Linux Sh Documentation

The kernel development community

Jan 15, 2023

CONTENTS

Author Paul Mundt

CHAPTER

ONE

DEVICETREE BOOTING

Device-tree compatible SH bootloaders are expected to provide the physical address of the device tree blob in r4. Since legacy bootloaders did not guarantee any particular initial register state, kernels built to inter-operate with old bootloaders must either use a builtin DTB or select a legacy board option (something other than CONFIG_SH_DEVICE_TREE) that does not use device tree. Support for the latter is being phased out in favor of device tree.

CHAPTER

TWO

ADDING A NEW BOARD TO LINUXSH

Paul Mundt <lethal@linux-sh.org>

This document attempts to outline what steps are necessary to add support for new boards to the LinuxSH port under the new 2.5 and 2.6 kernels. This also attempts to outline some of the noticeable changes between the 2.4 and the 2.5/2.6 SH backend.

2.1 1. New Directory Structure

The first thing to note is the new directory structure. Under 2.4, most of the board-specific code (with the exception of stboards) ended up in arch/sh/kernel/ directly, with board-specific headers ending up in include/asm-sh/. For the new kernel, things are broken out by board type, companion chip type, and CPU type. Looking at a tree view of this directory hierarchy looks like the following:

Board-specific code:

```
.
|-- arch
|   `-- sh
|       `-- boards
|           |-- adx
|           |   `-- board-specific files
|           |-- bigsur
|           |   `-- board-specific files
|           |
|           ... more boards here ...
|
`-- include
    `-- asm-sh
        |-- adx
        |   `-- board-specific headers
        |-- bigsur
        |   `-- board-specific headers
        |
        .. more boards here ...
```

Next, for companion chips:

```
.
`-- arch
    `-- sh
        `-- cchips
            `-- hd6446x
                `-- hd64461
                    `-- cchip-specific files
```

... and so on. Headers for the companion chips are treated the same way as board-specific headers. Thus, include/asm-sh/hd64461 is home to all of the hd64461-specific headers.

Finally, CPU family support is also abstracted:

```
.
|-- arch
|   `-- sh
|       |-- kernel
|       |   `-- cpu
|       |       |-- sh2
|       |       |   `-- SH-2 generic files
|       |       |-- sh3
|       |       |   `-- SH-3 generic files
|       |       `-- sh4
|       |           `-- SH-4 generic files
|       `-- mm
|           `-- This is also broken out per CPU family, so each family can
|               have their own set of cache/tlb functions.
|
`-- include
    `-- asm-sh
        |-- cpu-sh2
        |   `-- SH-2 specific headers
        |-- cpu-sh3
        |   `-- SH-3 specific headers
        `-- cpu-sh4
            `-- SH-4 specific headers
```

It should be noted that CPU subtypes are _not_ abstracted. Thus, these still need to be dealt with by the CPU family specific code.

2.2 2. Adding a New Board

The first thing to determine is whether the board you are adding will be isolated, or whether it will be part of a family of boards that can mostly share the same board-specific code with minor differences.

In the first case, this is just a matter of making a directory for your board in arch/sh/boards/ and adding rules to hook your board in with the build system (more on this in the next section). However, for board families it makes more sense to have a common top-level arch/sh/boards/ directory and then populate that with sub-directories for each member of the family. Both the Solution Engine and the hp6xx boards are an example of this.

After you have setup your new arch/sh/boards/ directory, remember that you should also add a directory in include/asm-sh for headers localized to this board (if there are going to be more than one). In order to interoperate seamlessly with the build system, it's best to have this directory the same as the arch/sh/boards/ directory name, though if your board is again part of a family, the build system has ways of dealing with this (via incdir-y overloading), and you can feel free to name the directory after the family member itself.

There are a few things that each board is required to have, both in the arch/sh/boards and the include/asm-sh/ hierarchy. In order to better explain this, we use some examples for adding an imaginary board. For setup code, we're required at the very least to provide definitions for get_system_type() and platform_setup(). For our imaginary board, this might look something like:

```
/*
* arch/sh/boards/vapor/setup.c - Setup code for imaginary board
*/
#include <linux/init.h>

const char *get_system_type(void)
{
        return "FooTech Vaporboard";
}

int __init platform_setup(void)
{
        /*
        * If our hardware actually existed, we would do real
        * setup here. Though it's also sane to leave this empty
        * if there's no real init work that has to be done for
        * this board.
        */

        /* Start-up imaginary PCI ... */

        /* And whatever else ... */

        return 0;
}
```

Our new imaginary board will also have to tie into the machvec in order for it to be of any use.

machvec functions fall into a number of categories:

- I/O functions to IO memory (inb etc) and PCI/main memory (readb etc).
- I/O mapping functions (ioport_map, ioport_unmap, etc).
- a 'heartbeat' function.
- PCI and IRQ initialization routines.
- Consistent allocators (for boards that need special allocators, particularly for allocating out of some board-specific SRAM for DMA handles).

There are machvec functions added and removed over time, so always be sure to consult include/asm-sh/machvec.h for the current state of the machvec.

The kernel will automatically wrap in generic routines for undefined function pointers in the machvec at boot time, as machvec functions are referenced unconditionally throughout most of the tree. Some boards have incredibly sparse machvecs (such as the dreamcast and sh03), whereas others must define virtually everything (rts7751r2d).

Adding a new machine is relatively trivial (using vapor as an example):

If the board-specific definitions are quite minimalistic, as is the case for the vast majority of boards, simply having a single board-specific header is sufficient.

- add a new file include/asm-sh/vapor.h which contains prototypes for any machine specific IO functions prefixed with the machine name, for example vapor_inb. These will be needed when filling out the machine vector.

  Note that these prototypes are generated automatically by setting __IO_PREFIX to something sensible. A typical example would be:

  ```
  #define __IO_PREFIX vapor
  #include <asm/io_generic.h>
  ```

  somewhere in the board-specific header. Any boards being ported that still have a legacy io.h should remove it entirely and switch to the new model.

- Add machine vector definitions to the board's setup.c. At a bare minimum, this must be defined as something like:

  ```
  struct sh_machine_vector mv_vapor __initmv = {
          .mv_name = "vapor",
  };
  ALIAS_MV(vapor)
  ```

- finally add a file arch/sh/boards/vapor/io.c, which contains definitions of the machine specific io functions (if there are enough to warrant it).

2.3 3. Hooking into the Build System

Now that we have the corresponding directories setup, and all of the board-specific code is in place, it's time to look at how to get the whole mess to fit into the build system.

Large portions of the build system are now entirely dynamic, and merely require the proper entry here and there in order to get things done.

The first thing to do is to add an entry to arch/sh/Kconfig, under the "System type" menu:

```
config SH_VAPOR
        bool "Vapor"
        help
        select Vapor if configuring for a FooTech Vaporboard.
```

next, this has to be added into arch/sh/Makefile. All boards require a machdir-y entry in order to be built. This entry needs to be the name of the board directory as it appears in arch/sh/boards, even if it is in a sub-directory (in which case, all parent directories below arch/sh/boards/ need to be listed). For our new board, this entry can look like:

```
machdir-$(CONFIG_SH_VAPOR)  += vapor
```

provided that we've placed everything in the arch/sh/boards/vapor/ directory.

Next, the build system assumes that your include/asm-sh directory will also be named the same. If this is not the case (as is the case with multiple boards belonging to a common family), then the directory name needs to be implicitly appended to incdir-y. The existing code manages this for the Solution Engine and hp6xx boards, so see these for an example.

Once that is taken care of, it's time to add an entry for the mach type. This is done by adding an entry to the end of the arch/sh/tools/mach-types list. The method for doing this is self explanatory, and so we won't waste space restating it here. After this is done, you will be able to use implicit checks for your board if you need this somewhere throughout the common code, such as:

```
/* Make sure we're on the FooTech Vaporboard */
if (!mach_is_vapor())
        return -ENODEV;
```

also note that the mach_is_boardname() check will be implicitly forced to lowercase, regardless of the fact that the mach-types entries are all uppercase. You can read the script if you really care, but it's pretty ugly, so you probably don't want to do that.

Now all that's left to do is providing a defconfig for your new board. This way, other people who end up with this board can simply use this config for reference instead of trying to guess what settings are supposed to be used on it.

Also, as soon as you have copied over a sample .config for your new board (assume arch/sh/configs/vapor_defconfig), you can also use this directly as a build target, and it will be implicitly listed as such in the help text.

Looking at the 'make help' output, you should now see something like:

Architecture specific targets (sh):

zImage	Compressed kernel image (arch/sh/boot/zImage)
adx_defconfig	Build for adx
cqreek_defconfig	Build for cqreek
dreamcast_defconfig	Build for dreamcast
…	
vapor_defconfig	Build for vapor

which then allows you to do:

```
$ make ARCH=sh CROSS_COMPILE=sh4-linux- vapor_defconfig vmlinux
```

which will in turn copy the defconfig for this board, run it through oldconfig (prompting you for any new options since the time of creation), and start you on your way to having a functional kernel for your new board.

CHAPTER

THREE

NOTES ON REGISTER BANK USAGE IN THE KERNEL

3.1 Introduction

The SH-3 and SH-4 CPU families traditionally include a single partial register bank (selected by SR.RB, only r0 ... r7 are banked), whereas other families may have more full-featured banking or simply no such capabilities at all.

3.2 SR.RB banking

In the case of this type of banking, banked registers are mapped directly to r0 ... r7 if SR.RB is set to the bank we are interested in, otherwise ldc/stc can still be used to reference the banked registers (as r0_bank ... r7_bank) when in the context of another bank. The developer must keep the SR.RB value in mind when writing code that utilizes these banked registers, for obvious reasons. Userspace is also not able to poke at the bank1 values, so these can be used rather effectively as scratch registers by the kernel.

Presently the kernel uses several of these registers.

- r0_bank, r1_bank (referenced as k0 and k1, used for scratch registers when doing exception handling).
- r2_bank (used to track the EXPEVT/INTEVT code)
  - Used by do_IRQ() and friends for doing irq mapping based off of the interrupt exception vector jump table offset
- r6_bank (global interrupt mask)
  - The SR.IMASK interrupt handler makes use of this to set the interrupt priority level (used by local_irq_enable())
- r7_bank (current)

CHAPTER

FOUR

FEATURE STATUS ON SH ARCHITECTURE

Subsystem	Feature	Kconfig	Status	Desc
core	cBPF-JIT	HAVE_CBPF_JIT	TODO	arch
core	eBPF-JIT	HAVE_EBPF_JIT	TODO	arch
core	generic-idle-thread	GENERIC_SMP_IDLE_THREAD	ok	arch
core	jump-labels	HAVE_ARCH_JUMP_LABEL	TODO	arch
core	thread-info-in-task	THREAD_INFO_IN_TASK	TODO	arch
core	tracehook	HAVE_ARCH_TRACEHOOK	ok	arch
debug	debug-vm-pgtable	ARCH_HAS_DEBUG_VM_PGTABLE	TODO	arch
debug	gcov-profile-all	ARCH_HAS_GCOV_PROFILE_ALL	ok	arch
debug	KASAN	HAVE_ARCH_KASAN	TODO	arch
debug	kcov	ARCH_HAS_KCOV	TODO	arch
debug	kgdb	HAVE_ARCH_KGDB	ok	arch
debug	kmemleak	HAVE_DEBUG_KMEMLEAK	ok	arch
debug	kprobes	HAVE_KPROBES	ok	arch
debug	kprobes-on-ftrace	HAVE_KPROBES_ON_FTRACE	TODO	arch
debug	kretprobes	HAVE_KRETPROBES	ok	arch
debug	optprobes	HAVE_OPTPROBES	TODO	arch
debug	stackprotector	HAVE_STACKPROTECTOR	ok	arch
debug	uprobes	ARCH_SUPPORTS_UPROBES	TODO	arch
debug	user-ret-profiler	HAVE_USER_RETURN_NOTIFIER	TODO	arch
io	dma-contiguous	HAVE_DMA_CONTIGUOUS	TODO	arch
locking	cmpxchg-local	HAVE_CMPXCHG_LOCAL	TODO	arch
locking	lockdep	LOCKDEP_SUPPORT	ok	arch
locking	queued-rwlocks	ARCH_USE_QUEUED_RWLOCKS	TODO	arch
locking	queued-spinlocks	ARCH_USE_QUEUED_SPINLOCKS	TODO	arch
perf	kprobes-event	HAVE_REGS_AND_STACK_ACCESS_API	ok	arch
perf	perf-regs	HAVE_PERF_REGS	TODO	arch
perf	perf-stackdump	HAVE_PERF_USER_STACK_DUMP	TODO	arch
sched	membarrier-sync-core	ARCH_HAS_MEMBARRIER_SYNC_CORE	TODO	arch
sched	numa-balancing	ARCH_SUPPORTS_NUMA_BALANCING	—	arch
seccomp	seccomp-filter	HAVE_ARCH_SECCOMP_FILTER	ok	arch
time	arch-tick-broadcast	ARCH_HAS_TICK_BROADCAST	ok	arch
time	clockevents	!LEGACY_TIMER_TICK	ok	arch
time	context-tracking	HAVE_CONTEXT_TRACKING	TODO	arch
time	irq-time-acct	HAVE_IRQ_TIME_ACCOUNTING	TODO	arch
time	virt-cpuacct	HAVE_VIRT_CPU_ACCOUNTING	TODO	arch

Table 1 – continued from p

Subsystem	Feature	Kconfig	Status	Desc
vm	batch-unmap-tlb-flush	ARCH_WANT_BATCHED_UNMAP_TLB_FLUSH	TODO	arch
vm	ELF-ASLR	ARCH_HAS_ELF_RANDOMIZE	TODO	arch
vm	huge-vmap	HAVE_ARCH_HUGE_VMAP	TODO	arch
vm	ioremap_prot	HAVE_IOREMAP_PROT	ok	arch
vm	PG_uncached	ARCH_USES_PG_UNCACHED	TODO	arch
vm	pte_special	ARCH_HAS_PTE_SPECIAL	ok	arch
vm	THP	HAVE_ARCH_TRANSPARENT_HUGEPAGE	—	arch

CHAPTER

FIVE

MEMORY MANAGEMENT

5.1 SH-4

5.1.1 Store Queue API

void **sq_flush_range**(unsigned long start, unsigned int len)
Flush (prefetch) a specific SQ range

Parameters

unsigned long start the store queue address to start flushing from

unsigned int len the length to flush

Description

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

unsigned long **sq_remap**(unsigned long phys, unsigned int size, const char *name, pgprot_t prot)
Map a physical address through the Store Queues

Parameters

unsigned long phys Physical address of mapping.

unsigned int size Length of mapping.

const char *name User invoking mapping.

pgprot_t prot Protection bits.

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.

void **sq_unmap**(unsigned long vaddr)
Unmap a Store Queue allocation

Parameters

unsigned long 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.

CHAPTER

SIX

MACHINE SPECIFIC INTERFACES

6.1 mach-dreamcast

int **aica_rtc_gettimeofday**(struct device *dev, struct rtc_time *tm)
Get the time from the AICA RTC

Parameters

struct device *dev the RTC device (ignored)

struct rtc_time *tm pointer to resulting RTC time structure

Description

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

int **aica_rtc_settimeofday**(struct device *dev, struct rtc_time *tm)
Set the AICA RTC to the current time

Parameters

struct device *dev the RTC device (ignored)

struct rtc_time *tm pointer to new RTC time structure

Description

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

6.2 mach-x3proto

int **ilsel_enable**(ilsel_source_t set)
Enable an ILSEL set.

Parameters

ilsel_source_t 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()*.

int **ilsel_enable_fixed**(ilsel_source_t set, unsigned int level)
    Enable an ILSEL set at a fixed interrupt level

Parameters

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

unsigned int 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()*).

void **ilsel_disable**(unsigned int irq)
    Disable an ILSEL set

Parameters

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

Description

Disable a previously enabled ILSEL set.

CHAPTER

SEVEN

BUSSES

7.1 SuperHyway

int **superhyway_add_device**(unsigned long base, struct superhyway_device *sdev, struct superhyway_bus *bus)

Add a SuperHyway module

Parameters

unsigned long base Physical address where module is mapped.

struct superhyway_device *sdev SuperHyway device to add, or NULL to allocate a new one.

struct superhyway_bus *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.

int **superhyway_register_driver**(struct superhyway_driver *drv)

Register a new SuperHyway driver

Parameters

struct superhyway_driver *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.

void **superhyway_unregister_driver**(struct superhyway_driver *drv)

Unregister a SuperHyway driver

Parameters

struct superhyway_driver *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.

7.2 Maple

int **maple_driver_register**(struct maple_driver *drv)
register a maple driver

Parameters

struct maple_driver *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.

void **maple_driver_unregister**(struct maple_driver *drv)
unregister a maple driver.

Parameters

struct maple_driver *drv maple driver to unregister.

Description

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

void **maple_getcond_callback**(struct maple_device *dev, void (*callback)(struct mapleq *mq), unsigned long interval, unsigned long function)
setup handling MAPLE_COMMAND_GETCOND

Parameters

struct maple_device *dev device responding

void (*callback) (struct mapleq *mq) handler callback

unsigned long interval interval in jiffies between callbacks

unsigned long function the function code for the device

int **maple_add_packet**(struct maple_device *mdev, u32 function, u32 command, size_t length, void *data)
add a single instruction to the maple bus queue

Parameters

struct maple_device *mdev maple device

u32 function function on device being queried

u32 command maple command to add

size_t length length of command string (in 32 bit words)

void *data remainder of command string

INDEX