Guide to Porting B2G

Sasken Communication Technologies Ltd.

Version: 1.0
Date: 20 July 2012

CTO / Intern

Revision History

Version	Approval Date	Description of change(s)	Author(s)
1.0	20 July 2012	First draft of report	Praveen Venkatesh, S R V Vishwanath, Vivek Kumar Bagaria

Contents

1 Summary

For the uninitiated, “Boot 2 Gecko” (hereafter shortened to B2G) is Mozilla corporation’s new operating system for mobile phones. As of the writing of this document, this operating system has only been ported to high-end phones such as the Samsung Galaxy S II (GT-I9100), Galaxy Nexus and Galaxy Nexus S. The primary aim of this project was to attempt the porting of B2G to a low-end android phone and document the issues faced in this process.

To this end, the team chose the Huawei Ideos U8150 as the primary porting target, reasons for this being its extremely low cost and large support in the android developer community. Over the course of the project, however, B2G was also successfully compiled for two other mobile phones - the Samsung Galaxy Y (and by extension its dual-sim version, the Samsung Galaxy Y DUOS) and the HTC Explorer. Testing of B2G was performed on the Samsung Galaxy Y DUOS, but results were not favourable, particularly because this phone has much less support in the open source community. Testing was not performed for the HTC Explorer due to time constraints, but the team believes that a port to this phone would be much easier to attempt.

This document describes the motivation behind the project, and then attempts to be a complete guide for someone attempting a follow-up of this project. This it does by presenting a break-down of the B2G source code, system setup information, valuable online code and information sources, the build process, expected errors and possible solutions, a little background on kernels and architectures and where to get help if required.

2 Background

2.1 Motivation for this port

There are several reasons for “choosing B2G over android”, and although that is not what this document intends to prove, it serves to justify the expenditure of effort on this porting task.

2.1.1 B2G is more open source than Android

Android’s contribution system is controlled by Google, as a result of which any patches must go through an approval process laid down by Google, and not by the community. It is therefore not an open, community driven project such as Mozilla’s Firefox. It has also scored poorly on studies performed to analyse the openness of various open source programs. Google also has the power to decide which versions will be open source and which versions won't (eg: Honeycomb). B2G, on the other hand will always remain open source and its design decisions are more community-driven.

2.1.2 Apps written in HTML5

All applications written for B2G are written in HTML5 (to be rendered by the Gecko rendering engine). This includes online as well as offline applications. What this means is that the much larger community of web developers can create applications for mobile phones without knowing a hardcore programming language such as Java. Also, any existing applications available online (for PCs) can be ported to a phone with relatively small effort.

2.1.3 Complete control over low-level devices and API

B2G through an Application Program Interface allows for HTML5 applications to control devices and peripherals on the phone while maintaining security. This allows for web applications to control devices such as the camera, vibrator and the display framebuffer.

2.2 Languages

The code-base of B2G is very large. Thus we will require lots of tools to understand the code and debug the code. Languages used :

1. Gonk - C, C++
2. Gecko - C, C++
3. Gaia - HTML5, CSS, JavaScript - Knowledge of this is only required for application development, not for porting itself.

A thorough knowledge of C, C++ is required.

2.3 Choosing a phone

2.3.1 Ideos

Advantages

1. Low Cost - Rs 4000/-
2. CyanogenMod ICS was available
3. The vendor and device folder was available on github
4. Support - This phone had good support on the internet

Disadvantages

1. Armv6 Processor - B2G developers are concentrating on armv7, thus lots of changes are required to build the code for armv6 processor. Their is no official toolchain available for armv6.
2. It has no GPU, therefore, all rendering is done by software, making B2G extremely slow on this device.

2.3.2 HTC Explorer

Advantages

1. Armv7 Processor
2. Low cost - Rs 7,500/-
3. Custom-Rom for ICS was available on XDA forums
4. The vendor and device folder was available on github

2.4 System setup

This is described in a separate file.

2.5 Pre-installation procedure

2.5.1 ClockworkMod (CWM)

ClockworkMod is a custom Android recovery tool, which can be used to take a complete backup of the existing phone. It does this by capturing each partition in the phone as a separate image and then saving these images (along with a hash) in a folder in the SDCard. Prior to flashing anything onto the phone, ClockworkMod recovery must be installed in order to prevent “bricking” the phone. Here’s a good tutorial - http://www.addictivetips.com/mobile/what-is-clockworkmod-recovery-and-how-to-use-it-on-android-complete-guide/

Installation of ClockworkMod can be done in two ways depending upon how ClockworkMod is available for your phone. Here are the official ports: ROM Manager

Method 1 (ClockworkMod.img)

For this, you will need a phone-specific flashing tool. For most phones, fastboot (from the Android platform tools) will work. For Samsung phones, Heimdall or Odin can be used.

1. Start the phone in download mode. This is usually achieved by pressing the power, volume down and end call (red) keys simultaneously.

2. Connect the phone to your computer and check whether or not it is detected:

```
$ fastboot devices
??????????     fastboot              # device is detected
```

3. This command then tells fastboot to flash into the recovery partition:

```
$ fastboot flash recovery /path/to/clockworkmod.img
```

Method 2 (ClockworkMod.zip)

In this case, no flashing tool is necessary.

1. Get the ClockworkMod recovery zip file from the internet.

2. Add the zip file to the mobile’s SD Card.

3. Switch off the mobile and reboot the phone in recovery mode. This is usually achieved by pressing the power, volume up and pick call (green) buttons simultaneously.

4. Choose the option to “Install update from SD Card”

5. Choose the ClockworkMod.zip file and install it.

Once ClockworkMod is installed, switch of the phone and reboot it in recovery mode. This can usually be achieved by pressing the power, volume up and pick call (green) buttons simultaneously. Follow the menu to take a backup (navigation is achieved by using the volume, power and home buttons). This will automatically create a folder in the SD Card.

Once the phone is backed up, we have secured our phone and it can be recovered from all possible situations. In the event that the recovery partition gets corrupted, simply reinstall ClockworkMod. The recovery file itself will not be affected as long as the SD Card is not damaged. To be even more safe, a copy of the recovery folder can be made from the SD Card onto an external disk.

2.5.2 CyanogenMod 9

CyanogenMod is a customized, aftermarket firmware distribution for several Android devices. It is designed to increase the performance of the mobile and offers new features.

CyanogenMod 9 is the custom mod for Android 4.0.4 (ICS). CyanogenMod is usually available as a zip package and can be installed in the same manner as ClockworkMod.

3 Approach

This section will describe a rough roadmap followed through the porting process.

1. Port B2G to a similar phone on which it is known to work. Also, compile for the emulator. In this team's case, the only available reference was the Samsung Galaxy 2 (refer this guide).
2. Edit the B2G manifest file. For this, search for the right device and vendor folders (if the vendor folder is not available, there should at least be an extract-files.sh in the device folder) and include them in the manifest.
3. Get a kernel for the phone (refer separate section).
4. Build B2G and sort out the build errors (errors documented separately).
5. Find rooting and flashing software for the concerned phone. For the Huawei Ideos U8150, rooting can be achieved by using z4root and flashing by using fastboot.
6. Find a ClockworkMod Recovery tool for the phone, install it, and take a backup of the existing Android OS. If a ClockworkMod (or other such similar recovery tool) does not exist, then all hopes of porting to such a phone (reversibly) can be abandoned.
7. Find a custom ROM, such as CyanogenMod for the phone, (preferably version 9 - Ice Cream Sandwich) and install it. If the custom ROM does not work, then once again, it becomes extremely hard to port because the device directory will have to be written from scratch.
8. Flash B2G onto the phone.
9. Make sure the kernel is up (refer the debugging section for these points).
10. Get adbd running on the phone.
11. Make the b2g process start.
12. Get gdbserver to start on the phone. Get run-gdb.sh to work.
13. Debug the device drivers.

Please refer this document for a more detailed roadmap.

4 Design and Implementation

4.1 Getting The B2G Source Code

The B2G source code is available as a git repository from http://github.com/mozilla-b2g/B2G.git.

B2G uses the same mechanism as Android to manage its source tree: git and git-repo. git-repo is a python-based wrapper over git which manges multiple repositories. For a quick tutorial, look here.

A repository is initialised by using "repo init". "repo help init" will provide details. After this, the .repo folder contains a manifests repository. This contains a default.xml file which is the manifest file that will by synchronised with if you run repo sync in the main folder. The manifest file contains a list of git repositories which are supposed to be present in the build tree. Each record on the list consists of a remote url path and the directory path where the repository lives.

4.2 B2G Build System

4.2.1 config.sh

config.sh contains a list of devices to configure for. It essentially removes all existing manifests (discarding changes!) and clones a new manifest from the remote url hardcoded within it. It then checks out the branch according to the device being configured. Therefore, while adding repositories to the manifest, it is safest to commit and push after every change.

4.2.2 build.sh

build.sh initiates the build by first setting up the environment: this it does by calling build/envsetup.sh, which sets a number of variables and functions in the environment. A function called "lunch" is called to choose the product to be built and set up product-specific build variables.

Each folder inside the build tree has an Android.mk file. This file usually contains calls to recursively execute all Android.mk files in subfolders. Therefore, in order to disable a module, the easiest way would be to rename the Android.mk file to something else.

Android.mk files have an extremely simplistic mechanism. There are several functions which can be used to clear local variables, build shared libraries, recursively call sub-makefiles, etc. These are amply described in build/core/definitions.mk. Each Android.mk file roughly follows the following pattern:

```
clear variables
define local C includes (directories to be added to the include path in this folder)
define local source files
define local shared libraries (libraries to be linked against)
define local module (name of the output library after compiling)
call a build-library function
```

Throughout the build process, many errors were resolved by adding include paths via LOCAL_C_INCLUDES or by adding extra source files via LOCAL_SRC_FILES. All these changes can be seen in the commit history of the respective repositories.

4.2.3 flash.sh

flash.sh is used to flash all or selected partitions of the device. When porting to a new device, a new "case" statement will have to be added to flash.sh depending on what flashing software it uses.

4.3 B2G Directory Layout

The B2G directory layout is subject to changes depending upon the device, but broadly there are a number of paths which remain constant, even though the respective repository sources may change.

4.3.1 bionic

Bionic is a stripped down version of the standard C and C++ libraries, specifically made for android. As such, it is much lighter and more amenable to porting across different platforms.

4.3.2 build

This is the directory which controls how the source tree is built. Apart from containing the makefile code which makes the Android.mk build system (described in the previous section) work, it also contains architecture-specific controls. build/core/combo/arch/arm contains architecture-specific makefiles (one for each arm instruction set version). Depending upon the architecture specified in BoardConfig.mk (in the device directory), a corresponding .mk file will be chosen and executed from this directory.

4.3.3 device/<vendor>/<device>/

The device directory is by far the most important directory as far as the porting job is concerned. This directory will have to be cloned from someone who has already ported a custom ROM to the phone, for example, CyanogenMod. This directory contains a number of important files that set multiple configuration settings:

AndroidBoard.mk Contains lines to enable/disable kernel compilation. includes the non-open source counterpart from the vendor directory, viz. AndroidBoardVendor.mk.

AndroidProducts.mk Has a link to full_<device>.mk.

BoardConfig.mk Contains a large number of configuration options. Many of these options appear inside C++ files as defined macros. Therefore, they control which pieces of code are compiled and which are not through #ifdef statements.

Other options control whether some files are created or copied. For example:

```
TARGET_PROVIDES_INIT_RC := true
TARGET_PROVIDES_UEVENTD_RC := true
TARGET_PROVIDES_INIT_TARGET_RC := true
```

These lines say that the init.rc file is present in this device folder and that it need not be created by B2G. This is an important part of the porting process, since we would prefer to use the phone's stock init.rc file instead of the one that B2G creates.

The BOARD_KERNEL_* commands are used to set kernel address values in bootimg.cfg. Depending on these values, the bootloader will try to load bits into memory for executing the kernel.

device_<device>.mk Contains some very important configuration settings. The product name, device, model and brand are reset.

The PRODUCT_PACKAGES setting gives the list of libraries to be added to /system/lib/hw in the phone. These libraries are used by device drivers. Therefore, when adding a new device driver to the hardware directory, it would be necessary to add a package to ensure that the library is copied into the phone.

The PRODUCT_COPY_FILES setting gives a list of files to be directly copied into the phone's system. Most of these are prebuilt libraries that are pulled from the phone at the time of running config.sh (or from the vendor directory). The format is:

```
PRODUCT_COPY_FILES += \

    source/path/wrt/B2G/folder/file1:destination/path/wrt/out/folder/file1 \
    source/path/wrt/B2G/folder/file2:destination/path/wrt/out/folder/file2 \
    ...
```

full_<device>.mk Inherits device configuration from <device>.mk. Also, sets some generic make variables.

<device>.mk Contains calls to each of the other makefiles: device_<device>.mk, and some other such makefiles from the vendor directory. Also specifies path to the kernel, if we are using a prebuilt kernel, and adds it to PRODUCT_COPY_FILES.

extract-files.sh This is optional. It is not required if you can find a vendor folder repository that already has all device drivers.

extract-files.sh is called by config.sh, at the time when Android is still running on the phone. It makes use of ADB to pull essential closed-source device driver libraries and other such files that cannot be compiled from source. In the case of the Samsung Galaxy S II, this file checks the firmware version of the phone before pulling files. It demands that the phone be upgraded to Ice Cream Sandwich (as of the writing of this document), and otherwise fails.

kernel (binary) Prebuilt kernel binary, usually the stock kernel pulled out of the phone (or CyanogenMod's kernel). This is easier than compiling a kernel from source.

libraries Depending upon the phone, there may be several device driver sources present in the device folder. Examples include (courtesy of Huawei U8150):

```
libaudio
libcamera
libcopybit
libgralloc
liblights
```

and more.

prebuilt The prebuilt folder contains many files which are copied directly into the out directory by using PRODUCT_COPY_FILES. It usually has an incomplete replica of the phone's directory structure:

```
init.rc
init.<device>.rc
ueventd.rc
initlogo.rle
system/build.prop
system/bin/
system/xbin/
system/lib/
usr/
etc/
```

and so on. The stock init.rc file should be added to this folder along with a PRODUCT_COPY_FILES directive in device_<device>.mk.

setup-makefiles.sh Creates two makefiles in the vendor directory: <device>-vendor.mk and <device>-vendor-blobs.mk. The former file simply inherits the latter. The latter file contains a host of PRODUCT_COPY_FILES directives. These are directly copied from proprietary-files.txt.

proprietary-files.txt Contains a list of proprietary files that are to be copied from the vendor directory into the out directory. Used by setup-makefiles.sh.

vendorsetup.sh Adds a lunch combo, viz. full_<device>-eng. For example, for the Ideos U8150, this would be:

```
add_lunch_combo full_u8150-eng
```

This enables

```
lunch full_u8150
```

to be called. However, the option still needs to be added manually to the config.sh file. Otherwise it does not appear as one of the options, although it is available.

4.3.4 external

This contains a number of external (non-android-specific) programs that need to be compiled. Examples include wpa_supplicant, apache-http, bluetooth, bzip2, dbus, dhcpcd, jpeg, etc.

4.3.5 frameworks

This directory contains source code for a number of libraries used by Android. It is not modified by B2G.

4.3.6 gaia

The gaia source tree. Not much compilation is involved here. Most of the files are just HTML and JavaScript files which are copied to the out directory.

4.3.7 gecko

This contains the main parts of B2G, including XUL and the b2g program. It is quite close to mozilla-central. More can be found out by reading the README file in this folder. The gecko build system is itself quite different from the android build system. It is therefore called by gonk-misc. The gecko build process has been documented as a separate section.

4.3.8 gonk-misc

The gecko directory itself does not contain an Android.mk file. It is instead started by gonk-misc. This directory contains a few important files:

init.b2g.rc This is copied to the out directory. It must be imported by the main init.rc script. It contains commands that are used to start b2g-relevant processes.

default-gecko-config This is the equivalent of mozilla-central's mozconfig file. It contains the compiler options used to build gecko. One important option is –with-arch=armv6 (for armv6 devices). Otherwise, it defaults to armv7. Without this option, b2g will not compile properly and hence will not start.

4.3.9 hardware

This directory contains device drivers for the various peripherals in the phone. Device specific drivers will have to be searched for and added to the manifest. For example, the Huawei U8150 required:

qcom/display Drivers for the phone's display. Without these, the display will not render.

qcom/camera Drivers for the phone's camera. These are compiled into a libcamera module.

msm7k Miscellaneous qualcomm drivers for audio, camera, lights and graphics

broadcom/wlan WiFi drivers.

4.3.10 kernel

This directory is optional - it is only required if a kernel has to be built from source for some reason. In case a prebuilt kernel is available, either by pulling out the stock kernel or from CyanogenMod, the binary can be added to the device directory and be copied directly into the out directory.

4.3.11 ndk

This is Android's Native Development Kit. It allows for compilation of custom code by android developers.

4.3.12 out

The out directory is where all the compiled code goes. The host subdirectory contains tools required for building the rest of B2G on the host. The out/target subdirectory contains the final binaries (with and without debugging symbols) that are compressed into image files to be flashed onto the phone. An overview of the out/target/product/<device>/ directory (which is largely referred to as simply the "out" directory in this document) is given below:

boot.img, ramdisk.img, system.img, userdata.img, recovery.img The boot, system and userdata images are flashed into the boot, system and data partitions of the phone respectively by flash.sh. The recovery image is also flashed by default by heimdall, but not by fastboot. The ramdisk image is essentially an image of the root folder along with the kernel. It exists as a part of boot.img. These files can be opened for reading by using the abootimg and unyaffs programs.

data Contains all the gaia files. It will also contain any media files that go into the data partition. It gets compressed into userdata.img after building.

obj Contains several compilation intermediates. Depending upon the source tree, it may also contain objdir-gecko, which contains everything compiled by gecko, along with debugging symbols.

recovery Contains clockworkmod recovery compiled for the phone. This actually does not need to be flashed if clockworkmod was already installed before flashing B2G. Gets compressed into recovery.img.

root This contains the directory structure which goes into the / directory on the phone. Gets compressed into ramdisk.img and then into boot.img. It also contains the final versions of files such as inti.rc, ueventd.rc (and their derivatives), the init binary and a default.prop.

system Contains most of the binaries used by the phone. These are compiled without debug symbols in order to make them as light as possible.

4.4 Gecko build system

Building gecko is different compared to the android build system. The gecko build system comprises of two main files at the user end, configure and client.mk. client.mk is the make file for building the source. configure, as the name suggests configures the build system according to the input configuration option.

To build gecko, use the following command:

```
$ cd path/to/gecko
$ make -f client.mk <options>
```

In case options are not given, it builds firefox. The options field will decide the app to build, the toolchain to use, the architecture to build for and so on. Some of the options are:

```
--with-arch=<target-arch> # For example, the present project target-arch is armv6
--enable-application=<app> # b2g in our case
--enable-media-plugins
```

However, a typical 'non firefox' build requires a lot of options to be passed. Hence we use a config file, which is called MOZCONFIG file. The MOZCONFIG file will have two types of configuration options:

1. *mk_add_options:* These will be passed to the make file. Some options are MOZ_MAKE_FLAGS, MOZ_OBJDIR
2. *ac_add_options:* These will be passed to the configuration file. Some options are –with-arch=, –with-thumb= .

4.4.1 Object directory

The compiled output is dumped in the object directory. The object directory is passed as an option to client.mk. A typical example is:

```
$ MOZ_OBJDIR=../objdir-gecko make -f client.mk
```

This will create a directory called *objdir-gecko* at the top level, in which all the compiled output will be stored. Make sure that object directory is not the same directory as the gecko folder. This may give rise to lot of errors.

Also, never do *./configure*, though it seems intuitive.

4.4.2 Building gecko for B2G

The process of building gecko for B2G is automated through the android build system. The building of gecko is invoked from the Android.mk file in gonk-misc folder. The config file, *default-gecko-config* file is also found in the same folder. The main configuration options to be noted, apart from the standard option:

```
ac_add_options --with-arch=armv6
```

In case you are not sure of the fpu:

```
ac_add_options --with-fpu=toolchain-default
```

In case the processor doesn't have neon processor:

```
ac_add_options --disable-neon
```

MethodJIT may give rise to some illegal instructions error, in which case you might want to:

```
ac_add_options --disable-methodjit
```

5 Learning

5.1 Domain and Technology

This project falls broadly under the intersecting domains of B2G, Android, Linux and device drivers. A good understanding of one or more of these areas will strongly help in undertaking such a project.

5.1.1 Overview of the B2G architecture

Although most of this is described in much greater detail online, this section will attempt to summarise the same. Boot 2 Gecko comprises three layers, Gonk (the operating system), Gecko (the rendering engine) and Gaia (the user interface):

Gaia Everything that the user sees on the phone in a complete version of B2G will in one way or another be a part of Gaia. All applications, screens, buttons, special effects, etc. are implemented through HTML5, CSS and JavaScript. This UI is called Gaia. It usually does not come in the way of porting.

Gecko This layer comprises of the engine which renders the HTML, CSS and JS (Gaia files), and hardware abstraction layers to interface with the operating system (Gonk). The main b2g process is defined within Gecko, along with other libraries such as libxul. They form the core processes that make up the B2G operating system (and differentiate it from Android).

Gonk This is the "operating system" layer. Gonk is mostly just Android, stripped of the DVM and all java files. It contains the linux kernel and device drivers. Many of these are taken directly from the Android Open Source Project. Gonk is where most of the porting effort lies, in ensuring that the kernel works, the device drivers work, etc.

Booting The boot process of B2G is very similar to the boot process of any other linux system, since Gecko is practically a small linux distribution. Much of B2G's boot process (especially init and init scripts) can be understood by looking at Android's boot process. The init systems of both are identical and the init script syntax of B2G is taken directly from Android.

Userspace Process Architecture The main init script includes an init.b2g.rc script, which contains calls to the main processes that make up B2G. This includes b2g itself, the mediaserver, etc. The b2g process is the main system process which runs with high privileges. Applications run as low-permission content processes that communicate with b2g via IPDL. B2G controls the applications' permissions in this manner,

5.2 Tools

5.2.1 Bash

We will be using bash to understand and write scripts. Some of the important bash scripts (which will need editing) are config.sh, build.sh and flash.sh. These are described more thoroughly in the section under B2G's build system.

Some frequently used bash commands are:

locate <file_name> Locates the file "file_name".

grep <pattern> Search word patterns in one or more files.

Bash scripting will also be handy to automate certain tasks.

Tutorial - www.freeos.com/guides/lsst/

5.2.2 Make

This file is used to compile the code. This reduces the compilation time during recompilation. One of the most important makefile is Android.mk. This will be explained later. Other important make files are BoardConfig.mk, AndroidBoard.mk etc. When we new code in a folder, the complication procedure has to added to the respective Android.mk.

A few short notes on Android.mk files:

- When we build B2G, it searches all the folders and sub-folders recursively for Android.mk file.
- This file defines environment variables.
- First step to understand the contents of a folder is to read it Android.mk. This gives us the brief idea of the relations between the files.
- Compilation of B2G takes a lot of time. We can reduce it by compiling only a part of code. We can rename Android.mk in a particular folder to Android.mk.bkp if we want to avoid the compiling that part of code.
- Complete Documentation - Source 1, Source 2
- Short tutorial - http://wlug.org.nz/MakefileHowto

5.2.3 git and git-repo

git is a version control tool. This is helps us to keep track of all the changes made and to recall versions when required. This is handy when the code size is huge. All components of B2G (and Android) are available only as git repositories. Look here for a tutorial.

git-repo is a python-based wrapper over git for managing multiple repositories simultaneously by using a manifest file. Details about this tool are documented in a separate section.

5.2.4 Android Debugging Bridge (ADB)

This is command line tool through which we can communicate with the device. This is like having shell access to Linux on phone. This has few commands compared to bash shell.

We can push/pull files and folders from the computer (client) to/from mobile (server).

adb logcat displays all the logs from gonk, gecko and gaia. Thus all the errors and warnings can be noted. We can even add our own LOG messages in the code. Thus this is a essential method for debugging.

Documentation - http://developer.android.com/tools/help/adb.html

5.2.5 gdbserver and gdb

This a very simple and important tool. Gives the exact line in the code which has errors. Can be used to set break points in the code and execute line by line. run-gdb.sh is used to start gdbserver in the mobile.

5.2.6 Linux

We strongly recommend Linux as the operating system to be used for developing. According to the B2G docs, 64-bit linux and Ubuntu version 11.10 is is preferred. Compilation will take a lot of time, thus a computer with a fast processor (with more parallel processing capability) is preferred. Understanding the Linux architecture will be an advantage.

5.3 Issues faced and solutions

Please refer this document.

5.4 Open issues

Device drivers are yet to be debugged. Below is a short description.

5.4.1 Display driver

The display driver was not working because of a problem with Just in Time (JIT). This problem has been fixed in the latest commit.

However, even after the driver itself was fixed, no applications show up on the screen. Evidence that the driver was fixed includes display of signal strength, battery level and volume (when volume buttons are pressed). It takes some time to display, however, since rendering is being done by software (making it quite slow).

This issue is mostly gaia-related, meaning that the correct files are not being picked up for rendering, or that the files themselves exist, but cannot be rendered for some strange reason.

5.4.2 Camera driver

The camera driver can be checked by using two command line tools, viz. record and recordvideo.

```
record /data/test.jpg 1
```

is supposed to capture an image and store it in the named file. The second argument has something to do with formats. It can be either 0 or 1. Currently, this program crashes due to a segmentation fault. The exact problem ought to be discoverable by using run-gdb.sh.

```
recordvideo
```

run with no arguments, is supposed to create an output file /sdcard/output.mp4. This should be a 10 second recorded video. Currently, the command works, but the video has only a blank green screen and no sound. It is yet to be ascertained whether the driver (libcamera.so and those similar) is being called at all. If so, then they will have to be debugged.

5.5 Kernel

The kernel is the main part of Operating System. Along with other device drivers, it provides hardware abstraction layer (low-level APIs). The main tasks of kernel are

1. Allocating resources to different processes
2. Memory management
3. Handling of system calls
4. Handling various devices

Further information can be found here.

5.5.1 Choosing the right kernel

Most of the companies officially release kernels for their mobiles for a particular version of Android. A company doesn't necessarily release kernels for future versions of Android, for a specific mobile. For example Huawei released *froyo* kernel for *ideos u8150* but they did not release kernel for *ics*. Thus we have to find the kernel from other reliable sources like CyanogenMod. Many of the reliable links have been documented under the heading 'Code Sources'.

5.5.2 Adding a custom kernel

Method 1 If the kernel image is available. Add the custom kernel to the path device/<device>/<name>/prebuilt/ and add a rule to make kernel. Add the following line can be added to device.mk file to make a rule to make kernel

```
PRODUCT_COPY_FILE += \
device/<device>/<name>/prebuilt/kernel:kernel
```

Method 2 If kernel source is available, include the following line in AndroidBoard.mk

```
ifeq ($(KERNEL_DEFCONFIG),)
KERNEL_DEFCONFIG := cyanogen_u8150_defconfig
endif
include kernel/AndroidKernel.mk
file := $(INSTALLED_KERNEL_TARGET)
ALL_PREBUILT += $(file)
$(file) : $(TARGET_PREBUILT_KERNEL) | $(ACP)
    $(transform-prebuilt-to-target)include
```

5.6 The ARM architecture

ARM (Advanced RISC Machine) is a RISC processor architecture, developed by ARM holdings (Quoted from Wikipedia). Unlike Intel, which develops the architecture as well as the silicon chip, ARM holdings only develops the architecture. It licenses its architecture to various companies who manufacture and sell them. Some of the companies include Atmel, QualComm, Broadcom, Freescale, etc. Companies like Qualcomm and Broadcom make System on a Chip (SoC). The SoCs contain the main application processor along with the telephone modem and a digital signal processor.

5.6.1 Instruction set

Instruction sets is the language of the processor. ARM has evolved its instruction set from *armv1* to *armv7,* with *armv8* being developed now.The instruction sets are backward compatible but not forward compatible. *armv6* code will work on *armv7* processor but the vice versa is not true. This will be one of the main issues when porting a code to an *armv6* platform which has been written for *armv7* platform. This will be the source for illegal instructions in the program.

Gecko, natively did not support armv6 instruction set. However, it supports it now. The JIT code may also cause some illegal instructions. Disabling it will make the code slow, but will solve the problem.

5.6.2 Floating point unit

Apart from the application processor, the SoCs will have a floating point unit. *armv6* architecture is complimented with Vector Floating Point unit. *armv7* on the other hand is complimented with NEON processor. They are not compatible with each other. The neon feature can be disabled when compiling gecko to make sure the code runs on non- NEON processors.

5.7 Runtime debugging tools

Once B2G is flashed, it will be necessary to debug it for errors. The following set of tools should help to debug the mobile, from booting to first sight of home screen

5.7.1 Kernel logging

It is possible to get the kernel messages onto the mobile screen. This will be useful if the kernel gets loaded but the mobile gets stuck at a certain point before b2g process starts. To do so, the kernel messages must be redirected to standard output, *tty0.* It must be changed in the *BoardConfig.mk* file. The following line should be edited

```
BOARD_KERNEL_CMDLINE := mem=211M console=ttyMSM2,115200n8 androidboot.hardware=u8150
```

to

```
BOARD_KERNEL_CMDLINE := mem=211M console=tty0,115200n8 androidboot.hardware=u8150
```

This would output all the kernel messages to the mobile screen. However, the messages scroll very fast. A workaround would be to take a video of the boot process and split the video into images and analyze them

5.7.2 Android Debugging Bridge

Android Debug Bridge (adb) is a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device(Quoted from *developer.android.com*). ADB can be used for the following:

1. Installing applications
2. Copying files from host to mobile and vice versa
3. Getting a log
4. Getting a shell for the mobile

The last feature can be used for debugging purpose. The shell is a standard sh/bash shell, with all the basic shell commands. Through it, the following two tools can be used for debugging:

dmesg It gives the kernel logging. This can be used to find out if the low level drivers are working and that the file systems are working as expected

logcat It provides a mechanism for collecting and viewing system debug output. Logs from various applications and portions of the system are collected in a series of circular buffers, which then can be viewed and filtered by the logcat command.

5.7.3 GDB

The gdb program is the most preferred way of debugging errors. Unlike a normal debugging session, in which a program is compiled and gdb is started, debugging for mobile requires a host program and a target program. A *gdbserver* program is required on the target mobile and *gdb* is required on the host. The *gdbserver* program must be forwarded to a port and should be attached to a program to debug it. *gdb* should be listening to the same port to output the debug information onto the console. A simple implementation would be

```
adb shell LD_LIBRARY_PATH=/data/local/gdbserver :8080 /system/b2g/b2g
gdb -x b2g.gdbinit out/target/product/u8150/obj/objdir-gecko/dist/bin/b2g
```

The gdbinit file defines the path to look for symbols and executables.

The file *run-gdb.sh* file in the B2G directory automates this whole process.

5.8 Debugging using logging

An alternative to gdb is using the traditional 'print debugging', i.e, using print statements to debug the program. To print statements, the android logging function will be used. This prints the statements to android logcat. The following lines must be included to log the statements to logcat

```
#include “android/log.h”
#define LOG(args...) __android_log_print(ANDROID_LOG_INFO, <logtag>, ##args)
```

The logtag is user defined. An example would be “gecko_egl” if EGL related logging is required. To log a statement, simple invoke *LOG* with the desired statement. The format of the statement is similar to printf statement. Example:

```
LOG(“This statement will be logged in android logcat”);
```

In most of the files, the android/log.h file will be included. Hence the `#define alone should be enough`

For logging in javascript files, use *dump* function. Same as printf, the statements will be printed out in the android logcat. Example:

```
dump(“Logging in java script\n”);
```

6 Conclusion

It is possible to port B2G to a low cost phone such as Huawei Ideos U8150, which uses an ARMv6 instruction set and has no GPU. However, it would not be possible were it not for existing device and vendor directories from CyanogenMod.

Porting of B2G was attempted on two other phones, Samsung Galaxy Y DUOS and HTC Explorer. While the attempt failed for the Galaxy Y DUOS (possibly due to the code having been compiled for the Galaxy Y, or possibly due to lack of resources and support from the Android developer community), time constraints prevented us from checking feasibility on the HTC Explorer, although compilation was successful.

All in all it can be said that the difficulty in porting was compounded mainly due to lack of support for ARMv6. Trying such a project on a phone that uses an ARMv7 instruction set processor is much more likely to yield good results (at least, at a much faster rate).

7 References

7.1 Code sources

Github - This is the main source for various code modules. https://github.com/mozilla-b2g

1. https://github.com/mozilla-b2g/B2G - Source code for b2g
2. https://github.com/mozilla-b2g/wpa_supplicant_8_lib- wpa supplicant version 8. This can be set in BoardConfig.sh (Verify).
3. https://github.com/mozilla-b2g/b2g-manifest- Sample manifest file

- https://github.com/CyanogenMod- This contains vendor and device folders for various standard mobiles. This is the first place to to search for STOCK folders
- https://github.com/alanorth- This is contains vendor and device folders for Huawei and some other standard phones
- https://github.com/Psyke83- This contains some more files pertaining Huawei.
- https://github.com/ColdFusionX- This also certain important folders.
- https://github.com/tilal6991- As of now this contains some folders the above contain.
- https://github.com/CommonFusionX- Contains few more commonly used files.

The above three are sources to find standard files. Always search this before searching whole of net.

- https://github.com/apitrace- Contains files for debugging of OpenEGL
- https://github.com/android- contains image of android. It will be rarely used. This is helpful when u want to change some folders which have their repo only here

android.googlesource.com - Since B2G has a stripped down version of android, many of problems which arises during compilation will have its solution in android documentation.

Some other useful sites -

1. codeaurora.org

7.2 Information sources

The above mentioned sites are mainly for code source. But we will need answers to lot of frequently arising doubts and errors.

1. **Mozilla wiki** (wiki.mozilla.org/B2G) - This is the starting point to know B2G basics. Some of the important links are The rough order of importance
   - B2G/Architecture
   - B2G/Porting (very important)
   - B2G/gdb B2G/Hacking
   - B2G/Debugging OpenGL
   - B2G/DeveloperPhone
   - B2G/FAQ
   - B2G/QA
2. **Mozilla developer network** (https://developer.mozilla.org/en/Mozilla/Boot_to_Gecko/) - This site is being updated currently.

3. **XDA Forums** (forum.xda-developers.com) - This is the most important site for all the queries.
   (a) Lots of Custom Roms are available
   (b) Lots of common build errors are posted and solved
   (c) It is advised to create a account if u have not don't have one because questions posted here are answered soon
   (d) Lots of tutorials are available - Eg. Rooting a mobile, how to use ClockworkMod.
4. **General Information** (gsmarena.com)- contains information for a layman. Eg, RAM, GPU, etc. Quick reference
5. **en.wikipedia.org** - Major phones have a wiki page. This has lot more information than gsmarena
6. **Others** -
   (a) http://kobablog.wordpress.com/2011/05/19/start-up-sequence-of-system-server-of-android-memo/- Basics of android debugging
   (b) http://linux-encyclopedia.blogspot.in/2011/05/android-init-language.html - Basics of init script.
   (c) http://www.freeyourandroid.com/guide/introdution_to_edify- Edify script, useful when you want to add Custom-Rom i.e, port your phone without tools like heimdall, fastboot.

7.3 Getting help

The following source will prove invaluable when information about a particular procedure is required

7.3.1 Internet Relay Chat

IRC is used for real time internet text messaging. It is designed for group communication in discussion forums. For the B2G project, the following IRC servers and the respective channels are helpful

1. Freenode(irc.freenode.org):
   (a) #android-root: For getting help on android
   (b) #cyanogenmod-dev: For getting help on developing cyanogenmod
2. Mozilla(irc.mozilla.org):
   (a) #b2g: Help on B2G development
   (b) #jsapi: Help on java script engine
   (c) #mobile: For mobile based help. Help on fennec, firefox for android can be found here

Mibbit is a famous IRC client. For freenode, use webchat
XChat is a good standalone IRC client.

7.3.2 Mailing list

The mailing list can be used for more non interactive help. People post questions and experts answer them. Some of the mailing list:

1. dev-b2g-request@list.mozilla.org

7.3.3 XDA forum

The XDA forum is a community driven forum which focuses on android development mainly. Information about android mobiles, their kernels, and android development can be obtained here. Website: XDA forum

7.4 Links to the code repository

The source code of the entire process was distributed among the three members working on it, nevertheless, it can all be got from github.com.

1. B2G main directory - https://github.com/vishwa91/B2G.git. After obtaining this repository, performing the following command should be sufficient to configure for the ideos:

   ```
   ./config.sh ideos
   ```

2. B2G manifest (this link is already present in B2G/config.sh) - https://github.com/praveenv253/b2g-manifest.git
3. B2G build directory (this link is present in the manifest) - https://github.com/vivekkuamrbagaria/platform_build.git
4. Device directory for the Ideos (present in manifest) - https://github.com/vivekkuamrbagaria/android_device_huawei_u
5. Webrtc for Ideos (present in manifest) - https://github.com/vivekkuamrbagaria/webrtc.git
6. Documentation for this project (including editable formats of these files) - https://github.com/praveenv253/b2g-docs.git