Genode on Secure Tablet

A project report submitted by

N Kranthi Tej
EE13B037

in partial fulfilment of the requirements
for the award of the degree

BACHELOR OF TECHNOLOGY

Department of Electrical Engineering
Indian Institute of Technology Madras

May 19, 2017

Thesis Certificate

This is to certify that the thesis titled **Genode on Secure Tablet**, submitted by **N Kranthi Tej**, to the Indian Institute of Technology, Madras, for the award of the degree of Bachelor of Technology, is a bonafide record of the project work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. V. Kamakoti
Project Guide
Professor
Dept. of Computer Science & Engineering
IIT Madras, Chennai - 600036

Place: Chennai
Date: 09 May, 2017

Acknowledgements

The work that I've done on this project was done under the guidance of Dr. V. Kamakoti. I would like to thank Dr. V. Kamakoti for all the motivation and guidance he has provided throughout the span of this project.

I would also like to thank Mr. Vasan for his guidance throughout the course of this project. He had provided me with all the support I required without any delay and had helped me out solve the problems I've encountered in this project. I wouldn't have made this much progress without his guidance.

I would also like to thank my parents for encouraging me throughout this project. Their understanding of how I had to dedicate my time for this project and that I had only very few occasions to return home has really been a driving factor for me to progress further.

Lastly, I would like to thank the Ministry of Human Resource and Development, Government of India and Defence Research and Development Organization for the financial aid and equipment provided.

Kranthi Tej

Abstract

Key words: Genode, Image, Flashing, Booting, Touchscreen

This report contains the implementation of Genode on the secure tablet which is a joint project between Indian Institute of Technology Madras and Defence Research and Development Organization. It is mostly based on how to bring up Genode on a **proprietory tablet designed by RISE Lab of IIT Madras**. This document contains extensive details about the kind of errors one could face in each stage of bringing up Genode on this tablet - building the Genode image, flashing it onto the tablet and booting it up. The fixes have also been provided for most of the problems encountered. Along with the resolved issues, this document also contains different solutions that have been tested and their outcomes. Further areas of investigation have also been indicated in the document. The objective of this document is to provide a head start for the person who wishes to work on this tablet. The steps mentioned in this document will reduce time consumption on issues which have already been worked on and will provide the person to build further on this foundation, thus contributing more to this new field of microkernels.

This document also contains details about the touchscreen component of the tablet. Currently, there is no driver for the touchscreen on the Genode kernel (for i.MX6 processor). However, the framebuffer support for i.MX53 (which is the previous version of ARM processor to i.MX6 by Freescale/NXP) has been implemented and incorporated into the Genode code base. The driver can be built on this foundation. A brief about the *framebuffer*, *I2C* interface and Capacitive Touch Panel controller (*FT5x06*) has been provided. The driver can be developed by taking insights from the Linux kernel implementation of the same. This document provides a mapping between the interfaces used on the Linux kernel side and Genode side with respect to touchscreen driver implementation.

Contents

1 Introduction

1.1 Motivation

Information and device safety has been a matter of increasing concern over the past decade. Hackers try to exploit the vulnerabilities in the software to extract confidential data from electronic devices. The number of vulnerabilities in the software shoots up when the code base becomes larger and complicated. Therefore, it is very essential to have a simple code base which does not compromise on the functionalities. Linux kernel is one case of a huge code base. The probability of finding new vulnerabilities (bugs) in the code becomes higher because of its large source code. This is the case with most of the Monolithic kernels.

Monolithic kernels have the entire functionality of the Operating System in the kernel itself. This makes it complicated and the amount of code running in the privileged mode is very large. As a solution to this problem, Microkernels came into existence. Microkernels only include the parts of code which should be necessarily running in privileged mode (kernel space). All the others parts are placed in the user space. This resulted in the significant reduction of the size of the code base.

Genode is one such framework based on Microkernels. Our aim is to make the tablet (designed by RISE Lab of IIT Madras) secure with as little room for vulnerabilities as possible. Currently, most of the devices around the world are running Android Operating System. This has been developed with Linux kernel as the foundation. Our aim is to replace the Linux kernel with a microkernel as the foundation for Android, thus achieving more security.

1.2 Objective

The objective of this project is to bring up Genode on a **proprietary tablet designed by RISE Lab of IIT Madras**. It has been developed into a finished tablet. The tablet is meant to serve as an end product for the consumer. From a security point of view, having a SD card support for the device is a vulnerability. Therefore, the SD card support for this tablet has been removed.

Due to absence of the support of SD card, the interfaces utilized to success-

fully bring up Genode on the tablet are quite different than the ones used on development boards. *Bringing up Genode on this tablet without the support of SD card is one of the prime objectives of this project.*

Keeping this objective in mind, the ***eMMC Flash*** available on the tablet has been utilized to flash the Genode image onto it. It has been successfully implemented and all the issues that were encountered in the process have also been mentioned along with their fixes. The scope of this project covers how to build the suitable Genode image for our device, flashing the image onto it (without the help of an external component) and successfully booting it up.

This objective has been achieved in this project. The custom device now runs Genode successfully on it.

1.3 Organization of this Document

This document is organized into 7 parts:

1. **Introduction:** Motivation for this project
2. **Build Genode Image:** This part describes the procedure to build a Genode uImage. It also outlines the problems that were encountered in building the uImage and how they were resolved
3. **Flashing the uImage:** This parts details the procedure to be followed for flashing the Genode Operating System onto the tablet. MfgTool is used for flashing. A brief on how to use the tool and how to resolve the issues which may arise while flashing are described
4. **Booting up Genode:** This part mainly deals with setting up the Uboot environment to boot Genode. The problems faced while booting up are also described in detail. The different approaches tried to resolve these problems have also been described in detail
5. **Touchscreen Overview:** This part talks about the touchscreen of the tablet
6. **Mapping between Linux and Genode - Touchscreen drivers:** This part outlines the common features between touchscreen driver implemen-

tation in the Linux kernel and Genode and provides a mapping between them

7. **Future Scope:** Potential for future work in this area

2 Building the Image for Tablet

2.1 Generation of uImage

We require a uImage to get the Genode kernel running on the tablet. A uImage is a compressed version of the kernel image along with uboot wrapper (OS type and loader information). We use this uImage and run it on the uboot (bootloader) to start the kernel on the tablet. In order to generate the uImage of Genode, several steps have to be followed. The remaining part of this section describes the build procedure.

The build procedure has been described in detail here. It first involves setting up the Genode build environment. Once this is done, one can get started with the build procedure.

2.2 Setting up the Build environment

- Download the latest Genode source code from Genode's git repository [1]. Use the "Download Zip" option. Uncompress the zip file. Let the obtained Genode directory be *<genode-dir>*
- Download the Genode toolchain under the name "*genode-toolchain-<version>-<arch>.tar.bz2*" from [2]
- Open the terminal and navigate to the directory where the toolchain has been downloaded. Run the following line:

```
sudo tar xPfj genode-toolchain-<version>-<arch>.tar.bz2
/* Example: sudo tar xPfj genode-toolchain-16.05-x86_64.tar.bz2 */
```

- Check your GNU Make version by running "make –version" on the terminal. Ensure that it is 3.81 (or higher)
- Install the following packages on your system

```
u-boot-tools, libSDL-dev, tclsh, expect, qemu,
genisoimage, byacc, autoconf2.64, autogen,
bison, flex, g++, git, gperf, libxml2-utils,
subversion, xsltproc
```

- You can install the packages in the following way on the terminal:

```
sudo apt-get install <package-name>
```

With having completed these steps, the build environment for building Genode is set up. One can now move onto the next section of building the uImage.

2.3 Building the uImage

Note: *<genode-dir>* refers to the path of the Genode directory obtained after uncompressing the file downloaded from [1] as it has been mentioned above. This notation will be used subsequently.

- Open the terminal and run the following:

```
<genode-dir>/tool/create_builddir <hardware-platform>
<hardware-platform> can be the following (Supported by
Genode Community):
  • wand_quad (i.MX6 processor)
  • x86_64
  • pbxa9
  • riscv
  • rpi
  • zynq
  • panda (Pandaboard)
  • usb_armory
  • odroid_xu
  • imx53_qsb
  • arndale
    For the subsequent steps, wand_quad will be used as
    an example.
```

- The above command would have created a build directory at *<genode-dir>/build/wand_quad*. Now, make the following changes in *<genode-dir>/build/wand_quad/etc/build.conf* file:

- Uncomment line 6 or the corresponding code for enabling parallel build (remove the # before MAKE += -j4)
- Add the following line:

```
RUN_OPT += -include image/uboot
```

- Since, there is no direct support for our customized tablet, the Wandboard configuration settings (which uses an i.MX6 processor) are utilized. However, some adjustments need to be made to make it compatible with the device at hand. The tablet has 1 GB RAM (whereas, Wandboard has 2 GB RAM). To change the RAM size, make the following changes in the *<genode-dir>/repos/base/include/spec/imx6/drivers/board_base.h* file:

  - Change *RAM0_SIZE = 0x80000000,* to *RAM0_SIZE = 0x40000000,* in line 33

- We currently have support of upto 2 cores in the device. So, make the following changes in *<genode-dir>/repos/base-hw/lib/mk/spec/imx6/*.mk* files:

  - Change *NR_OF_CPUS = 4* to *NR_OF_CPUS = 1* (or, *NR_OF_CPUS = 2*)

- One is all set to build the uImage now. Proceed in the following manner to build the uImage:
  - Open the terminal and navigate to *<genode-dir>/build/wand_quad* directory using:

    ```
    cd <genode-dir>/build/wand_quad
    ```

  - Run the following command:

    ```
    make cleanall
    ```

  - Now, run the following command:

    ```
    make run/log
    ```

Note: "log.run" runscript has been used here. One could as well try other runscripts like run/demo etc. located in <genode-dir>/repos/os/run folder. The following is a brief about 2 of the runscripts:

- **Log runscript**
  This is the most basic runscript available in the Genode code base. It only has the essential components for the Operating system to get running. It builds the core and init components which are the most necessary components. After building these components, it prints a set of messages indicating that the build was successful. This runscript can be found under *<genode-dir>/repos/base/run/log.run*
- **Demo runscript**
  Demo runscript is a slightly more complex runscript than the log runscript. It builds the timer, nitpicker, pointer, status bar, xray trigger, rom filter, report rom modules along with core and init. When run on a suitable hardware, this runscript generates a user interface (UI) of the launchpad which has interactive buttons. This runscript can be found under *<genode-dir>/repos/os/run/demo.run*

More information about the how a runscript functions can be found in Section 2.5.4 of [11]

- After following the above steps, the uImage can be found in the following folder:

  `<genode-dir>/build/wand_quad/var/run/log`

2.4 Resolved Build issues

1. **Build not successful in Genode - 15.02 (IIT Madras repository [3]):**
   If the Genode - 15 repository is used and a uImage is built with it, it may pop up a compilation error. The build error will look like the following:

```
spawn make core init drivers/timer server/nitpicker app/pointer app/status_bar server/liquid_framebuffer app/launchpad app/scout test/nitpicker server/nitlo
amebuffer drivers/pci drivers/input server/report_rom
make[1]: Entering directory '/home/kranthitej/Downloads/genode-hw_sabrelite_tz_support/build/hw_wand_quad'
checking library dependencies...
Skip target drivers/framebuffer/exynos5 because it requires exynos5
Skip target drivers/framebuffer/imx53 because it requires imx53
Skip target drivers/framebuffer/omap4 because it requires omap4
Skip target drivers/framebuffer/pl11x/pbxa9 because it requires pl11x platform_pbxa9
Skip target drivers/framebuffer/pl11x/vea9x4 because it requires pl11x platform_vea9x4
Skip target drivers/framebuffer/pl11x/vpb926 because it requires pl11x platform_vpb926
Skip target drivers/framebuffer/rpi because it requires platform_rpi
Skip target drivers/framebuffer/sdl because it requires linux sdl
Skip target drivers/input/imx53 because it requires imx53
Skip target drivers/input/ps2/pl050 because it requires pl050
Skip target drivers/input/ps2/x86 because it requires x86 ps2
Skip target drivers/pci/device_pd because it requires nova
Skip target drivers/pci/x86 because it requires x86
  Library platform
    MERGE    platform.lib.a
  Library cxx
    COMPILE  exception.o
    COMPILE  guard.o
    COMPILE  malloc_free.o
In file included from /home/kranthitej/Downloads/genode-hw_sabrelite_tz_support/repos/base/include/base/capability.h:21:0,
                 from /home/kranthitej/Downloads/genode-hw_sabrelite_tz_support/repos/base/include/parent/capability.h:17,
                 from /home/kranthitej/Downloads/genode-hw_sabrelite_tz_support/repos/base/include/base/env.h:17,
                 from /home/kranthitej/Downloads/genode-hw_sabrelite_tz_support/repos/base/src/base/cxx/malloc_free.cc:17:
/home/kranthitej/Downloads/genode-hw_sabrelite_tz_support/repos/base-hw/include/base/native_types.h:320:41: error: reinterpret_cast from integer to pointer
    & ~((1 << MIN_MAPPING_SIZE_LOG2) - 1));
                                        ^
/home/kranthitej/Downloads/genode-hw_sabrelite_tz_support/repos/base/mk/generic.mk:56: recipe for target 'malloc_free.o' failed
make[3]: *** [malloc_free.o] Error 1
var/libdeps:77: recipe for target 'cxx.lib' failed
make[2]: *** [cxx.lib] Error 2
Makefile:209: recipe for target 'gen_deps_and_build_targets' failed
make[1]: *** [gen_deps_and_build_targets] Error 2
make[1]: Leaving directory '/home/kranthitej/Downloads/genode-hw_sabrelite_tz_support/build/hw_wand_quad'
Error: Genode build failed
Makefile:231: recipe for target 'run/demo' failed
make: *** [run/demo] Error 252
```

Fix:
In case of such errors make the following change in *<genode-dir>/repos/base-hw/include/base/native_types.h* file:
Remove *constexpr* from *line 318* (or, the line which is 2 lines prior to the line mentioned in the error)

Now, make a clean (*make cleanall*) and make a fresh build. The compilation error will not be encountered.

2. **Build error while trying to generate *.img* disk image:**
When one tries to build a *".img"* format image (*RUN_OPT += –include*

image/disk), an error regarding missing directory (*/usr/local/genode-rump*) might be encountered. The *rump* toolchain has to be installed to build a *.img* file.

Fix:
To install the rump toolchain, run the following commands in the terminal:
- *cd <genode-dir>/tool*
- *./tool_chain_rump build*
- *./tool_chain_rump install*

These commands will set up the *rump* toolchain on the system and compilation error will be resolved. *Please note that one may have to look into some relevant ".mk" files and comment out the problem causing lines if the toolchain is not getting built properly.*

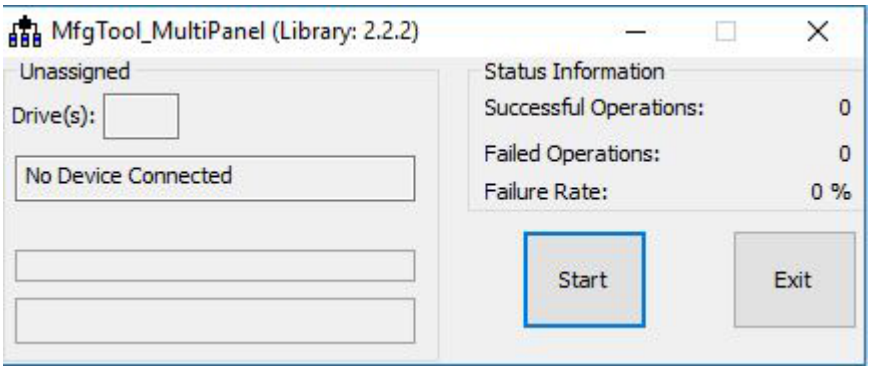
3 Flashing the uImage

3.1 Flashing Tool & Procedure

MfgTool

With the uImage in hand (as mentioned in the previous part), one can go ahead with flashing it onto the device. Flashing the uImage onto the device essentially means that the uImage will be stored onto the device's ROM and this part will be loaded by the bootloader (uboot in our case) everytime we bootup the device. The eMMC Flash available on the device is utilized for this purpose.

For flashing the uImage using the eMMC flash, MfgTool with *MX6Q Linux Update* is required. It is a tool which runs on the Windows Operating System. The following is a screenshot of how the tool looks:



The MfgTool is generally used to flash Android onto the tablets. The tool has to be tweaked at some parts to make it compatible with the Operating System that is being flashed (Genode). The following are the steps to be followed for setting up the MfgTool:

- Download the MfgTool with *MX6Q Linux Update* from the internet [15] (or, it is available in the systems in Network Systems Lab, Department of Computer Science and Engineering, IIT Madras under the name *Indus2012*)

- Let the downloaded MfgTool directory be *<MfgTool-folder>*
- Remove all (except *ucl2.xml*) the *XML* files (**.xml extension*) in *<MfgTool-folder>/Profiles/MX6Q Linux Update/OS Firmware* folder

Now, changes have to be made in the *ucl2.xml* file to make it compatible with the OS being flashed. The code which has to replace all the content present in *ucl2.xml* file has been presented first. The following is the code:

```
<UCL>
<CFG>
    <STATE name="BootStrap" dev="MX6Q" vid="15A2" pid="0054"/>
<STATE name="Updater"   dev="MSC" vid="066F" pid="37FF"/>
   </CFG>
<LIST name="Android-SabreSD-eMMC" desc="Choose eMMC android as media">
<CMD state="BootStrap" type="boot" body="BootStrap" file ="u-boot-mx6q-sabresd.bin" >
Loading U-boot</CMD>
<CMD state="BootStrap" type="load" file="uImage" address="0x10800000"
         loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" >
Loading Kernel.</CMD>
   <CMD state="BootStrap" type="load" file="initramfs.cpio.gz.uboot"
 address="0x10C00000"         loadSection="OTH" setSection="OTH"
 HasFlashHeader="FALSE" >Loading Initramfs.</CMD>
    <CMD state="BootStrap" type="jump" > Jumping to OS image. </CMD>
<CMD state="Updater" type="push" body="$ dd if=/dev/zero of=/dev/mmcblk0
 bs=512 seek=1536 count=16">clean up u-boot parameter</CMD>
<CMD state="Updater" type="push" body="$ echo 0 > /sys/block/mmcblk0boot0/force_ro">
access boot partition 1</CMD>
<CMD state="Updater" type="push" body="send" file="files/android/u-boot-6q.bin">
Sending U-Boot</CMD>
<CMD state="Updater" type="push" body="$ dd if=$FILE of=/dev/mmcblk0boot0
 bs=1k seek=1 skip=1 conv=fsync">write U-Boot to sd card</CMD>
<CMD state="Updater" type="push" body="$ echo 8 >
 /sys/devices/platform/sdhci-esdhc-imx.3/mmc_host/mmc0/mmc0:0001/boot_config">
access user partition and enable boot partion 1 to boot</CMD>
<CMD state="Updater" type="push" body="send" file="mksdcard-android.sh.tar">
Sending partition shell</CMD>
<CMD state="Updater" type="push" body="$ tar xf $FILE "> Partitioning...</CMD>
<CMD state="Updater" type="push" body="$ sh mksdcard-android.sh /dev/mmcblk0">
 Partitioning...</CMD>
<CMD state="Updater" type="push" body="$ ls -l /dev/mmc* ">
Formatting sd partition</CMD>
<CMD state="Updater" type="push" body="$ mkfs.ext2 /dev/mmcblk0p1">
```

```
Creating ext2 file system</CMD>
<CMD state="Updater" type="push" body="$ mkdir -p /mnt/mmcblk0p1"/>
<CMD state="Updater" type="push" body="$ mount -t ext2 /dev/mmcblk0p1
 /mnt/mmcblk0p1"/>
<CMD state="Updater" type="push" body="pipe tar -jxv -C /mnt/mmcblk0p1"
 file="files/uImage.tar.bz2">Sending and writing rootfs</CMD>
<CMD state="Updater" type="push" body="frf">Finishing rootfs write</CMD>
<CMD state="Updater" type="push" body="$ umount /mnt/mmcblk0p1">
Unmounting rootfs partition</CMD>
<CMD state="Updater" type="push" body="$ echo Update Complete!">Done</CMD>
</LIST>
</UCL>
```

Explanation: The flashing procedure is described by the above XML script. It is divided into 2 parts:

- Loading the Linux kernel (for operations)
- Flashing the Genode kernel uImage

  Loading the Linux kernel: The first part of the XML script involves loading the Linux kernel uImage via the uboot (bootloader). This is essential because the device that is at hand is purely hardware and there is no software built on it. Some basic functionalities (like the file system and partitioning commands) are required to flash the uImage successfully. The *u-boot-mx6q-sabresd.bin* file is the Linux kernel image. *initramfs.cpio.gz.uboot* file has the filesystem functionalities. This part of the flashing procedure plays an important part.

  Flashing the Genode kernel uImage: This is the part of the flashing procedure where the uImage is flashed onto the device's ROM. It is done by first partitioning the device to create a boot partition. Then, an *ext2* file system is created on the boot partition. Once the file system has been created, the uImage is placed in this partition. It is from this partition that the kernel is loaded on boot up.

Note: All the content in the *ucl2.xml* file has to be replaced with the XML code given above.

3.2 Flashing procedure

One can now move ahead with the flashing procedure.

Proceed with the following steps ***paying close attention*** (one mistake can lead to unsuccessful flashing) in a ***serial order***:

- Compress the uImage (built in *<genode-dir>/build/wand_quad/var/run/log* folder) to *uImage.tar.bz2* using a compression tool. (Note: compressing it to .tar.bz2 format is necessary. Otherwise, flashing will not be successful. Also, the compressed file name has to be *uImage.tar.bz2*)
- Place the compressed file in *<MfgTool-folder>/Profiles/MX6Q Linux Update/OS Firmware/files* folder
- Connect the tablet to the system running MfgTool with a USB cable
- Switch on the tablet by pressing the *Volume Down + Power buttons* (*Volume down* first immediately followed by *Power Button*)
- Now, start the *MfgTool2.exe* in the *<MfgTool-folder>*. "HID-compliant device" should be seen (if "HID-compliant device" is not seen, switch off the tablet and switch it on as mentioned above)
- Hit the "start" button. The image starts flashing onto the device
- Hit "stop" when green coloured bars and a message saying "Done" appear on the tool
- Switch off the device (Long press *power button*)

3.3 Resolved flashing problems:

1. **Image size is greater than 8 MB:**
   The image generated from the build process of Genode can sometimes be over 8 MB. In such cases, flashing will not be successful because the boot partition can only hold 8 MB of data. This is the usually the case when one wants to use **.img* format for the image (because the Android counterpart of flashing uses *.img* format). *".img"* format Genode image is generated when the following is included
   RUN_OPT += –include image/disk (instead of *RUN_OPT += –include image/uboot*)

in the *<genode-dir>/build/wand_quad/etc/build.conf* file.

Fix:
In such cases, the partitioning has to be changed. *sfdisk* has to be used in the XML script to adjust the partitions to include the image. Look into the sfdisk man page (*man sfdisk*) for more details.

2. **Wrong image format for bootm:**
This is the common error which is encountered when the device is booted after flashing. The most probable cause for this problem is that the flashing was unsuccessful. It can be due to the non-existence of a filesystem on the boot partition. Because of the absence of a valid filesystem, the placing of uImage into the ROM will not be successful. Hence, the boot error.

Fix:
Create a filesystem in the boot partition before flashing the uImage onto it. An *ext2* filesystem has been created in the boot partition before the actual flashing was done. This modification has been incorporated in the *XML* script given above.

3. **Can't get kernel image:**
This message is accompanied by the error stated in the previous point. If the above solution doesn't fix the problem, then the possible cause for the problem is the corruption of uImage (either while generation *or* in the transition from a Linux system to Windows system).

Fix:
Generate the uImage freshly after making a clean (*make cleanall*). Copy the uImage into a USB stick with *NTFS* file system from the Linux system. Use this USB stick to paste the uImage onto the Windows system.

4 Booting up Genode

4.1 Boot settings & Issues

Setting Uboot environment & Booting

One is now equipped with a tablet which can boot Genode. The boot partition has been set on the device and Genode image has been flashed onto it. But the bootloader has to identify the correct location to load the uImage. For this, few command have to be passed in the uboot prompt. Before moving to that part, a brief of uboot has been provided.

Uboot

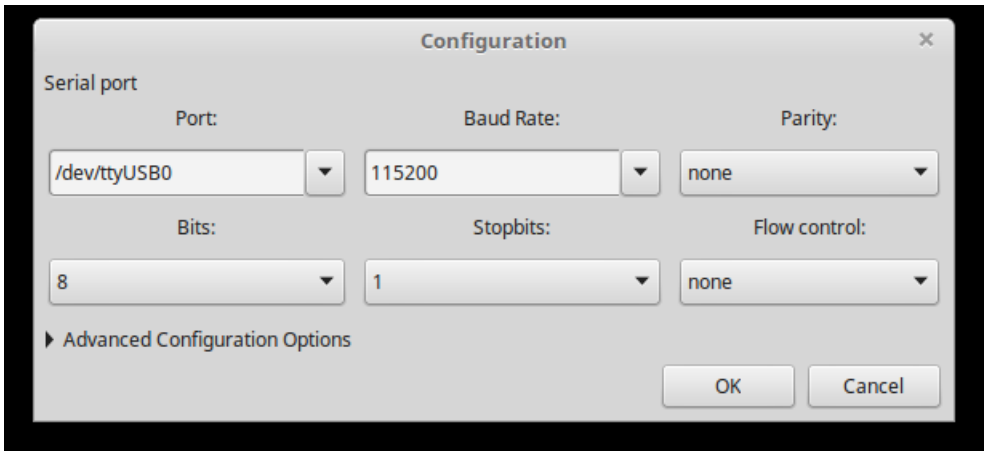
Uboot is a bootloader which is the first component to get loaded from the ROM. Uboot acts as a framework to load the kernel and the operating system on the device. There is a hierarchy: Operating system works on the foundation of the kernel. The kernel itself is loaded from the uboot. Uboot becomes an essential component in booting up the Genode OS on the device.

Now, one can move on to booting up the device.

Booting

The following are the steps to successfully boot up Genode on the tablet:

- Before turning on the device, connect it to a system via serial port. Make the connections according to the following:
  - Black pin (Ground or GND)
  - Grey pin (Receive or RX)
  - Yellow pin (Transmit or TX)
- Configure gtkTerm or TeraTerm on the system to the following settings:



This can be done by opening gtkterm via terminal (*sudo gtkterm*) and then *Configuration->Port*

- Now, switch on the device (Press *power button*)
- In the gtkterm, stop the autoboot by pressing any key (in case autoboot isn't halted, reboot the device and try again. If the problem still persists, make sure that the TX/RX connections are properly in place)
- Once autoboot is halted, the uboot prompt is presented. Execute the following commands in the uboot prompt to run the uImage:

```
- ext2load mmc 3:1 0x30000000 uImage
- bootm 0x30000000
```

The above steps will get the kernel image running and it gets initialized. Genode will be successfully running on the tablet once the above procedure has been completed.

4.2 Boot up Issues

This section of the document deals with the possible problems at boot up. The fixes for these problems have also been provided. However, there might be some problems which may not be completely fixed by the solutions provided.

Hence, a list of solutions that have been tested (and their results) have also been provided. It could help get some insight about the flow of things in the Genode boot up process.

1. **System hangs after *"Starting kernel ..."*:**
   When the device is booted up as per the procedure mentioned above, the system hanging after "Starting kernel ..." message. The log will look something like this:

   MX6Q SABRESD U-Boot > bootm 0x30000000
   ## Booting kernel from Legacy Image at 30000000 ...
   Image Name: Image Type: ARM Linux Kernel Image (gzip compressed)
   Data Size: 608176 Bytes = 593.9 kB
   Load Address: 10001000
   Entry Point: 10001000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK

   Starting kernel ...

   The reason for this problem can be due to 4 reasons:

   (a) UART port settings are not correctly set
   (b) Setting number of cores to a value more than physically available
   (c) *SMP* variable might have been set to *false*
   (d) "Log" message before in Bootstrap code

   UART port settings are not correctly set:
   The proprietory tablet device's pins correspond to the *UART1* port which has a *base address of 0x02020000* and an *interrupt value of 58.* These have to be correctly defined in the *<genode-dir>/repos/base/include/spec/imx6/drivers/board_base.h* file. If these values are incorrectly defined, such a problem might arise (where no further output is observed).

   However, if *Sabrelite development board* is being used, the pins correspond to the *UART2* port. In such a case, the *UART base address*

should be *0x021e8000* and the *interrupt value* should be *59*.

Setting number of cores to a value more than physically available:
When the *NR_OF_CPUS* value is set to a value greater than 2 in *<genode-dir>/repos/base-hw/lib/mk/spec/imx6/*.mk* files, this issue may arise. Make sure that the value of *NR_OF_CPUS = 1 or NR_OF_CPUS = 2* in both the files.

***SMP* variable might have been set to *false*:**
The system can hang at *"Starting kernel ..."* message if the *SMP* variable has been set to false in *<genode-dir>/repos/base-hw/src/core/include/spec/cortex_a9/board_support.h* file (line 29). Make sure that this variable has been set to ***true***.

"Log" message before in Bootstrap code:
There might not be any ouput if there is a log message (*Genode::log("")*) in any part of the code in the *<genode-dir>/repos/base-hw/src/bootstrap/init.cc* file. Check this parameter if the above two fixes are already in place.

Note: Once the Memory Management Unit (MMU) has been enabled, the *LOG* service cannot be utilized until core is started. This is because there is a mismatch between virtual address and physical address when *LOG* service is used. Once the core has started, the mapping between virtual address and physical address has been established. So, the *LOG* service after the core has started. In the light of such event, the *Genode::error("")* function (discussed later) can be utilized.

2. **Error: page fault in core thread (core):**
   While booting up the device, one may encounter a page fault in the core thread. The error may look like the following:

MX6Q SABRESD U-Boot > bootm 0x30000000
Booting kernel from Legacy Image at 30000000 ...
Image Name:
Image Type: ARM Linux Kernel Image (gzip compressed)
Data Size: 608176 Bytes = 593.9 kB
Load Address: 10001000
Entry Point: 10001000
Verifying Checksum ... OK
Uncompressing Kernel Image ... OK

Starting kernel ...

:virt_alloc: Allocator 0x200c40b8 dump:
Block: [00001000,10001000) size=256M avail=256M max_avail=256M
Block: [10585000,20001000) size=256496K avail=256496K max_avail=3144208K
Block: [2017b000,2017c000) size=4K avail=0 max_avail=0
Block: [2017c000,e0000000) size=3144208K avail=3144208K max_avail=3144208K
Block: [f0004000,f0005000) size=4K avail=0 max_avail=3144208K
Block: [f0007000,f0008000) size=4K avail=0 max_avail=0
Block: [f0009000,f000a000) size=4K avail=0 max_avail=262036K
Block: [f000a000,fffef000) size=262036K avail=262036K max_avail=262036K
=> mem_size=4019097600 (3832 MB) / mem_avail=4019081216 (3832 MB)

:phys_alloc: Allocator 0x200c304c dump:
Block: [10585000,10586000) size=4K avail=0 max_avail=0
Block: [10586000,10587000) size=4K avail=0 max_avail=1042644K
Block: [10587000,10588000) size=4K avail=0 max_avail=0
Block: [105ca000,105cb000) size=4K avail=0 max_avail=1042644K
Block: [105cb000,50000000) size=1042644K avail=1042644K max_avail=1042644K
=> mem_size=1067683840 (1018 MB) / mem_avail=1067667456 (1018 MB)

:io_mem_alloc: Allocator 0x200c5130 dump:
Block: [00000000,10585000) size=267796K avail=267796K max_avail=267796K
Block: [10588000,105ca000) size=264K avail=264K max_avail=2952790015

Block: [50000000,ffffffff) size=2952790015 avail=2952790015 max_avail=2952790015
=> mem_size=3227283455 (3077 MB) / mem_avail=3227283455 (3077
MB)

:io_port_alloc: Allocator 0x200c619c dump:
=> mem_size=0 (0 MB) / mem_avail=0 (0 MB)

:irq_alloc: Allocator 0x200c7208 dump:
Block: [00000000,00000001) size=1 avail=1 max_avail=1
Block: [00000002,0000001d) size=27 avail=27 max_avail=994
Block: [0000001e,00000400) size=994 avail=994 max_avail=994
=> mem_size=1022 (0 MB) / mem_avail=1022 (0 MB)

:rom_fs: ROM modules:
ROM: [1017e000,1017e158) config
ROM: [10154000,1017a900) init
ROM: [100d6000,10153b64) ld.lib.so
ROM: [1017b000,1017d598) test-log

kernel initialized
Error: page fault in core thread (core): ip=0x20037b34 fault=0x68c88038

Fix:
In such cases, the problem could be with the device. Try changing the device. This should resolve the problem. The cause for this problem is not clear yet. It seems to be working with some devices and it shows up a pagefault in some others. It could be an issue with the way the devices have been manufactured or it could be an issue with the memory of the device.

4.3 Tested Solutions & their outcomes

From now on, a tablet which successfully runs Genode is referred as "working tablet" and the tablet which gives out a pagefault is referred as "faulty tablet".

1. **Comparing Genode 15, 16 & 17 in case of Pagefault**

   All Genode codebases (15.05, 16.05, 17.02) seem to give out pagefault when a faulty device is used. It can be inferred that, fundamentally, the flow of execution of the code has not much dependency on the version of Genode source code used. Only the organization of the files is different across the different versions of Genode source code.

   From this, it can be inferred that the problem is with the device and not the source code itself.

2. **Case of hard-coded values in Uboot source code**

   Uboot is a bootloader which we are using for for booting up the RISE Lab's tablet. It has a source code of its own. The source code of Uboot has been examined for any hard-coding of addresses which might have caused the pagefault (on devices which gave out a pagefault).

   It was found that there is **NO hard-coding** of any address in the Uboot source code. All the addresses have been initialized by the Genode source code itself. There is no reason to suspect the Uboot source code in any case. One can be almost be sure that the problem is not with Uboot (if any such problem arises in future).

3. **Comparison between addresses in Linux Kernel and Genode**

   Android runs on Linux kernel. Android has been successfully flashed onto the tablet and it runs without any problems. A thorough comparison between the addresses mentioned in the Linux Kernel side and the Genode side has been made. All the relevant addresses (UART, SDHC, EPIT2, AIPS1, AIPS2, CORTEX_A9_PRIVATE_MEM, PL310, SRC) on the Genode side match exactly with the Linux kernel side.

   Hence, one can be sure that there is no problem with the addresses mentioned in the Genode codebase if any problem arises in future (unless, the addresses have been changed in newer versions of Genode).

4. **Usage of *objdump* and its output**

A way to find out the cause for the pagefault is to run *objdump* on the image that has been built by the Genode source code. The steps to be followed to use *objdump* have been mentioned below:

- Once the image is successfully built, open the terminal and run the following commands:

```
cd /usr/local/genode-gcc/bin
./genode-arm-objdump -DCl <genode-dir>/build/wand_quad/var/run/log.core
```

Upon running these commands, the output of *objdump* shows up on the terminal. Stop the execution once the pagefault address (ip mentioned in the output log) is crossed. Search for the *ip* in the *objdump* output. The *objdump* output of the corresponding *ip* will be similar to:

_ZN6Genode17Native_capabilityaSERKS0_():
/home/kranthitej/Desktop/BTP/BTP_Latest/genode-master/repos/base/include/base/native_capability.h:93
200374f4: e7975003 ldr r5, [r7, r3]

This output suggests that the pagefault is occuring due to the above assembly code. ***However, it has been verified with the developers of Genode that there is no problem with the assembly instructions whatsoever.***

It shows that *objdump* output can be misleading sometimes. It must be used with caution. Also, the assembly instructions should not be suspected. If the *objdump* output suggests a problem with the assemble instruction, one can almost be sure that it is not the cause of the problem.

5. **Correctness of *Entrypoint Address***

The *entrypoint address and load address* which are a part of the uboot output have been determined to be 0x10001000 and 0x10001000 respectively. These addresses are correct because they have been verified on the tablet which is running Genode successfully. Moreover, these are the addresses mentioned by most people on the Genode mailing lists.

Based on this, one can be sure that the addresses are the following:
Entrypoint Address : 0x10001000
Load Address : 0x10001000

Please note that the *entrypoint address* can be changed in <genode-dir>/tool/run/boot_dir/hw file in the following manner (in case there is a need to change it for some reason):

```
/* In line 13, change the value that has been mentioned for wand_quad */
if {[have_spec "wand_quad"]}  { return "0x10001000" }
```

6. **Disabling Alignment Check**

   Genode 16.05 (and newer) version has introduced a new bit for alignment check in the System Control Register (*Sctlr*) which is not to be found in the 15.05 or earlier versions. This bit used to be disabled by default in the 15.05 and earlier versions. However, from the 16.05 (and newer versions) this bit has been enabled. The *Sctlr* data structure can be found in *<genode-dir>/repos/base-hw/src/core/include/spec/arm/cpu_support.h* file.

   Genode seems to run successfully with the alignment check enabled as well as disabled on the working tablets. However, the faulty tablets still seem to give out the pagefault (irrespective of enabling or disabling the alignment check bit). For now, it can be said that this bit does not affect the functioning of Genode. However, in future, it is quite possible that this part can cause a problem. In such cases, one can disable the alignment bit by adding the following code snippet at the relevant place:

```
access_t v = read();
/* disable alignment checks */
A::set(v, 0);
```

7. **Issues with *write* function in *Sctlr* structure**

   There might be a stage where one can find a problem with the *write* function under *Sctlr* structure in *<genode-dir>/repos/base-hw/src/core/include/spec/arm/cpu_support.h* file. The *write* function may not seem function as per the expectation. This is the case where *LOG* messages (*Genode::log("")*)

are placed at each step for debugging.

However, it should be noted that there is ***NOTHING WRONG*** with the *write* function. This has been confirmed with the developers at Genode Labs. The issue can be with the *LOG* messages placed in the bootstrap code in *<genode-dir>/repos/base-hw/src/bootstrap/init.cc* file.

Hence, if in future, a similar problem arises, the functions that have been written in the Genode source code should not be suspected. Check for any stray parts of code that may have been placed for the purpose of debugging.

8. **Enabling/Disabling SMP (Symmetric Multiprocessing)**

   Symmetric Multiprocessing (SMP) can either be set to *true* or *false*. This can be done by changing it in line 29 of *<genode-dir>/repos/base-hw/src/core/include/spec/ cortex_a9/board_support.h* file.

   It has been observed that this parameter hasn't had any effect on the faulty tablets (as opposed what has been mentioned in [4]). We still get the pagefault irrespective of what the *SMP* variable has been set to. In the case of working tablets, the system hangs at *"Starting kernel ..."* message (for any value of $NR\_OF\_CPUS > 2$). This parameter can be an area of investigation incase a problem arises in the future.

9. ***M, C, I* combinations in *Sctlr***

   The M, C, I bits of the System Control Register (*Sctlr*) can be set to either 0 or 1 giving rise to 8 possible combinations. These bits can be found under *Sctlr* structure in *<genode-dir>/repos/base-hw/src/core/include/spec/ arm/cpu_support.h* file. The bits stand for the following:

   M - Enabling/Disabling MMU cache
   C - Enabling/Disabling Data cache
   I - Enabling/Disabling Instruction cache

These bits shouldn't be meddled with unless extremely necessary. However, if they have to be modified, the valid combinations are mentioned below with reference to the information given in ARM Infocenter. The bit combinations given below correspond to *MCI* bits respectively (in that order):

000 - valid
001- valid
100 - valid
110 - valid
101 - valid
111 - valid
010 - invalid
011 - invalid

For more information, please refer to [5].

10. **Finding the Memory map initializations in Boot output**

The boot output of a working tablet looks like the following:

MX6Q SABRESD U-Boot > bootm 0x30000000
Booting kernel from Legacy Image at 30000000 ...
Image Name:
Image Type: ARM Linux Kernel Image (gzip compressed)
Data Size: 608730 Bytes = 594.5 kB
Load Address: 10001000
Entry Point: 10001000
Verifying Checksum ... OK
Uncompressing Kernel Image ... OK

Starting kernel ...

:virt_alloc: Allocator 0x200d40b8 dump:
Block: [00001000,10001000) size=256M avail=256M max_avail=256M
Block: [10595000,20001000) size=256432K avail=256432K max_avail=3144144K
Block: [2018b000,2018c000) size=4K avail=0 max_avail=0

Block: [2018c000,e0000000) size=3144144K avail=3144144K max_avail=3144144K
Block: [f0004000,f0005000) size=4K avail=0 max_avail=3144144K
Block: [f0007000,f0008000) size=4K avail=0 max_avail=0
Block: [f0009000,f000a000) size=4K avail=0 max_avail=262036K
Block: [f000a000,ffffef000) size=262036K avail=262036K max_avail=262036K
=> mem_size=4018966528 (3832 MB) / mem_avail=4018950144 (3832 MB)

:phys_alloc: Allocator 0x200d304c dump:
Block: [10595000,10596000) size=4K avail=0 max_avail=0
Block: [10596000,10597000) size=4K avail=0 max_avail=1042516K
Block: [10597000,10598000) size=4K avail=0 max_avail=0 Block: [105ea000,105eb000) size=4K avail=0 max_avail=1042516K
Block: [105eb000,50000000) size=1042516K avail=1042516K max_avail=1042516K
=> mem_size=1067552768 (1018 MB) / mem_avail=1067536384 (1018 MB)

:io_mem_alloc: Allocator 0x200d5130 dump:
Block: [00000000,10595000) size=267860K avail=267860K max_avail=267860K
Block: [10598000,105ea000) size=328K avail=328K max_avail=2952790015
Block: [50000000,ffffffff) size=2952790015 avail=2952790015 max_avail=2952790015
=> mem_size=3227414527 (3077 MB) / mem_avail=3227414527 (3077 MB)

:io_port_alloc: Allocator 0x200d619c dump:
=> mem_size=0 (0 MB) / mem_avail=0 (0 MB)

:irq_alloc: Allocator 0x200d7208 dump:
Block: [00000000,00000001) size=1 avail=1 max_avail=1
Block: [00000002,0000001d) size=27 avail=27 max_avail=994
Block: [0000001e,00000400) size=994 avail=994 max_avail=994
=> mem_size=1022 (0 MB) / mem_avail=1022 (0 MB)

:rom_fs: ROM modules:
ROM: [1018e000,1018e158) config
ROM: [10164000,1018a900) init
ROM: [100e6000,10163b64) ld.lib.so

ROM: [1018b000,1018d598) test-log

kernel initialized
Genode 17.02-127-gf6386c6 <local changes>
1017 MiB RAM assigned to init
[init -> test-log] hex range: [0e00,1680)
[init -> test-log] empty hex range: [0abc0000,0abc0000) (empty!)
[init -> test-log] hex range to limit: [f8,ff]
[init -> test-log] invalid hex range: [f8,08) (overflow!)
[init -> test-log] negative hex char: 0xfe
[init -> test-log] positive hex char: 0x02
[init -> test-log] multiarg string: "parent -> child.7"
[init -> test-log] String(Hex(3)): 0x3
[init -> test-log] Test done.

The address ranges seen in the output have been generated as a part of calculation in the code. However, the start and size of Virtual Memory Allocation (*virt_alloc*) have been identified to be mentioned in *<genode-dir>/repos/base-hw/src/include/base/internal/native_utcb.h* file (lines 34 and 35) under the names - *VIRT_ADDR_SPACE_START & VIRT_ADDR_SPACE_SIZE*. These memory addresses may come in handy in future.

The other address range start and end points could not be identified as they have been calculated on the go. This area may be a good point to start investigating the pagefault that is encountered on faulty tablets because there has been a consistent offset in the address ranges between the working and faulty tablet outputs.

11. **"Error" function as a debugging tool:** An alternative to the *LOG* service is the *Genode::error("")* function. It is similar to the *LOG* service except that, it can also be used even when the core isn't started. It's usage is exactly the same as the *LOG* service. This function may come in handy for debugging. An example of its usage has been given below:

```
/* Print the value of v */
int v = 10;
Genode::error(“Value of v = ”, v);
```

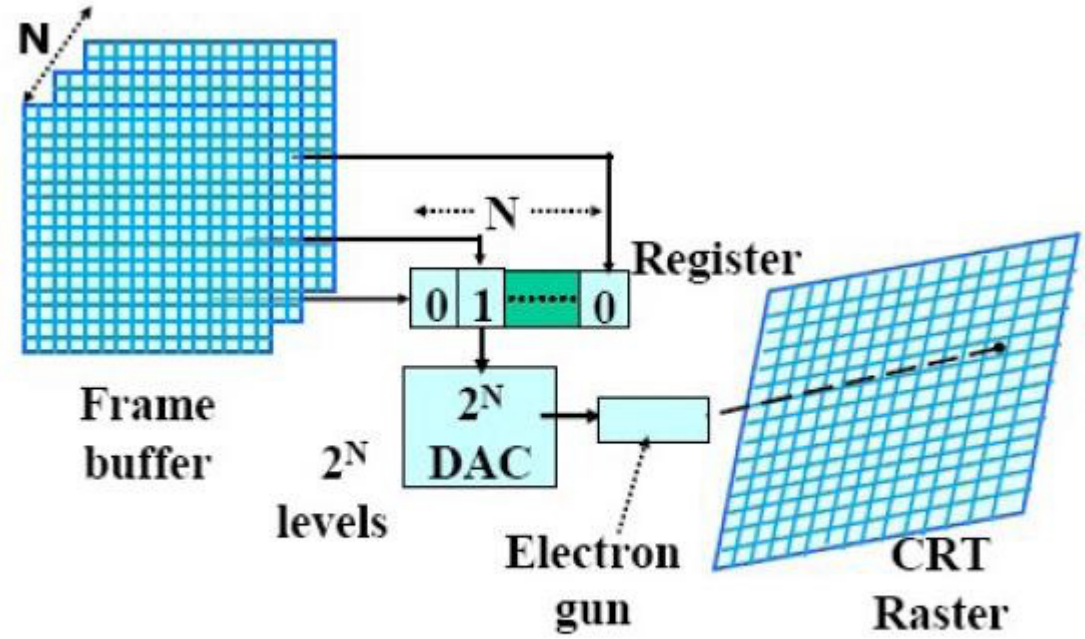
5 Touchscreen Specification

5.1 Overview

The secure tablet is a project undertaken by IIT Madras in collaboration with Defence Research and Development Organization (DRDO). It is designed by RISE Lab of IIT Madras. The tablet has a touchscreen with a 2-point capacitive multi-touch sensor (*Focaltech FT5X06*). The touchscreen drivers for this device have already been implemented for Android in the Linux Kernel. The corresponding driver files can be found in [6] under the names *ft5x06_ts.c* and *ft5x06_ts.h*. The focus of this project is on the development of the touchscreen drivers for the device in Genode.

The touchscreen drivers make use of the Framebuffer as the foundation. Before describing how framebuffer is useful for the display and touchscreen driver implementations, a brief about Framebuffer has been provided below.

Framebuffer is a portion of memory which holds a complete bitmap of the image which is sent to the monitor (or, display device) to be displayed. This part of the memory generally resides in the memory chips on the video adapter. It is sometimes integrated into the motherboard. There exists at least one bit for each pixel in the image. The amount of such memory is called the bit plane. Information is passed from the framebuffer onto a Digital to Analog Converter (DAC). This is passed on to the electron gun which fires the electrons accordingly to light up the screen.



An N- bit plane gray level frame buffer

**picture has been taken from [7]*

For more details on framebuffer, refer to [7].

The framebuffer driver has already been implemented in Genode for i.MX53 processor. The corresponding code has been incorporated into the Genode source code. This can be taken as reference for the implementation of framebuffer for i.MX6 processor as well as the touchscreen driver in Genode. The framebuffer itself is built on the Image Processing Unit (IPU). To learn about the IPU, refer to chapter 37 of [8].

5.2 Capacitive Touch Panel Module (CTPM)

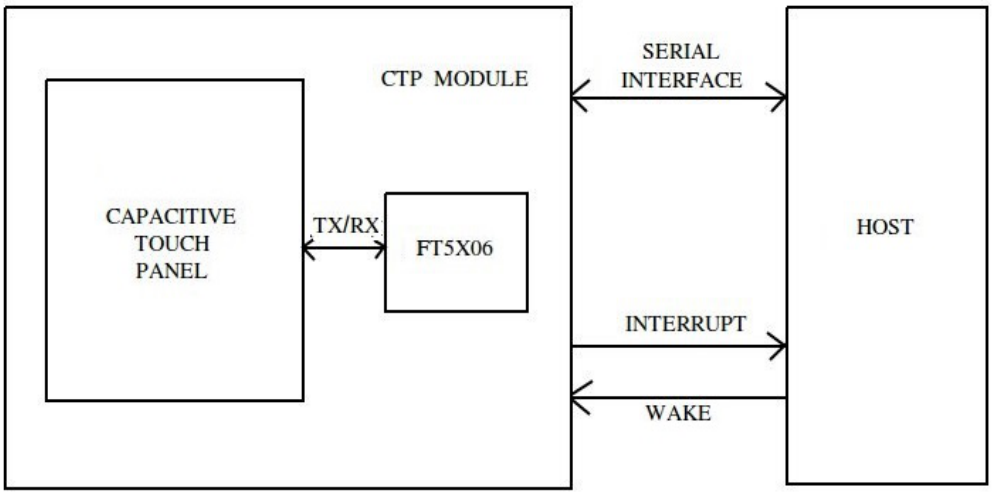
This part has been referred from [9]. This section comprises of two parts:

1. Capacitive Touch Panel (CTP)
2. Capacitive Touch Panel Controller (*FT5x06*)

The Capacitive Touch Panel Module is responsible for all the communication between the host (which is the component calling the display/touch function) and the touchscreen component. Whenever, a touch event is detected on the touch panel (CTP), it is communicated suitably to the controller (*FT5x06*). The *FT5x06* controller module then generates an interrupt and sends it to the host system to communicate the event of touch. This interrupt tells the host that the data is ready for the host to receive. This basically acts as a channel of communication between the Host system and the Controller. The host can also send an interrupt signal to the *FT5x06* controller module in the event of having to wake up the module from hibernation.

The data input and output from the CTPM to the host takes place through a serial interface. It can be *I2C* interface or the SPI interface. In this case, the I2C interface is used. *I2C* interface is a protocol which has been developed by Philips. This utilizes only 2 wires for communicating between the ICs. It is a half duplex serial communication protocol and hence, the data can flow in both the directions. The 2 bidirectional wires used are referred to as SDA (Serial Data) and SCL (Serial Clock). For more information on the *I2C* interface, refer to [10].

The following is the block diagram of the Capacitive Touch Panel Module which describes all the flow of information which has been describe above:



CAPACITIVE TOUCH PANEL MODULE BLOCK DIAGRAM

6 Mapping between Linux and Genode - Touchscreen Drivers

In the previous part, the Capacitive Touch Panel Module and its role of taking the input and passing on the information to the host system in suitable form has been described. In this part, insights are drawn from the touchscreen driver implementation of i.MX53 processor on Genode and suitable suggestions on the necessary interfaces and structures are mentioned to implement the same for i.MX6 processor.

As it has been mentioned in the earlier part, the touchscreen driver for the i.MX6 processor has already been implemented on the Linux kernel. In the further sections of this part, some commonalities are drawn between the Linux kernel implementation and the Genode implementation (for i.MX53 processor). This will give a brief idea on how to organize the code and data for implementation on Genode.

The corresponding Linux kernel driver files can be found in [6] under the names *ft5x06_ts.c* and *ft5x06_ts.h*. All the register addresses and macros have been provided in the header file in [6]. Any touch screen event is captured in a structure. The following data is captured in the structure:

- x coordinate (of the event on the CTP)
- y coordinate (of the event on the CTP)
- Touch event type
- Touch ID
- Pressure
- Touch Point

The touchscreen driver implementation on the Genode side for i.MX53 processor can be found in the following files:
<genode-dir>/repos/os/src/drivers/input/spec/imx53/egalax_ts.h and *<genode-dir>/repos/os/src/drivers/input/spec/imx53/driver.h*
Please note that, these files have implemented the touchscreen driver for a screen which has been manufactured from a different vendor. So, suitable changes have to be made for accounting for the change of screen as well as change of processor. However, there are some similarities between the touchscreen drivers of these 2 processors. Some of the relevant similarities have been identified and a mapping has been provided.

The following mapping has been identified between Linux kernel and Genode:

1. **I2C interface:**

   I2C interface is the basic component which is essential for the communication between the host and the CTP controller (FT5x06). In the linux kernel implementation of the driver, this interface has been used. The implementation of I2C interface already exists in the Genode source code. It can be found in the following file:
   <genode-dir>/repos/os/src/drivers/input/spec/imx53/i2c.h
   Genode implementation of the I2C interface includes *write, send, receive, start* and *stop* functions. The *send* and *receive (recv)* functions are extensively utilized for the serial communication between the host and the CTP controller.

2. **Touchscreen event structure:**

   The touchscreen event (*ts_event*) structure which has been mentioned

above has it's Linux kernel implementation of it in the following file:

kernel-imx/drivers/input/touchscreen/ft5x06.c

It's counterpart on the Genode side of implementation has been found in the following file:

<genode-dir>/repos/os/include/input/event.h

This event implementation has some additional features like the relative x and y coordinates and keycodes. These may not be necessary for the driver implementation because they haven't been used in the Linux side of implementation.

Note: The event type definitions have been defined in the following file on the Linux side:

kernel-imx/include/linux/input.h

The event type values may come in handy while implementing the drivers

3. **Work queue:**

   All the touch events whose data are stored in the structure (mentioned above) are added to a queue before they are processed. This will ensure that all the touch events are captured and are given the appropriate response. As the queue has a FIFO (First In First Out) implementation, all the touch events will be processed in the serial order. The linux kernel implementation uses a structure *workqueue_struct* to implement the queue. It is defined under *ft5x0x_ts_data* structure in the C file in [6].

   In Genode, the event queue has already been implemented in the following file:

   <genode-dir>/repos/os/include/input/event_queue.h

   This implementation provides the basic enqueue (*add*) function, dequeue (*get*) function as well as functions for returning if the queue is empty (*empty*), available capacity (*avail_capacity*) and also for resetting the

queue (*reset*) and submitting a signal (*submit_signal*).

4. **Support Key:**

   In the Linux kernel implementation of the driver (i.e., in the C file of [6]), one can find the parameter - *CFG_SUPPORT_TOUCH_KEY* at several places. For now, it's value has been set to *0* in the header file in [6]. However, in future, if one wants to make use of the *HOME, SEARCH, RETURN* and *touch key* functions on the tablet, this will be set to *1*. (This is essentially if the tablet has buttons along with the touchscreen. The device currently designed by RISE Lab of IIT Madras does not have these buttons).

   It's counterpart in Genode can be found in the following file:

   <genode-dir>/repos/os/include/input/mpr121.h

   This may not be useful immediately and it can be avoided for time being.

5. **Interrupts:**

   All the interrupts which have been used in the C file in [6] have a counterpart in Genode. Genode implements something called as interrupt handler to take care of the interrupts called upon by functions. For the i.MX53 processor, the interrupt handler can be found in the following file:

   <genode-dir>/repos/os/src/drivers/input/spec/imx53/irq_handler.h

   This interrupt handler can be utilized as it is without any changes. All the changes which arise due to the differences between i.MX53 and i.MX6 processors will be in the interrupt numbers beings passed to the functions in the interrupt handler.

   I've identified some interrupt values for the i.MX6 processor taken from the reference manual of i.MX6 [8]. The interrupt values are:

- GPIO1 INT0 interrupt request = 97

6. **Factory mode:**

   When the tablet has to function in the *factory mode,* a function under the name *ft5x0x_enter_factory* has to be called. This function can be found in the C file in [6]. This function uses a *flush_workqueue* function which flushes the existing queue of touchscreen events. This is on the Linux kernel implementation side of it. On Genode, the flush_workqueue's counterpart resides in the following file:

   <genode-dir>/repos/os/include/input/component.h

   This function flushes the entire work queue which has the information of the touchscreen events accumulated. This is an essential step for the tablet to be put in factory mode.

7. **GPIO interface:**

   The Linux kernel implementation uses the GPIO interface to *write* and *read* suitable values to the registers of the device. It's Genode counterpart can be found in the following file:

   <genode-dir>/repos/os/include/gpio/component.h

   This interface has *write, direction and read* functions. These are the same functions used in the C file in [6] for writing to the registers.

Mapping of functions between Linux and Genode

Function Name in Linux	Purpose of Function	Interfaces used	Genode implementation
i2c_transfer	Returns number of messages processed	I2C (Definition)	I2C Interface (mentioned above)
ft5x0x_i2c_rxdata	Returns number of messages received	i2c_transfer	NA
ft5x0x_i2c_txdata	Returns number of messages transmitted	i2c_transfer	NA
ft5x0x_write_reg	Write to register	i2c_transfer	NA
ft5x0x_read_reg	Read from register	i2c_transfer	NA
delay_qt_ms	Delay in milliseconds	-	Can be implemented using *for* loop
i2c_master_recv	Returns number of bytes read	I2C definition	I2C Interface
i2c_read_interface	Read data from CTPM. Returns *true* if successful	I2C	NA
i2c_master_send	Returns number of bytes written to the slave	I2C definition	I2C Interface
i2c_write_interface	Write data to CTPM. Returns *true* if successful	I2C	NA
cmd_write	Sending a command to CTPM	i2c_write_interface	NA
byte_write	Write to CTPM	i2c_write_interface	NA
byte_read	Read out data from CTPM	i2c_read_interface	NA
fts_ctpm_fw_upgrade	Burn the Firmware to CTPM	ft5x0x_write_reg, cmd_write, byte_read, delay_qt_ms, byte_write, ft5x0x_i2c_txdata	NA
i2c_get_clientdata	Fetch client data (in touchscreen event structure)	I2C definition	To be added to I2C Interface
fts_ctpm_auto_clb	Enable CTPM Auto Calibration mode	ft5x0x_write_reg, ft5x0x_read_reg	NA

continued next page.

ft5x0x_read_data	Read Touchsreen event data into the defined structure	ft5x0x_i2c_rxdata, ts_event	NA (touchscreen event structure is implemented - mentioned above)
input_event	Reports new input event to the Host	spin_lock, spin_unlock, input_handle_event	Spinlock and spinunlock have been defined under the names - *spinlock_lock & spinlock_unlock* in [12]
input_report_abs	Report absolute data about input to Host	input_event	NA
input_sync	Report Synchronous input data	input_event	NA
ft5x0x_report_value	Report touch point data	input_report_abs, input_sync	NA (touchscreen event structure is implemented - mentioned above)
enable_irq	Enable interrupts	-	NA - Interrupt Handler implemented (mentioned above)
ft5x0x_ts_pen_irq_work	Pen interrupt	ft5x0x_read_data, ft5x0x_report_value, enable_irq	NA - Interrupt Handler implemented
ft5x0x_ts_interrupt	Send touchscreen interrupt to host	disable_irq_nosync, queue_work	NA - Interrupt Handler, Event queue implemented
ft5x0x_enter_factory	Function for entering factory mode	flush_workqueue, disable_irq_nosync, ft5x0x_write_reg	NA - flush_workqueue implemented in *component.h* (mentioned above)
ft5x0x_enter_work	Function to return to normal mode	ft5x0x_write_reg, ft5x0x_read_reg, enable_irq	NA - Interrupt Handler implemented
ft5x0x_ts_init	Read and write using registers using GPIO interface - See *kernel-imx/drivers/input/touchscreen/ft5x06.c*	gpio_request, gpio_direction_output, gpio_direction_in, put, gpio_set_value, I2C	NA - GPIO interface has been implemented in [13]

*Some irrelevant functions have been omitted. Please refer to the C file in [6].

Differences between i.MX53 and i.MX6 Processors (Image Processing Unit)

The framebuffer for i.MX53 processor has already been implemented in Genode and can be found in the following directory:

<genode-dir>/repos/os/src/drivers/framebuffer/spec/imx53

Since, i.MX53 and i.MX6 processors are very similar, one can make use of the framebuffer of i.MX53 processor. However, some changes have to be made to get it fully functioning. Fundamentally, these two processors differ in the Image Processing Unit (IPU) as far as the framebuffer is concerned. It was found from the reference manuals of both the processors that the Image Processing Unit memory map is different.

The IPU memory map of i.MX6 processor can be found in section 37.5 in [8] while the memory map of i.MX53 processor can be found in section 45.5.1 in [14]. There is a definite change in the memory addresses between both the processors. This has to be accommodated when framebuffer for the i.MX6 is implemented. The other modules are identical between the processors.

The memory addresses for i.MX53 processor on the Genode side can be found in the following file:

<genode-dir>/repos/os/src/drivers/framebuffer/spec/imx53/ipu.h

The above file can be taken as a reference for the implementation of framebuffer for i.MX6 processor. The touchscreen driver can be implemented using the interfaces described above.

7 Future Scope

Genode is now successfully running on the tablet which is a proprietory device designed by RISE Lab of IIT Madras. The next step is to develop drivers to get all the hardware components of the tablet fully functioning. This documents provides the common interfaces and the mapping between Linux kernel implementation and Genode implementation of the touchscreen driver. Other components which also need support are:

- Wifi
- Audio
- GPRS
- USB

to name a few. Once all the devices are fully functional with adequate support, it can be expected to move ahead with the idea of changing the kernel foundation under Android from Linux kernel to microkernel.

Conclusion

Microkernel has complete knowledge of the memory access by components and essentially places a lock on the memory. An attacker can get into the memory through a vulnerability in the code. The microkernel doesn't allow the required memory access in the event of an attack. Thus, a microkernel makes the operating system more secure. Hence, going forward with Genode is a step towards a secure future.

The organization of the code makes it comfortable to integrate code into Genode. This makes it an ideal setup to port Android onto microkernel. The secure tablet (product of IIT Madras) runs Genode successfully. Device drivers for the peripherals will result in a fully functioning secure tablet. This is meant to be an end product for the consumer. The same functionalities can be achieved with enhanced security on this tablet.

Summary

- Microkernels enhance security by restricting access to the memory
- Genode OS Framework allows integration of code while not compromising on security with ease
- Secure tablet hardware specifications have been identified
- Genode Operating System successfully runs on the secure tablet (designed by RISE Lab of IIT Madras)
- Flashing is done with the help of MfgTool
- Uboot is the boot-loader used to bring up the kernel on the device
- Framebuffer can be utilized to develop display and touchscreen drivers for the tablet
- Device drivers can be developed on Genode by drawing insights from the Linux kernel implementation

Appendix

This appendix gives the technical details of the customized Secure tablet. The following are the technical specifications of the device:

- Uses an i.MX6Q processor (Quad core)
- The motherboard is SabreSD with 1 GB of physical RAM
- The pins coming out of the tablet correspond to UART1 port (*Base address = 0x02020000, Interrupt = 58*) and stand for the following:
  - Black - Ground or GND
  - Grey - Receive or RX
  - Yellow - Transmit or TX

  Please refer to the picture below for the location of pins on the tablet.

References

[1] Genode's git repository - https://github.com/genodelabs/genode

[2] Genode toolchain - https://sourceforge.net/projects/genode/files/genode-toolchain/16.05/

[3] IIT Madras' Genode git repository - https://github.com/iitmadras/genode

[4] Praveen Srinivas' query - https://sourceforge.net/p/genode/mailman/message/33648061/

[5] System Control Register - http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0464f/BABJAHDA.html

[6] Linux kernel directory for i.MX6 (Available in Network Systems Lab, Department of Computer Science & Engineering, IIT Madras) - *hki/myandroid/kernel_imx/drivers/input/touchscreen*

[7] Picture credits - http://ecomputernotes.com/computer-graphics/basic-of-computer-graphics/what-is-frame-buffer

[8] i.MX6 Processor Reference Manual - https://community.nxp.com/docs/DOC-101840

[9] FocalTech Capacitive Touch Panel Controller - https://www.newhavendisplay.com/app_notes/FT5x06.pdf

[10] I2C Interface - https://www.engineersgarage.com/tutorials/twi-i2c-interface

[11] Genode Foundations - 16.05 - http://genode.org/documentation/genode-foundations-16-05.pdf

[12] Spin Lock Header - <genode-dir>/repos/base/src/include/base/internal/spin_lock.h

[13] GPIO Component Header - <genode-dir>/repos/os/include/gpio/component.h

[14] i.MX53 Reference Manual - https://cache.freescale.com/files/32bit/doc/ref_manual/iMX53RM.pdf

[15] MfgTool Download link - https://www.nxp.com/webapp/sps/download/license.jsp?colCode=IMX_6DQ_MFG_TOOL