanting to package a VMU file should create one of these somewhere;
00030     eventually it will be turned into a flat file that you can save using
00031     fs_vmu.
00032
00033     \headerfile dc/vmu_pkg.h
00034 */
00035 typedef struct vmu_pkg {
00036     char        desc_short[20];     /**< \brief Short file description */
00037     char        desc_long[36];      /**< \brief Long file description */
00038     char        app_id[20];         /**< \brief Application ID */
00039     int         icon_cnt;           /**< \brief Number of icons */
00040     int         icon_anim_speed;    /**< \brief Icon animation speed */
00041     int         eyecatch_type;      /**< \brief "Eyecatch" type */
00042     int         data_len;           /**< \brief Number of data (payload) bytes */
00043     uint16      icon_pal[16];       /**< \brief Icon palette (ARGB4444) */
00044     const uint8 *icon_data;         /**< \brief 512*n bytes of icon data */
00045     const uint8 *eyecatch_data;     /**< \brief Eyecatch data */
00046     const uint8 *data;              /**< \brief Payload data */
00047 } vmu_pkg_t;
00048
00049 /** \brief  Final VMU package type.
00050
00051     This structure will be written into the file itself, not vmu_pkg_t.
00052
00053     \headerfile dc/vmu_pkg.h
00054 */
00055 typedef struct vmu_hdr {
00056     char    desc_short[16];     /**< \brief Space-padded short description */
00057     char    desc_long[32];      /**< \brief Space-padded long description*/
00058     char    app_id[16];         /**< \brief Null-padded application ID */
00059     uint16  icon_cnt;           /**< \brief Number of icons */
00060     uint16  icon_anim_speed;    /**< \brief Icon animation speed */
00061     uint16  eyecatch_type;      /**< \brief Eyecatch type */
00062     uint16  crc;                /**< \brief CRC of the file */
00063     uint32  data_len;           /**< \brief Payload size */
00064     uint8   reserved[20];       /**< \brief Reserved (all zero) */
00065     uint16  icon_pal[16];       /**< \brief Icon palette (ARGB4444) */
00066     /* 512*n Icon Bitmaps */
00067     /* Eyecatch palette + bitmap */
00068 } vmu_hdr_t;
00069
00070 /** \defgroup vmu_ectype    Eyecatch types.
00071
00072     All eyecatches are 72x56, but the pixel format is variable. Note that in all
00073     of the cases which use a palette, the palette entries are in ARGB4444 format
00074     and come directly before the pixel data itself.
00075
00076     @{
```

```
00077 */
00078 #define VMUPKG_EC_NONE      0   /**< \brief No eyecatch */
00079 #define VMUPKG_EC_16BIT     1   /**< \brief 16-bit ARGB4444 */
00080 #define VMUPKG_EC_256COL    2   /**< \brief 256-color palette */
00081 #define VMUPKG_EC_16COL     3   /**< \brief 16-color palette */
00082 /** @} */
00083
00084 /** \brief  Convert a vmu_pkg_t into an array of uint8s.
00085
00086     This function converts a vmu_pkg_t structure into an array of uint8's which
00087     may be written to a VMU file via fs_vmu, or whatever.
00088
00089     \param  src             The vmu_pkg_t to convert.
00090     \param  dst             The buffer (will be allocated for you).
00091     \param  dst_size        The size of the output.
00092     \return                 0 on success, <0 on failure.
00093 */
00094 int vmu_pkg_build(vmu_pkg_t *src, uint8 ** dst, int * dst_size);
00095
00096 /** \brief  Parse an array of uint8s into a vmu_pkg_t.
00097
00098     This function does the opposite of vmu_pkg_build and is used to parse VMU
00099     files read in.
00100
00101     \param  data            The buffer to parse.
00102     \param  pkg             Where to store the vmu_pkg_t.
00103     \retval -1              On invalid CRC in the data.
00104     \retval 0               On success.
00105 */
00106 int vmu_pkg_parse(uint8 *data, vmu_pkg_t *pkg);
00107
00108
00109 __END_DECLS
00110
00111 #endif  /* __DC_VMU_PKG_H */
00112
```

9.260 kernel/arch/dreamcast/include/dc/vmufs.h File Reference

Low-level VMU filesystem driver.

```
#include <sys/cdefs.h>
#include <dc/maple.h>
```

Data Structures

- struct vmu_timestamp_t

  BCD timestamp, used several places in the vmufs.
- struct vmu_root_t

  VMU FS Root block layout.
- struct vmu_dir_t

  VMU FS Directory entries, 32 bytes each.

Macros

- #define VMUFS_OVERWRITE 1

  Overwrite existing files.
- #define VMUFS_VMUGAME 2

  This file is a VMU game.
- #define VMUFS_NOCOPY 4

  Set the no-copy flag.

Functions

- void vmufs_dir_fill_time (vmu_dir_t *d)

  Fill in the date on a vmu_dir_t for writing.
- int vmufs_root_read (maple_device_t *dev, vmu_root_t *root_buf)

  Reads a selected VMU's root block.
- int vmufs_root_write (maple_device_t *dev, vmu_root_t *root_buf)

  Writes a selected VMU's root block.
- int vmufs_dir_blocks (vmu_root_t *root_buf)

  Given a VMU's root block, return the amount of space in bytes required to hold its directory.
- int vmufs_fat_blocks (vmu_root_t *root_buf)

  Given a VMU's root block, return the amount of space in bytes required to hold its FAT.
- int vmufs_dir_read (maple_device_t *dev, vmu_root_t *root_buf, vmu_dir_t *dir_buf)

  Given a selected VMU's root block, read its directory.
- int vmufs_dir_write (maple_device_t *dev, vmu_root_t *root, vmu_dir_t *dir_buf)

  Given a selected VMU's root block and dir blocks, write the dirty dir blocks back to the VMU. Assumes the mutex is held.
- int vmufs_fat_read (maple_device_t *dev, vmu_root_t *root, uint16 *fat_buf)

  Given a selected VMU's root block, read its FAT.
- int vmufs_fat_write (maple_device_t *dev, vmu_root_t *root, uint16 *fat_buf)

  Given a selected VMU's root block and its FAT, write the FAT blocks back to the VMU.
- int vmufs_dir_find (vmu_root_t *root, vmu_dir_t *dir, const char *fn)

  Given a previously-read directory, locate a file by filename.
- int vmufs_dir_add (vmu_root_t *root, vmu_dir_t *dir, vmu_dir_t *newdirent)

  Given a previously-read directory, add a new dirent to the dir.
- int vmufs_file_read (maple_device_t *dev, uint16 *fat, vmu_dir_t *dirent, void *outbuf)

  Given a pointer to a directory struct and a previously loaded FAT, load the indicated file from the VMU.
- int vmufs_file_write (maple_device_t *dev, vmu_root_t *root, uint16 *fat, vmu_dir_t *dir, vmu_dir_t *newdirent, void *filebuf, int size)

  Given a pointer to a mostly-filled directory struct and a previously loaded directory and FAT, write the indicated file to the VMU.
- int vmufs_file_delete (vmu_root_t *root, uint16 *fat, vmu_dir_t *dir, const char *fn)

  Given a previously-read FAT and directory, delete the named file.
- int vmufs_fat_free (vmu_root_t *root, uint16 *fat)

  Given a previously-read FAT, return the number of blocks available to write out new file data.
- int vmufs_dir_free (vmu_root_t *root, vmu_dir_t *dir)

  Given a previously-read directory, return the number of dirents available for new files.
- int vmufs_mutex_lock (void)

  Lock the vmufs mutex.
- int vmufs_mutex_unlock (void)

  Unlock the vmufs mutex.
- int vmufs_readdir (maple_device_t *dev, vmu_dir_t **outbuf, int *outcnt)

  Read the directory from a VMU.
- int vmufs_read (maple_device_t *dev, const char *fn, void **outbuf, int *outsize)

  Read a file from the VMU.
- int vmufs_read_dirent (maple_device_t *dev, vmu_dir_t *dirent, void **outbuf, int *outsize)

  Read a file from the VMU, using a pre-read dirent.
- int vmufs_write (maple_device_t *dev, const char *fn, void *inbuf, int insize, int flags)

  Write a file to the VMU.

- int vmufs_delete (maple_device_t *dev, const char *fn)

  Delete a file from the VMU.
- int vmufs_free_blocks (maple_device_t *dev)

  Return the number of user blocks free for file writing.
- int vmufs_init (void)

  Initialize vmufs.
- int vmufs_shutdown (void)

  Shutdown vmufs.

9.260.1 Detailed Description

Low-level VMU filesystem driver.

The VMU filesystem driver mounts itself on /vmu of the VFS. Each memory card has its own subdirectory off of that directory (i.e, /vmu/a1 for slot 1 of the first controller). VMUs themselves have no subdirectories, so the driver itself is fairly simple.

Files on a VMU must be multiples of 512 bytes in size, and should have a header attached so that they show up in the BIOS menu.

Author

Megan Potter

See also

dc/vmu_pkg.h

dc/fs_vmu.h

9.260.2 Macro Definition Documentation

VMUFS_NOCOPY

```
#define VMUFS_NOCOPY 4
```

Set the no-copy flag.

VMUFS_OVERWRITE

```
#define VMUFS_OVERWRITE 1
```

Overwrite existing files.

VMUFS_VMUGAME

```
#define VMUFS_VMUGAME 2
```

This file is a VMU game.

9.260.3 Function Documentation

vmufs_delete()

```
int vmufs_delete (
            maple_device_t * dev,
            const char * fn )
```

Delete a file from the VMU.

Return values

0	On success.
-1	If the file is not found.
-2	On other failure.

vmufs_dir_add()

```
int vmufs_dir_add (
            vmu_root_t * root,
            vmu_dir_t * dir,
            vmu_dir_t * newdirent )
```

Given a previously-read directory, add a new dirent to the dir.

Another file with the same name should not exist (delete it first if it does). This function will not check for dupes!

Parameters

root	The VMU root block.
dir	The VMU directory.
newdirent	The new entry to add.

Returns

0 on success, or $<$0 on failure.

vmufs_dir_blocks()

```
int vmufs_dir_blocks (
            vmu_root_t * root_buf )
```

Given a VMU's root block, return the amount of space in bytes required to hold its directory.

Parameters

root_buf	The root block to check.

Returns

The amount of space, in bytes, needed.

vmufs_dir_fill_time()

```
void vmufs_dir_fill_time (
            vmu_dir_t * d )
```

Fill in the date on a vmu_dir_t for writing.

Parameters

d	The directory to fill in the date on.

vmufs_dir_find()

```
int vmufs_dir_find (
            vmu_root_t * root,
            vmu_dir_t * dir,
            const char * fn )
```

Given a previously-read directory, locate a file by filename.

Parameters

root	The VMU root block.
dir	The VMU directory.
fn	The file to find (only checked up to 12 chars).

Returns

The index into the directory array on success, or <0 on failure.

vmufs_dir_free()

```
int vmufs_dir_free (
            vmu_root_t * root,
            vmu_dir_t * dir )
```

Given a previously-read directory, return the number of dirents available for new files.

Parameters

root	The VMU root block.
dir	The directory in question.

Returns

The number of entries available.

vmufs_dir_read()

```
int vmufs_dir_read (
            maple_device_t * dev,
            vmu_root_t * root_buf,
            vmu_dir_t * dir_buf )
```

Given a selected VMU's root block, read its directory.

This function reads the directory of a given VMU root block. It assumes the mutex is held. There must be at least the number of bytes returned by vmufs_dir_blocks() available in the buffer for this to succeed.

Parameters

dev	The VMU to read.
root_buf	The VMU's root block.
dir_buf	The buffer to hold the directory. You must have allocated this yourself.

Returns

0 on success, $<$0 on failure.

vmufs_dir_write()

```
int vmufs_dir_write (
            maple_device_t * dev,
            vmu_root_t * root,
            vmu_dir_t * dir_buf )
```

Given a selected VMU's root block and dir blocks, write the dirty dir blocks back to the VMU. Assumes the mutex is held.

Parameters

dev	The VMU to write to.
root	The VMU's root block.
dir_buf	The VMU's directory structure.

Returns

0 on success, <0 on failure.

vmufs_fat_blocks()

```
int vmufs_fat_blocks (
            vmu_root_t * root_buf )
```

Given a VMU's root block, return the amount of space in bytes required to hold its FAT.

Parameters

root_buf	The root block to check.

Returns

The amount of space, in bytes, needed.

vmufs_fat_free()

```
int vmufs_fat_free (
            vmu_root_t * root,
            uint16 * fat )
```

Given a previously-read FAT, return the number of blocks available to write out new file data.

Parameters

root	The VMU root block.
fat	The FAT to be examined.

Returns

The number of blocks available.

vmufs_fat_read()

```
int vmufs_fat_read (
            maple_device_t * dev,
            vmu_root_t * root,
            uint16 * fat_buf )
```

Given a selected VMU's root block, read its FAT.

This function reads the FAT of a VMU, given its root block. It assumes the mutex is held. There must be at least the number of bytes returned by vmufs_fat_blocks() available in the buffer for this to succeed.

Parameters

dev	The VMU to read from.
root	The VMU's root block.
fat_buf	The buffer to store the FAT into. You must pre-allocate this.

Returns

0 on success, <0 on failure.

vmufs_fat_write()

```
int vmufs_fat_write (
            maple_device_t * dev,
            vmu_root_t * root,
            uint16 * fat_buf )
```

Given a selected VMU's root block and its FAT, write the FAT blocks back to the VMU.

This function assumes the mutex is held.

Parameters

dev	The VMU to write to.
root	The VMU's root block.
fat_buf	The buffer to write to the FAT.

Returns

0 on success, <0 on failure.

vmufs_file_delete()

```
int vmufs_file_delete (
            vmu_root_t * root,
```

```
            uint16 * fat,
            vmu_dir_t * dir,
            const char * fn )
```

Given a previously-read FAT and directory, delete the named file.

No changes are made to the VMU itself, just the in-memory structs.

Parameters

root	The VMU root block.
fat	The FAT to be modified.
dir	The directory to be modified.
fn	The file name to be deleted.

Return values

0	On success.
-1	If fn is not found.

vmufs_file_read()

```
int vmufs_file_read (
            maple_device_t * dev,
            uint16 * fat,
            vmu_dir_t * dirent,
            void * outbuf )
```

Given a pointer to a directory struct and a previously loaded FAT, load the indicated file from the VMU.

An appropriate amount of space must have been allocated previously in the buffer. Assumes the mutex is held.

Parameters

dev	The VMU to read from.
fat	The FAT of the VMU.
dirent	The entry to read.
outbuf	A buffer to write the data into. You must allocate this yourself with the appropriate amount of space.

Returns

0 on success, $<$0 on failure.

vmufs_file_write()

```
int vmufs_file_write (
            maple_device_t * dev,
```

```
            vmu_root_t * root,
            uint16 * fat,
            vmu_dir_t * dir,
            vmu_dir_t * newdirent,
            void * filebuf,
            int size )
```

Given a pointer to a mostly-filled directory struct and a previously loaded directory and FAT, write the indicated file to the VMU.

The named file should not exist in the directory already. The directory and FAT will *not* be sync'd back to the VMU, this must be done manually. Assumes the mutex is held.

Parameters

dev	The VMU to write to.
root	The VMU root block.
fat	The FAT of the VMU.
dir	The directory of the VMU.
newdirent	The new entry to write.
filebuf	The new file data.
size	The size of the file in blocks (512-bytes each).

Returns

0 on success, <0 on failure.

vmufs_free_blocks()

```
int vmufs_free_blocks (
            maple_device_t * dev )
```

Return the number of user blocks free for file writing.

You should check this number before attempting to write.

Returns

The number of blocks free for writing.

vmufs_init()

```
int vmufs_init (
            void  )
```

Initialize vmufs.

Must be called before anything else is useful.

Return values

0	On success (no error conditions defined).

vmufs_mutex_lock()

```
int vmufs_mutex_lock (
            void  )
```

Lock the vmufs mutex.

This should be done before you attempt any low-level ops.

Return values

0	On success (no error conditions defined).

vmufs_mutex_unlock()

```
int vmufs_mutex_unlock (
            void  )
```

Unlock the vmufs mutex.

This should be done once you're done with any low-level ops.

Return values

0	On success (no error conditions defined).

vmufs_read()

```
int vmufs_read (
            maple_device_t * dev,
            const char * fn,
            void ** outbuf,
            int * outsize )
```

Read a file from the VMU.

The output buffer will be allocated for you using malloc(), and the size of the file will be returned. On failure, outbuf will not contain a dangling buffer that needs to be freed (no further action required).

Parameters

dev	The VMU to read from.
fn	The name of the file to read.
outbuf	A buffer that will be allocated where the file data will be placed.
outsize	Storage for the size of the file, in bytes.

Returns

0 on success, or <0 on failure.

vmufs_read_dirent()

```
int vmufs_read_dirent (
            maple_device_t * dev,
            vmu_dir_t * dirent,
            void ** outbuf,
            int * outsize )
```

Read a file from the VMU, using a pre-read dirent.

This function is faster to use than vmufs_read() if you already have done the lookup, since it won't need to do that.

Parameters

dev	The VMU to read from.
dirent	The entry to read.
outbuf	A buffer that will be allocated where the file data will be placed.
outsize	Storage for the size of the file, in bytes.

Returns

0 on success, <0 on failure.

vmufs_readdir()

```
int vmufs_readdir (
            maple_device_t * dev,
            vmu_dir_t ** outbuf,
            int * outcnt )
```

Read the directory from a VMU.

The output buffer will be allocated for you using malloc(), and the number of entries will be returned. On failure, outbuf will not contain a dangling buffer that needs to be freed (no further action required).

Parameters

dev	The VMU to read from.
outbuf	A buffer that will be allocated where the directory data will be placed.
outcnt	The number of entries in outbuf.

Returns

0 on success, or <0 on failure.

vmufs_root_read()

```
int vmufs_root_read (
            maple_device_t * dev,
            vmu_root_t * root_buf )
```

Reads a selected VMU's root block.

This function assumes the mutex is held.

Parameters

dev	The VMU to read from.
root_buf	A buffer to hold the root block. You must allocate this yourself before calling.

Return values

-1	On failure.
0	On success.

vmufs_root_write()

```
int vmufs_root_write (
            maple_device_t * dev,
            vmu_root_t * root_buf )
```

Writes a selected VMU's root block.

This function assumes the mutex is held.

Parameters

dev	The VMU to write to.
root_buf	The root block to write.

Return values

-1	On failure.
0	On success.

vmufs_shutdown()

```
int vmufs_shutdown (
            void  )
```

Shutdown vmufs.

Must be called after everything is finished.

vmufs_write()

```
int vmufs_write (
            maple_device_t * dev,
            const char * fn,
            void * inbuf,
            int insize,
            int flags )
```

Write a file to the VMU.

If the named file already exists, then the function checks 'flags'. If VMUFS_OVERWRITE is set, then the old file is deleted first before the new one is written (this all happens atomically). On partial failure, some data blocks may have been written, but in general the card should not be damaged.

Parameters

dev	The VMU to write to.
fn	The filename to write.
inbuf	The data to write to the file.
insize	The size of the file in bytes.
flags	Flags for the write (i.e, VMUFS_OVERWRITE, VMUFS_VMUGAME, VMUFS_NOCOPY).

Returns

0 on success, or $<$0 for failure.

9.261 vmufs.h

Go to the documentation of this file.

```
00001 /* KallistiOS ##version##
```

```
00002
00003    dc/vmufs.h
00004    Copyright (C) 2003 Megan Potter
00005
00006 */
00007
00008 /** \file   dc/vmufs.h
00009     \brief  Low-level VMU filesystem driver.
00010
00011     The VMU filesystem driver mounts itself on /vmu of the VFS. Each memory card
00012     has its own subdirectory off of that directory (i.e, /vmu/a1 for slot 1 of
00013     the first controller). VMUs themselves have no subdirectories, so the driver
00014     itself is fairly simple.
00015
00016     Files on a VMU must be multiples of 512 bytes in size, and should have a
00017     header attached so that they show up in the BIOS menu.
00018
00019     \author Megan Potter
00020     \see    dc/vmu_pkg.h
00021     \see    dc/fs_vmu.h
00022 */
00023
00024 #ifndef __DC_VMUFS_H
00025 #define __DC_VMUFS_H
00026
00027 #include <sys/cdefs.h>
00028 __BEGIN_DECLS
00029
00030 #include <dc/maple.h>
00031
00032 /* \cond */
00033 #define __packed__ __attribute__((packed))
00034 /* \endcond */
00035
00036 /** \brief  BCD timestamp, used several places in the vmufs.
00037     \headerfile dc/vmufs.h
00038 */
00039 typedef struct {
00040     uint8   cent;   /**< \brief Century */
00041     uint8   year;   /**< \brief Year, within century */
00042     uint8   month;  /**< \brief Month of the year */
00043     uint8   day;    /**< \brief Day of the month */
00044     uint8   hour;   /**< \brief Hour of the day */
00045     uint8   min;    /**< \brief Minutes */
00046     uint8   sec;    /**< \brief Seconds */
00047     uint8   dow;    /**< \brief Day of week (0 = monday, etc) */
00048 } __packed__ vmu_timestamp_t;
00049
00050 /** \brief  VMU FS Root block layout.
00051     \headerfile dc/vmufs.h
00052 */
00053 typedef struct {
00054     uint8           magic[16];      /**< \brief All should contain 0x55 */
00055     uint8           use_custom;     /**< \brief 0 = standard, 1 = custom */
00056     uint8           custom_color[4];/**< \brief blue, green, red, alpha */
00057     uint8           pad1[27];       /**< \brief All zeros */
00058     vmu_timestamp_t timestamp;      /**< \brief BCD timestamp */
00059     uint8           pad2[8];        /**< \brief All zeros */
00060     uint8           unk1[6];        /**< \brief ??? */
00061     uint16          fat_loc;        /**< \brief FAT location */
00062     uint16          fat_size;       /**< \brief FAT size in blocks */
00063     uint16          dir_loc;        /**< \brief Directory location */
00064     uint16          dir_size;       /**< \brief Directory size in blocks */
00065     uint16          icon_shape;     /**< \brief Icon shape for this VMS */
00066     uint16          blk_cnt;        /**< \brief Number of user blocks */
00067     uint8           unk2[430];      /**< \brief ??? */
00068 } __packed__ vmu_root_t;
00069
00070 /** \brief  VMU FS Directory entries, 32 bytes each.
00071     \headerfile dc/vmufs.h
00072 */
00073 typedef struct {
00074     uint8           filetype;       /**< \brief 0x00 = no file; 0x33 = data; 0xcc = a game */
00075     uint8           copyprotect;    /**< \brief 0x00 = copyable; 0xff = copy protected */
00076     uint16          firstblk;       /**< \brief Location of the first block in the file */
00077     char            filename[12];   /**< \brief Note: there is no null terminator */
00078     vmu_timestamp_t timestamp;      /**< \brief File time */
00079     uint16          filesize;       /**< \brief Size of the file in blocks */
00080     uint16          hdroff;         /**< \brief Offset of header, in blocks from start of file */
00081     uint8           dirty;          /**< \brief See header notes */
00082     uint8           pad1[3];        /**< \brief All zeros */
```

```
00083 } __packed__ vmu_dir_t;
00084 #undef __packed__
00085
00086 /* Notes about the "dirty" field on vmu_dir_t :)
00087
00088    This byte should always be zero when written out to the VMU. What this
00089    lets us do, though, is conserve on flash writes. If you only want to
00090    modify one single file (which is the standard case) then re-writing all
00091    of the dir blocks is a big waste. Instead, you should set the dirty flag
00092    on the in-mem copy of the directory, and writing it back out will only
00093    flush the containing block back to the VMU, setting it back to zero
00094    in the process. Loaded blocks should always have zero here (though we
00095    enforce that in the code to make sure) so it will be non-dirty by
00096    default.
00097  */
00098
00099
00100 /* ****************** Low level functions ******************** */
00101
00102 /** \brief  Fill in the date on a vmu_dir_t for writing.
00103
00104     \param  d               The directory to fill in the date on.
00105 */
00106 void vmufs_dir_fill_time(vmu_dir_t *d);
00107
00108 /** \brief  Reads a selected VMU's root block.
00109
00110     This function assumes the mutex is held.
00111
00112     \param  dev             The VMU to read from.
00113     \param  root_buf        A buffer to hold the root block. You must allocate
00114                             this yourself before calling.
00115     \retval -1              On failure.
00116     \retval 0               On success.
00117 */
00118 int vmufs_root_read(maple_device_t * dev, vmu_root_t * root_buf);
00119
00120 /** \brief  Writes a selected VMU's root block.
00121
00122     This function assumes the mutex is held.
00123
00124     \param  dev             The VMU to write to.
00125     \param  root_buf        The root block to write.
00126     \retval -1              On failure.
00127     \retval 0               On success.
00128 */
00129 int vmufs_root_write(maple_device_t * dev, vmu_root_t * root_buf);
00130
00131 /** \brief  Given a VMU's root block, return the amount of space in bytes
00132             required to hold its directory.
00133
00134     \param  root_buf        The root block to check.
00135     \return                 The amount of space, in bytes, needed.
00136 */
00137 int vmufs_dir_blocks(vmu_root_t * root_buf);
00138
00139 /** \brief  Given a VMU's root block, return the amount of space in bytes
00140             required to hold its FAT.
00141
00142     \param  root_buf        The root block to check.
00143     \return                 The amount of space, in bytes, needed.
00144 */
00145 int vmufs_fat_blocks(vmu_root_t * root_buf);
00146
00147 /** \brief  Given a selected VMU's root block, read its directory.
00148
00149     This function reads the directory of a given VMU root block. It assumes the
00150     mutex is held. There must be at least the number of bytes returned by
00151     vmufs_dir_blocks() available in the buffer for this to succeed.
00152
00153     \param  dev             The VMU to read.
00154     \param  root_buf        The VMU's root block.
00155     \param  dir_buf         The buffer to hold the directory. You must have
00156                             allocated this yourself.
00157     \return                 0 on success, <0 on failure.
00158 */
00159 int vmufs_dir_read(maple_device_t * dev, vmu_root_t * root_buf,
00160                    vmu_dir_t * dir_buf);
00161
00162 /** \brief  Given a selected VMU's root block and dir blocks, write the dirty
00163             dir blocks back to the VMU. Assumes the mutex is held.
```

```
00164
00165     \param  dev             The VMU to write to.
00166     \param  root            The VMU's root block.
00167     \param  dir_buf         The VMU's directory structure.
00168     \return                 0 on success, <0 on failure.
00169 */
00170 int vmufs_dir_write(maple_device_t * dev, vmu_root_t * root,
00171                     vmu_dir_t * dir_buf);
00172
00173 /** \brief Given a selected VMU's root block, read its FAT.
00174
00175     This function reads the FAT of a VMU, given its root block. It assumes the
00176     mutex is held. There must be at least the number of bytes returned by
00177     vmufs_fat_blocks() available in the buffer for this to succeed.
00178
00179     \param  dev             The VMU to read from.
00180     \param  root            The VMU's root block.
00181     \param  fat_buf         The buffer to store the FAT into. You must
00182                             pre-allocate this.
00183     \return                 0 on success, <0 on failure.
00184 */
00185 int vmufs_fat_read(maple_device_t * dev, vmu_root_t * root, uint16 * fat_buf);
00186
00187 /** \brief  Given a selected VMU's root block and its FAT, write the FAT blocks
00188             back to the VMU.
00189
00190     This function assumes the mutex is held.
00191
00192     \param  dev             The VMU to write to.
00193     \param  root            The VMU's root block.
00194     \param  fat_buf         The buffer to write to the FAT.
00195     \return                 0 on success, <0 on failure.
00196 */
00197 int vmufs_fat_write(maple_device_t * dev, vmu_root_t * root, uint16 * fat_buf);
00198
00199 /** \brief  Given a previously-read directory, locate a file by filename.
00200
00201     \param  root            The VMU root block.
00202     \param  dir             The VMU directory.
00203     \param  fn              The file to find (only checked up to 12 chars).
00204     \return                 The index into the directory array on success, or
00205                             <0 on failure.
00206 */
00207 int vmufs_dir_find(vmu_root_t * root, vmu_dir_t * dir, const char * fn);
00208
00209 /** \brief  Given a previously-read directory, add a new dirent to the dir.
00210
00211     Another file with the same name should not exist (delete it first if it
00212     does). This function will not check for dupes!
00213
00214     \param  root            The VMU root block.
00215     \param  dir             The VMU directory.
00216     \param  newdirent       The new entry to add.
00217     \return                 0 on success, or <0 on failure. */
00218 int vmufs_dir_add(vmu_root_t * root, vmu_dir_t * dir, vmu_dir_t * newdirent);
00219
00220 /** \brief  Given a pointer to a directory struct and a previously loaded FAT,
00221             load the indicated file from the VMU.
00222
00223     An appropriate amount of space must have been allocated previously in the
00224     buffer. Assumes the mutex is held.
00225
00226     \param  dev             The VMU to read from.
00227     \param  fat             The FAT of the VMU.
00228     \param  dirent          The entry to read.
00229     \param  outbuf          A buffer to write the data into. You must allocate
00230                             this yourself with the appropriate amount of space.
00231     \return                 0 on success, <0 on failure.
00232 */
00233 int vmufs_file_read(maple_device_t * dev, uint16 * fat, vmu_dir_t * dirent, void * outbuf);
00234
00235 /** \brief  Given a pointer to a mostly-filled directory struct and a previously
00236             loaded directory and FAT, write the indicated file to the VMU.
00237
00238     The named file should not exist in the directory already. The directory and
00239     FAT will _not_ be sync'd back to the VMU, this must be done manually.
00240     Assumes the mutex is held.
00241
00242     \param  dev             The VMU to write to.
00243     \param  root            The VMU root block.
00244     \param  fat             The FAT of the VMU.
```

```
00245     \param  dir             The directory of the VMU.
00246     \param  newdirent       The new entry to write.
00247     \param  filebuf         The new file data.
00248     \param  size            The size of the file in blocks (512-bytes each).
00249     \return                 0 on success, <0 on failure.
00250 */
00251 int vmufs_file_write(maple_device_t * dev, vmu_root_t * root, uint16 * fat,
00252                      vmu_dir_t * dir, vmu_dir_t * newdirent, void * filebuf, int size);
00253
00254 /** \brief  Given a previously-read FAT and directory, delete the named file.
00255
00256     No changes are made to the VMU itself, just the in-memory structs.
00257
00258     \param  root            The VMU root block.
00259     \param  fat             The FAT to be modified.
00260     \param  dir             The directory to be modified.
00261     \param  fn              The file name to be deleted.
00262     \retval 0               On success.
00263     \retval -1              If fn is not found.
00264 */
00265 int vmufs_file_delete(vmu_root_t * root, uint16 * fat, vmu_dir_t * dir, const char *fn);
00266
00267 /** \brief  Given a previously-read FAT, return the number of blocks available
00268             to write out new file data.
00269
00270     \param  root            The VMU root block.
00271     \param  fat             The FAT to be examined.
00272     \return                 The number of blocks available.
00273 */
00274 int vmufs_fat_free(vmu_root_t * root, uint16 * fat);
00275
00276 /** \brief  Given a previously-read directory, return the number of dirents
00277             available for new files.
00278
00279     \param  root            The VMU root block.
00280     \param  dir             The directory in question.
00281     \return                 The number of entries available.
00282 */
00283 int vmufs_dir_free(vmu_root_t * root, vmu_dir_t * dir);
00284
00285 /** \brief  Lock the vmufs mutex.
00286
00287     This should be done before you attempt any low-level ops.
00288
00289     \retval 0               On success (no error conditions defined).
00290 */
00291 int vmufs_mutex_lock(void);
00292
00293 /** \brief  Unlock the vmufs mutex.
00294
00295     This should be done once you're done with any low-level ops.
00296
00297     \retval 0               On success (no error conditions defined).
00298 */
00299 int vmufs_mutex_unlock(void);
00300
00301
00302 /* ****************** Higher level functions ******************** */
00303
00304 /** \brief  Read the directory from a VMU.
00305
00306     The output buffer will be allocated for you using malloc(), and the number
00307     of entries will be returned. On failure, outbuf will not contain a dangling
00308     buffer that needs to be freed (no further action required).
00309
00310     \param  dev             The VMU to read from.
00311     \param  outbuf          A buffer that will be allocated where the directory
00312                             data will be placed.
00313     \param  outcnt          The number of entries in outbuf.
00314     \return                 0 on success, or <0 on failure. */
00315 int vmufs_readdir(maple_device_t * dev, vmu_dir_t ** outbuf, int * outcnt);
00316
00317 /** \brief  Read a file from the VMU.
00318
00319     The output buffer will be allocated for you using malloc(), and the size of
00320     the file will be returned.  On failure, outbuf will not contain a dangling
00321     buffer that needs to be freed (no further action required).
00322
00323     \param  dev             The VMU to read from.
00324     \param  fn              The name of the file to read.
00325     \param  outbuf          A buffer that will be allocated where the file data
```

```
00326                           will be placed.
00327     \param  outsize       Storage for the size of the file, in bytes.
00328     \return               0 on success, or <0 on failure.
00329 */
00330 int vmufs_read(maple_device_t * dev, const char * fn, void ** outbuf, int * outsize);
00331
00332 /** \brief  Read a file from the VMU, using a pre-read dirent.
00333
00334     This function is faster to use than vmufs_read() if you already have done
00335     the lookup, since it won't need to do that.
00336
00337     \param  dev           The VMU to read from.
00338     \param  dirent        The entry to read.
00339     \param  outbuf        A buffer that will be allocated where the file data
00340                           will be placed.
00341     \param  outsize       Storage for the size of the file, in bytes.
00342     \return               0 on success, <0 on failure.
00343 */
00344 int vmufs_read_dirent(maple_device_t * dev, vmu_dir_t * dirent, void ** outbuf, int * outsize);
00345
00346 /* Flags for vmufs_write */
00347 #define VMUFS_OVERWRITE 1   /**< \brief Overwrite existing files */
00348 #define VMUFS_VMUGAME   2   /**< \brief This file is a VMU game */
00349 #define VMUFS_NOCOPY    4   /**< \brief Set the no-copy flag */
00350
00351 /** \brief Write a file to the VMU.
00352
00353     If the named file already exists, then the function checks 'flags'. If
00354     VMUFS_OVERWRITE is set, then the old file is deleted first before the new
00355     one is written (this all happens atomically). On partial failure, some data
00356     blocks may have been written, but in general the card should not be damaged.
00357
00358     \param  dev           The VMU to write to.
00359     \param  fn            The filename to write.
00360     \param  inbuf         The data to write to the file.
00361     \param  insize        The size of the file in bytes.
00362     \param  flags         Flags for the write (i.e, VMUFS_OVERWRITE,
00363                           VMUFS_VMUGAME, VMUFS_NOCOPY).
00364     \return               0 on success, or <0 for failure.
00365 */
00366 int vmufs_write(maple_device_t * dev, const char * fn, void * inbuf, int insize, int flags);
00367
00368 /** \brief  Delete a file from the VMU.
00369
00370     \retval 0             On success.
00371     \retval -1            If the file is not found.
00372     \retval -2            On other failure.
00373 */
00374 int vmufs_delete(maple_device_t * dev, const char * fn);
00375
00376 /** \brief  Return the number of user blocks free for file writing.
00377
00378     You should check this number before attempting to write.
00379
00380     \return               The number of blocks free for writing.
00381 */
00382 int vmufs_free_blocks(maple_device_t * dev);
00383
00384
00385 /** \brief  Initialize vmufs.
00386
00387     Must be called before anything else is useful.
00388
00389     \retval 0             On success (no error conditions defined).
00390 */
00391 int vmufs_init(void);
00392
00393 /** \brief  Shutdown vmufs.
00394
00395     Must be called after everything is finished.
00396 */
00397 int vmufs_shutdown(void);
00398
00399 __END_DECLS
00400
00401 #endif  /* __DC_VMUFS_H */
```

9.262 README.md File Reference

Index