SuperH Interfaces Guide

Paul Mundt

<lethal@linux-sh.org>

Table of Contents

Chapter 1. Memory Management

Table of Contents

SH-4

Store Queue API

Name

sq_flush_range — Flush (prefetch) a specific SQ range

Synopsis

```
void sq_flush_range (start,
                     len);

unsigned long  start;
unsigned int   len;
```

Arguments

start

the store queue address to start flushing from

len

the length to flush

Description

Flushes the store queue cache from `start` to `start` + `len` in a linear fashion.

Name

sq_remap — Map a physical address through the Store Queues

Synopsis

```
unsigned long sq_remap (phys,
                        size,
                        name,
                        flags);

unsigned long  phys;
unsigned int   size;
const char *   name;
unsigned long  flags;
```

Arguments

phys

Physical address of mapping.

size

Length of mapping.

name

User invoking mapping.

flags

Protection flags.

Description

Remaps the physical address *phys* through the next available store queue address of *size* length. *name* is logged at boot time as well as through the sysfs interface.

Name

sq_unmap — Unmap a Store Queue allocation

Synopsis

```
void sq_unmap (vaddr);

unsigned long  vaddr;
```

Arguments

vaddr

Pre-allocated Store Queue mapping.

Description

Unmaps the store queue allocation *map* that was previously created by `sq_remap`. Also frees up the pte that was previously inserted into the kernel page table and discards the UTLB translation.

SH-5

TLB Interfaces

Name

sh64_tlb_init — Perform initial setup for the DTLB and ITLB.

Synopsis

`int` **`sh64_tlb_init`** `(`*void*`);`

void;

Arguments

void

no arguments

Name

sh64_next_free_dtlb_entry — Find the next available DTLB entry

Synopsis

`unsigned long long` **`sh64_next_free_dtlb_entry`** `(`*void*`);`

void;

Arguments

void

no arguments

Name

sh64_get_wired_dtlb_entry — Allocate a wired (locked-in) entry in the DTLB

Synopsis

`unsigned long long` **`sh64_get_wired_dtlb_entry`** `(`*void*`);`

`void``;`

Arguments

`void`

no arguments

Name

sh64_put_wired_dtlb_entry — Free a wired (locked-in) entry in the DTLB.

Synopsis

`int` **`sh64_put_wired_dtlb_entry`** `(`*entry*`);`

`unsigned long long` *`entry`*`;`

Arguments

`entry`

Address of TLB slot.

Description

Works like a stack, last one to allocate must be first one to free.

Name

sh64_setup_tlb_slot — Load up a translation in a wired slot.

Synopsis

`void` **`sh64_setup_tlb_slot`** `(`*config_addr*,

```
                                         eaddr,
                                         asid,
                                         paddr);

unsigned long long  config_addr;
unsigned long       eaddr;
unsigned long       asid;
unsigned long       paddr;
```

Arguments

config_addr

Address of TLB slot.

eaddr

Virtual address.

asid

Address Space Identifier.

paddr

Physical address.

Description

Load up a virtual<->physical translation for *eaddr*<->*paddr* in the pre-allocated TLB slot *config_addr* (see sh64_get_wired_dtlb_entry).

Name

sh64_teardown_tlb_slot — Teardown a translation.

Synopsis

```
void sh64_teardown_tlb_slot (config_addr);

unsigned long long  config_addr;
```

Arguments

config_addr

Address of TLB slot.

Description

Teardown any existing mapping in the TLB slot *`config_addr`*.

Name

for_each_dtlb_entry — Iterate over free (non-wired) DTLB entries

Synopsis

`for_each_dtlb_entry` `(`*tlb*`);`

`tlb``;`

Arguments

`tlb`

TLB entry

Name

for_each_itlb_entry — Iterate over free (non-wired) ITLB entries

Synopsis

`for_each_itlb_entry` `(`*tlb*`);`

`tlb``;`

Arguments

`tlb`

TLB entry

Name

__flush_tlb_slot — Flushes TLB slot *`slot`*.

Synopsis

```
void __flush_tlb_slot (slot);

unsigned long long  slot;
```

Arguments

slot

Address of TLB slot.

Chapter 2. Clock Framework Extensions

Table of Contents

Name

clk_set_rate_ex — set the clock rate for a clock source, with additional parameter

Synopsis

```
int clk_set_rate_ex (clk,
                     rate,
                     algo_id);

struct clk *   clk;
unsigned long  rate;
int            algo_id;
```

Arguments

clk

clock source

rate

desired clock rate in Hz

algo_id

algorithm id to be passed down to ops->set_rate

Description

Returns success (0) or negative errno.

Chapter 3. Machine Specific Interfaces

Table of Contents

mach-dreamcast

Name

aica_rtc_gettimeofday — Get the time from the AICA RTC

Synopsis

```
void aica_rtc_gettimeofday (ts);

struct timespec *  ts;
```

Arguments

ts

pointer to resulting timespec

Description

Grabs the current RTC seconds counter and adjusts it to the Unix Epoch.

Name

aica_rtc_settimeofday — Set the AICA RTC to the current time

Synopsis

```
int aica_rtc_settimeofday (secs);

const time_t  secs;
```

Arguments

secs

contains the time_t to set

Description

Adjusts the given *tv* to the AICA Epoch and sets the RTC seconds counter.

mach-x3proto

Name

ilsel_enable — Enable an ILSEL set.

Synopsis

```
int ilsel_enable (set);

ilsel_source_t  set;
```

Arguments

set

ILSEL source (see ilsel_source_t enum in include/asm-sh/ilsel.h).

Description

Enables a given non-aliased ILSEL source (<= ILSEL_KEY) at the highest available interrupt level. Callers should take care to order callsites noting descending interrupt levels. Aliasing FPGA and external board IRQs need to use `ilsel_enable_fixed`.

The return value is an IRQ number that can later be taken down with `ilsel_disable`.

Name

ilsel_enable_fixed — Enable an ILSEL set at a fixed interrupt level

Synopsis

```
int ilsel_enable_fixed (set,
                        level);

ilsel_source_t  set;
unsigned int    level;
```

Arguments

`set`

ILSEL source (see ilsel_source_t enum in include/asm-sh/ilsel.h).

`level`

Interrupt level (1 - 15)

Description

Enables a given ILSEL source at a fixed interrupt level. Necessary both for level reservation as well as for aliased sources that only exist on special ILSEL#s.

Returns an IRQ number (as `ilsel_enable`).

Name

ilsel_disable — Disable an ILSEL set

Synopsis

```
void ilsel_disable (irq);

unsigned int  irq;
```

Arguments

`irq`

Bit position for ILSEL set value (retval from enable routines)

Description

Disable a previously enabled ILSEL set.

Chapter 4. Busses

Table of Contents

SuperHyway

Name

superhyway_add_device — Add a SuperHyway module

Synopsis

```
int superhyway_add_device (base,
                           sdev,
                           bus);

unsigned long               base;
struct superhyway_device *  sdev;
struct superhyway_bus *     bus;
```

Arguments

base

Physical address where module is mapped.

sdev

SuperHyway device to add, or NULL to allocate a new one.

bus

Bus where SuperHyway module resides.

Description

This is responsible for adding a new SuperHyway module. This sets up a new struct superhyway_device for the module being added if `sdev` == NULL.

Devices are initially added in the order that they are scanned (from the top-down of the memory map), and are assigned an ID based on the order that they are added. Any manual addition of a module will thus get the ID after the devices already discovered regardless of where it resides in memory.

Further work can and should be done in `superhyway_scan_bus`, to be sure that any new modules are properly discovered and subsequently registered.

Name

superhyway_register_driver — Register a new SuperHyway driver

Synopsis

```
int superhyway_register_driver (drv);

struct superhyway_driver *  drv;
```

Arguments

`drv`

SuperHyway driver to register.

Description

This registers the passed in *`drv`*. Any devices matching the id table will automatically be populated and handed off to the driver's specified probe routine.

Name

superhyway_unregister_driver — Unregister a SuperHyway driver

Synopsis

```
void superhyway_unregister_driver (drv);

struct superhyway_driver *  drv;
```

Arguments

`drv`

SuperHyway driver to unregister.

Description

This cleans up after `superhyway_register_driver`, and should be invoked in the exit path of any module drivers.

Maple

Name

maple_driver_register — register a maple driver

Synopsis

```
int maple_driver_register (drv);

struct maple_driver *  drv;
```

Arguments

`drv`

maple driver to be registered.

Description

Registers the passed in `drv`, while updating the bus type. Devices with matching function IDs will be automatically probed.

Name

maple_driver_unregister — unregister a maple driver.

Synopsis

```
void maple_driver_unregister (drv);

struct maple_driver *  drv;
```

Arguments

`drv`

maple driver to unregister.

Description

Cleans up after `maple_driver_register`. To be invoked in the exit path of any module drivers.

Name

maple_getcond_callback — setup handling MAPLE_COMMAND_GETCOND

Synopsis

```
void maple_getcond_callback (dev,
                             callback,
                             interval,
                             function);

struct maple_device *  dev;
void (*                callback(struct mapleq *mq);
unsigned long          interval;
unsigned long          function;
```

Arguments

dev

device responding

callback

handler callback

interval

interval in jiffies between callbacks

function

the function code for the device

Name

maple_add_packet — add a single instruction to the maple bus queue

Synopsis

```
int maple_add_packet (mdev,
                      function,
                      command,
                      length,
                      data);

struct maple_device *  mdev;
u32                    function;
u32                    command;
size_t                 length;
void *                 data;
```

Arguments

mdev

maple device

function

function on device being queried

`command`

maple command to add

`length`

length of command string (in 32 bit words)

`data`

remainder of command string