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.

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. 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>
#include <asm/rtc.h> /* for board_time_init() */

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.
         */

        /*
         * Presume all FooTech boards have the same broken timer,
         * and also presume that we've defined foo_timer_init to
         * do something useful.
         */
        board_time_init = foo_timer_init;

        /* 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).

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.