Intern roadmap

July 24, 2012

Contents

1 Preliminary research 1

2 Porting to Galaxy S II 1

3 Compiling for the emulator 2

4 Porting to the Beagle Board 2

5 Editing the manifest file for the Ideos 2

6 Build B2G for the Ideos 2

7 Install ClockworkMod recovery and create a backup 2

8 Flash a custom ROM 2

9 Flash B2G onto the phone 3

10 Make sure the kernel is up 3

11 Get the adb daemon to start 3

12 Get b2g to start 3

13 Get gdbserver to start 3

14 Tried fixing the GPU 4

15 Fixing the display 4

1 Preliminary research

The kick-off week (and the two weeks after) were used for research. Documentation done:

1. Setting up local environment
2. Build procedure for the Samsung Galaxy S II

2 Porting to Galaxy S II

This is fairly straightforward. Firmware version on the phone is checked by extract-files.sh, meaning that the phone will have to be upgraded to Ice Cream Sandwich.

Compilation was done using the new build system (the old one at http://github.com/andreasgal/B2G/ is no longer used).

3 Compiling for the emulator

Attempts were made to compile for the Android emulator. However, this failed because the new build system compiles for the emulator successfully only on a 64-bit computer.

It was not determined as to how useful the emulator is likely to be during the course of such a porting process since it was never used. However, in all likelihood, it would have only been useful for application development.

4 Porting to the Beagle Board

Attempts were made to port B2G to the BeagleBoard C4.

Ice Cream Sandwich was ported onto the board. To what extent this worked was not determined. However, the audio was tested and was seen to be working.

The attempt to port B2G itself was abandoned soon after because of non-availability of support for the device. The team was unable to find a suitable candidate for the "device" directory, without which porting becomes a much more difficult job.

Also, the team had fixed upon Huawei Ideos U8150 as the primary porting target by this time. Since the two targets had completely different processors, porting to one would not "help" in the process of porting to the other.

5 Editing the manifest file for the Ideos

In order to port to the Ideos, the first step was to create a suitable build tree. This involved getting the correct device-specific "device" and "vendor" folders. These were obtained from https://github.com/ColdFusionX/android_device_huawei_u815 and https://github.com/psyke83/proprietary_vendor_huawei respectively. The device folder was forked and a branch named "ideos" was created for development purposes.

The vendor folder itself is not an absolute necessity as long as an extract-files.sh script is present (and working) in the device folder. However, if available, it reduces the effort involved in debugging extract-files.sh and matching firmware versions.

More changes were made to the manifest file due to build errors. These are documented elsewhere.

6 Build B2G for the Ideos

Several errors were encountered during the build process. These are documented here. Most of the errors were solved by consulting, in turn, the internet (google and stackoverflow), mailing lists, and then posing questions on the respective IRC channel.

7 Install ClockworkMod recovery and create a backup

ClockworkMod for the Ideos can be obtained from here. Installation of ClockworkMod has been explained in the main guide document.

In order to flash the ClockworkMod image, the fastboot program is required from Android developer tools. This has been documented along with the initial setup.

A backup of the existing system is taken. For further safety, this recovery image is copied onto a development computer.

8 Flash a custom ROM

CyanogenMod 9 (ICS) was obtained from <here> and flashed onto the phone via recovery mode.

Since the B2G build tree uses ICS revisions for all of its android-related sources, it was worth while checking whether or not CM9 worked on the phone. The result was that it worked, albeit with reduced performance (when compared to Froyo, which was the stock OS). Also certain features, such as the gallery, did not work.

Additionally, the backup was verified to be working (by reinstalling Froyo).

9 Flash B2G onto the phone

For this, flash.sh needs to be suitably edited. A new case statement needs to be added for the Ideos. This can be based off Galaxy Nexus (maguro), or any other device that uses fastboot.

10 Make sure the kernel is up

This was the stage at which the team met its first major roadblock. The phone was in a state where the initial AIRCEL screen would display, but then immediately restart. This would happen repeatedly.

The reason for this was that the bootloader was not able to load the kernel into memory properly. The bootloader works by looking for the kernel's base address in bootimg.cfg. It then reads bytes of memory starting at this base address from the phone's flash memory and loads the program into RAM. However, at this stage, the kernel base address was incorrect, therefore the bootloader was loading meaningless streams of data into memory.

This was corrected by setting the BOARD_KERNEL_BASE setting in BoardConfig.mk to an appropriate value (by comparing with the values in the stock bootimg.cfg and in CyanogenMod's bootimg.cfg.

Additionally, a lot of experimentation was done with the init program and init.rc script. This enabled the switching-on or switching-off of various modules, in order to test the independent working of each module. Also, the team discovered that kernel logs can be displayed on the home screen. Details are, again, available in the main guide.

11 Get the adb daemon to start

After the kernel came up, it was still not possible to obtain a shell into the device. Kernel logs could be seen on the phone screen. Inspection of these logs (by recording a video and replaying in slow-motion) revealed that the kernel itself was working. The lsusb command revealed that the device was detected (after connection), but adb shell did not work. This meant that adbd had not started.

The call to start adbd is present in init.rc. It is started on activation of a certain property. It then seemed that the property never got activated. Hence, a call to start adbd was made from a property that was always activated. This started adbd on the phone. The ability to use adb shell showed that both the USB driver as well as the shell (sh or bash) worked with no further debugging required.

Once adbd was running, detailed log messages could be produced by using adb logcat (Android logs) and adb shell dmesg (kernel logs).

12 Get b2g to start

The next major roadblock was in getting the main process of the B2G operating system, called "b2g", to start. The process was stuck in a position similar to the kernel - it was starting, hitting an error, dying, and then starting again.

The reason for this error was because gecko has not been written in a compatible manner for ARMv6. It only works properly with ARMv7 devices. However, setting the architecture flag in the compiler fixes the problem:

```
ac_add_options --with_arch=armv6
```

in gonk-misc/default-gecko-config.

13 Get gdbserver to start

In order to better debug b2g as well as other programs on the system, it is extremely useful to get gdbserver started on the phone. This is usually achieved easily by executing the run-gdb.sh script present in the main B2G folder (which debugs the b2g program by default).

A short note on how this works:

1. gdbserver starts on the phone, and begins to debug a certain program. This binary need not have any debugging symbols.

2. gdbserver communicates to the host computer via TCP (using the port number with which it was called). It waits for commands from the host.

3. adb forward is used to forward commands to the TCP port on the phone from a TCP port on the computer.

4. gdb is started on the host computer (note that this must be the toolchain gdb, i.e. arm-linux-androideabi-gdb). It is called on a binary present on the host system. This binary must have debugging symbols.

5. gdb is instructed to forward commands via TCP using the host port. This is achieved by using the "target remote" command.

6. Debugging can now be done as if debugging on the host computer itself. Execution alone is done on the target phone.

gdbserver was not starting because the default gdbserver provided by the B2G build system was not compatible with the phone. Another gdbserver had to be taken from Android's Native Development Kit (the ndk/ folder in the B2G tree).

Also, the run-gdb.sh script calls gdbserver with the "–multi" option (which makes gdbserver restart once it dies), and the "target extended-remote" command is used in the host gdb. The "–multi" option had to be removed and "target remote" had to be used.

Note that binaries with debugging symbols can be found in the symbols/ folder in the out directory. Gecko-related binaries (such as b2g) can be found in the objdir-gecko/dist/bin/ folder.

14 Tried fixing the GPU

Even though the b2g process started, the display was not being displayed. This was assumed to be a problem due to the display driver. Incorrect sources led the team to believe that an Adreno 200 GPU was present on the mobile phone. Therefore attempts were made to compile drivers for this GPU.

Trying to compile the display device driver (hardware/qcom/display) created more build errors. These were fixed in a similar manner - documentation of this can be found here.

The related libraries were libgenlock, libgralloc and libqcomui. In fact, it was the logcat errors "Cannot load EGLConfig" and "Unable to find libgenlock" that led the team to start looking for these libraries.

Once it was determined that the GPU did not exist and that all rendering was being done in software, the team abandoned this path.

15 Fixing the display

The problem with the display was debugged using run-gdb.sh. It was discovered that the "Just In Time" compiler was posing issues. Details are present in the main guide document.