Wandboard porting guide / release notes

Tapani

May 19, 2014

1 Booting

1.1 Boot order

The Wandboard features two SD card slots, one on the CPU module and one on the baseboard. The default boot order is to boot from the CPU module first, and if no SD card is detected – try booting from the baseboard slot.

The boot order is determined by resistors on the module (and might be changeable with a soldering iron and a voided warranty).

1.2 Boot order issues

The iMX6 CPU support up to four SD card (MMC) slots; the Wandboard has two. The SD card slot on the CPU module is number three, and the slot on the baseboard is number one.

By default the linux kernel would initialize slot one first, and slot three afterwards – and then look for a root filesystem on SD1. There is a hack in the board file to initialize the SD-card slots in reverse order, but still, if the SD card in SD3 (module) is slow to be detected, the kernel can misdetect the order of the cards.

1.3 SD card layout

The iMX6 ROM will look for a bootloader at byte offset 1024 in the SD card. If using the recommended SPL based u-boot, the SPL binary should start there.

For instance a SPL binary can be copied to the SD card by:
`$ sudo dd if=SPL of=/dev/$dev bs=1k seek=1 oflag=dsync`
(where $dev is the SD card device).

The SPL looks for a u-boot 69kB into the SD card. A u-boot binary can be installed by
`$ sudo dd if=u-boot.img of=/dev/$dev bs=1k seek=69 oflag=dsync`

The default u-boot expects a kernel one megabyte into the card. Hence, the kernel can be installed to SD by:
`# dd if=arch/arm/boot/uImage of=/dev/$dev bs=1M seek=1`

Of course, a custom u-boot can change the location where it will attempt to load a kernel.

WARNING: Make sure you use the right device for your SD! Using the wrong device name can overwrite the content on your hard drive!

NOTE: When partitioning the SD card, reserve some space on the card for u-boot + kernel by leaving some "unused" space before the first partition. That is, start your first partition a few megabytes into the card. A large kernel can be ≈5 Mb in size, so leaving, say 8MB unpartitioned before the first partition should be safe.

2 Compiling

2.1 Setting up a cross-compiler

In order to compile the kernel or u-boot bootloader you need a *cross-compiler*, that is a compiler running on one machine but producing executables for another.

The recommended compiler for the Wandboard u-boot and kernel is some of the older CodeSourcery compilers. For instance version CodeSourcery G++ Lite 2010.09-50 has so far stood the test of time and provided stable kernel binaries. Unfortunately all cross compiler versions are not as stable.

To set up a cross compiler on a linux host, one way is to follow the steps:

1. Install a cross compiler somewhere, like /opt/arm-2010.09
2. Set the environment variable CROSS_COMPILE to the prefix of your cross compiler, i.e. `export CROSS_COMPILE=arm-none-linux-gnueabi-`.
3. Add the cross compilers bin folder to your PATH. i.e.
   `export PATH=$PATH:/opt/arm-2010.09/bin`
   The /opt/arm-2010.09/bin folder is an example location of `arm-none-linux-gnueabi-gcc`.

For software packages, often setting the environment variable ARCH to arm (by `export ARCH=arm`) is needed.

2.2 u-boot

The u-boot included in this release is a modified version of the mainline 2013.10 release. This uses SPL boot to avoid separate versions of u-boot for the different Wandboard models. The cost is having an additional boot-loader, SPL, that executes before u-boot. The CPU/memory detection is done in SPL which then proceeeds loading u-boot as usual.

To compile u-boot and SPL for the Wandboard, issue the following command in the u-boot folder:

```
% make -j4 wandboard
```

Now the generated binaries are named SPL and u-boot.img.

They can be install to an SD card by:

```
sudo dd if=SPL of=/dev/$dev bs=1k seek=1 oflag=dsync
sudo dd if=u-boot.img of=/dev/$dev bs=1k seek=69 oflag=dsync
```

2.3 u-boot splash screen

The Wandboard u-boot enables a u-boot splash-screen displayed on an attached HDMI monitor. In the boot_logo folder there is a script that converts and resizes common image formats to the expected bmp.gz file format.

The simple approacs would be:

```
% cd boot_logo
% ./mkbootlogo.sh wandboard-720x480-bw.png
# dd if=out.bmp.gz of=/dev/\$dev bs=520k seek=1 count=1
```

2.4 Linux kernel

To compile the linux kernel configure the kernel with

```
% make wandboard_defconfig
% make -j4 uImage modules
```

The Ubuntu kernel configuration is present as a file in the kernel folder (wandboard_ubuntu_config)

The Wandboard board file is in `/arch/arm/mach-mx6/board-wand.c`, and is the natural starting point for kernel hacking.

Note: the boardfile is for the module only. Components (that require drivers) on the baseboard should be placed in the corresponding baseboard file. As of now there is just one baseboard file (baseboard-wand.c) containing the initialization of the sgtl5000 codec.

The wireless driver should be compiled as modules so firmware can be loaded runtime. Also note that the Wandboard kernel uses the new fullmac (brcmfmac) driver instead of the old (bcm4329) driver.

3 Drivers

3.1 Revisions

As of the time of writing the Wandboard comes in three revisions, rev A, rev B and rev C1.

The first two, A and B, have no software differences. The same software will work on both. The hardware changes in rev C1 requires corresponding software changes.

The most notable hardware change in C1 compared with revs A and B, is the use of a bcm4330 WiFi/Bluetooth chip, instead of the bcm4329 as in revs A and B.

Revision A/B and C1 can be detected by reading the value of GPIO 60. This GPIO has a hardware pull-down on revision A and B, and a hardware pull-up on revision C1.

One way to detect the chip revision from userland is

```
#!/bin/sh

chip_version=0
if [ -f /sys/class/gpio/gpio60/value ]
then
chip_version=‘cat /sys/class/gpio/gpio60/value‘
fi
```

The variable $chip_version is 1 on rev C1 and 0 otherwise.

3.2 Bluetooth

The bluetooth part of the WiFi chip requires a userspace utility to load firmware. The utility, named brcm_patchram_plus, is included (with source code) in this release.

For Wandboard revisions A and B, the WIFI chip is a bcm4329 compatible, and for revision C1, the chip is a bcm4330 compatible. The firmware files and procedures to load them differ slightly.

The recommended procedure for loading the firmware on bcm4329 is:

```
# brcm_patchram_plus --timeout=6.00 --patchram /lib/firmware/
 brcm/bcm4329.hcd --baudrate 921600 --use_baudrate_for_download
 /dev/ttymxc2
# hciattach /dev/ttymxc2 any 921600
# hciconfig hci0 up
# hcitool -i hci0 scan
```

and on bcm4330

```
for i in ‘seq 1 5‘
do
   /usr/local/sbin/brcm_patchram_plus -d --timeout=6.0 \
       --patchram /lib/firmware/brcm/bcm4330.hcd \
       --baudrate 3000000 --no2bytes --tosleep=2000 \
       --enable_hci /dev/ttymxc2 > /dev/null 2>&1 &
   sleep 10
   if ! ( hciconfig -a | grep UP > /dev/null 2>&1 )
   then
      echo 0 > /sys/class/gpio/gpio149/value
      skill -9 brcm_patchram_plus
      sleep 1
      echo 1 > /sys/class/gpio/gpio149/value
      sleep 1
   fi
done
```

Where /lib/firmware/brcm/bcm4329.hcd is the 4329 bluetooth firmware, and /lib/firmware/brcm/bcm4330.hcd is the 4330 firmware.

The procedure for bcm4330 is similar, but more involved. The firmware loading can fail occasionally, and a check for this is needed.

3.3 Wireless

The procedure for WiFi is less compilcated than the one for Bluetooth. There is a driver patch that will load firmware files with different names depending on the wifi chip type. Using this driver, the filesystem needs to

contain the firmware files for both 4329 and 4330 (with specific names and paths), and the driver will then automatically load the right firmware.

Just make sure the firmware and nvram files are located in /lib/firmware/brcm/, with filenames:

- bcm4329_fw.bin
- bcm4329_nvram.txt
- bcm4330_fw.bin
- bcm4330_nvram.txt

This way the right firmware will be loaded by the kernel module upon insertion.

3.4 Audio

The wandboard features three audio devices

- The sgtl5000 codec
- S/PDIF
- HDMI audio

One possible ALSA configuration (.asoundrc) for these is:

```
pcm.sgtl5000audio {
        type hw
        card 0
        channels 2
}
ctl.sgtl5000audio {
        type hw
        card 0
        channels 2
}

pcm.imxspdif {
```

```
        type hw
        card 1
        channels 2
}
ctl.imxspdif {
        type hw
        card 1
        channels 2
}

pcm.imxhdmisoc {
        type hw
        card 2
        channels 2
}
ctl.imxhdmisoc {
        type hw
        card 2
        channels 2
}

pcm.copy {
        type plug
        slave {
                pcm hw
        }
        route_policy copy
}
```

To play a WAV over s/pdif can then be done by: `% aplay -Dimxspdif file.wav`