Migration of Diagnostics Application and Feature Addition in Android TV

Submitted By

Toshika Jain

16MCEC25

DEPARTMENT OF COMPUTER ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2018

Migration of Diagnostics Application and Feature Addition in Android TV

Thesis

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

Submitted By

Toshika Jain

(16MCEC25)

Guided By

Prof. Vishal Parikh

Nirma University, Ahmedabad.

Mrs. Anjuliz Ahmad

ARRIS India Pvt. Ltd.

DEPARTMENT OF COMPUTER ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2018

Certificate

This is to certify that the thesis entitled **"Migration of Diagnostics Application and Feature Addition in Android TV"** submitted by **Toshika Jain (Roll No: 16MCEC25)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering of Nirma University, Ahmedabad, is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this thesis, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Vishal Parikh
Guide & Assistant Professor,
Computer Engineering Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Priyanka Sharma
PG Coordinator-CSE,
Associate Professor,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Sanjay Garg
Professor and Head,
Computer Engineering Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Alka Mahajan
Director,
Institute of Technology,
Nirma University, Ahmedabad

7 May 2018

Internship Certificate

This is to certify that **Toshika Jain (Roll No: 16MCEC25)**, a student of M. Tech. Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad was working in this organization since 5 June 2017 and carried out her thesis work titled "**Migration of Diagnostics Application and Feature Addition in Android TV**". She was working as a Software Intern under the supervision of Mr. Santosh Honagudi (Manager), and Mrs. AnjuLiz Ahmad (Manager). She has successfully completed the assigned work and is allowed to submit her dissertation report. The results embodied in this project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

ARRIS team wishes you best in future endeavors.

For ARRIS Group India Private Limited

8/5/18

Santosh Honagudi
Manager
Software Engineering

Harish Kumar Puttaraju
Senior HR Systems Specialist
Human Resources

09/05/2018

AnjuLiz Ahmad
Manager
Software Engineering

ARRIS GROUP INDIA PRIVATE LIMITED
BANGALORE
560042

Statement of Originality

I, **Toshika Jain**, Roll.No.**16MCEC25**, give undertaking that the Thesis entitled "**Migration of Diagnostics Application and Feature Addition in Android TV**" submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science and Engineering** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made.It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

_____

Signature of Student

Date:

Place:

Endorsed by

Prof. Vishal Parikh

(Signature of Guide)

Acknowledgements

First and foremost, sincere thanks to Mr. Anil Radhakrishnan, Director, ARRIS India Private Limited, Bangalore.

I would like to thank, Mr. Santosh Honagudi and Mrs. AnjuLiz Ahmad, ARRIS India Private Limited, Bangalore for their valuable guidance.They have given me much valuable advice on this project work.

It gives me immense pleasure in expressing thanks and profound gratitude to **Prof. Vishal U. Parikh**, Assistant Professor, Computer Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his valuable guidance and continual encouragement throughout this work. The appreciation and continual support he has imparted has been a great motivation to me in reaching a higher goal. His guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

It gives me an immense pleasure to thank **Dr. Sanjay Garg**, Head of Computer Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr Alka Mahajan**,Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation she has extended throughout course of this work.

I would also thank the Institution, all faculty members of Computer Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

- Toshika Jain

16MCEC25

Abstract

Phase I

The scope of the work is migration of Diagnostic software (Autodiag) from older Graphics standard DirectFB to the latest standard OpenGL.In the process of migration, we may need to adapt application layer interfaces to port it on above OpenGL component. It also includes testing of necessary OpenGL graphics interfaces on the specified hardware platform.Before doing the migration directly on any new project, proof of concept (POC) will be done on one of the existing hardware platform (eg. Telefonica Hw) and use Diagnostic Software (AutoDiag) on which porting OpenGL on URSR nexus platform release 17.1.As part of this POC migration activity, we can also understand the performance of OpenGL and also implement and test some of the Graphics features using demo applications.

Phase II

For feature addition in Android TV, the task list started with giving the POC for Android TV with Android N, followed by adding the feature of Auto-Pairing in `DEV_cory` branch, the main reason for adding this feature in the particular branch was that, BLE RCU can be paired with STBs but for pairing and connecting, manual settings has to be done. So with this new feature as soon as the STB boots up, the BLE RCU should be automatically paired and connected to the STB. The next task was to add other BLE RCUs in this feature, followed by porting of the feature in Android O. The next task that followed after this was to configure GPIOs with respect to the RCU pairing, connection and disconnection. The last task was to test one app that is BCM OTA update app, so that the platforms can be flashed without taking apart the front cover and without using RS232.

Abbreviations

STB	Set Top Box
AOSP	Android Open Source Project
GTVS	Google TV Services
BCM	Broadcom
OTA	Over The Air
BLE	Bluetooth Low Energy
RCU	Remote Control Unit
UEI	Universal Electronics
AutoDiag	Auto-Diagnostic Application
AF	Application Framework
IF	Interface Foundation
DFB	DirectFB
BSG	Broadcom Scene Graph

Contents

List of Figures

Chapter 1

Phase I - Migration Of Auto-Diag Application

1.1 Overview

When there is any problem in set-top box, the customer goes to service centre.If the problem is related to any hardware components it is fixed at service centre and the box is returned to customer, if the problem is related to any software component the set-top box is sent to factory again.In factory, Auto-Diagnostic App is used to find any kind of software related problem in STB.

As of now, Broadcom is delivering DirectFB software component ported, integrated and delivered on top of every URSR Nexus release. Also, Broadcom supports every issue observed and raised on DirectFB component and delivers respective patches.Broadcom is clear on their roadmap to support DirectFB until end of this year(2017), and not sure of this support beyond this year. Its a risk to deliver and support AutoDiag on the current and new sets of platforms. Hence the advantage of moving to OpenGL is that Broadcom has plans to continue their support on OpenGL, OpenGL full of festures, also supports acceleration of 3D Graphics , and is supported on multiple platforms.In Arris, multiple projects are integrated with DirectFB on URSR nexus release 15.x and 16.x.

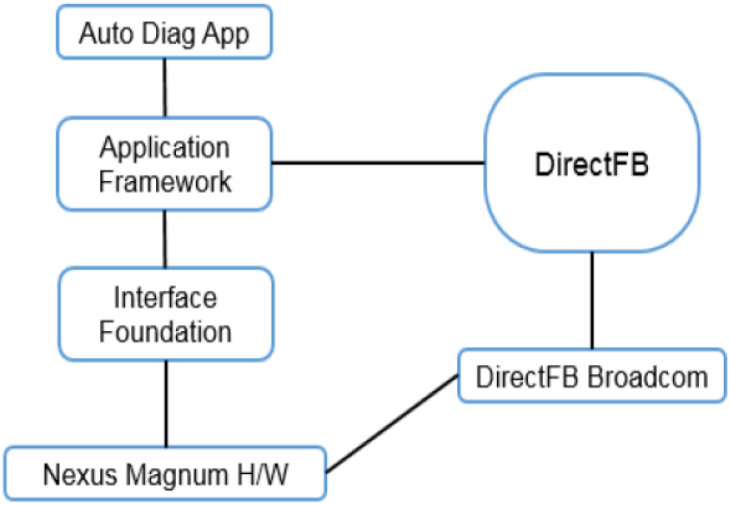

Figure 1.1: Auto-Diag Application Block Diagram

1.2 Implementation for migration of platform from DirectFB to OpenGL

First of all the code is checked out from server with the use of SVN main, it is build and then flashed on the board to see the Auto-Diag UI. As soon as the UI comes up on the TV, from there starts the research for finding the options for changing the platform. Three approaches have been taken in this platform migration. Before this the dependency of DirectFB has been found in the Auto-Diag Application.

Approaches taken for changing the Auto-Diag platform from DirectFB to OpenGL:

- Integrate OpenGL with Nexus.Build 3D examples and Converted the examples in 2D.
- Explore the Options for any above layer or libraries over OpenGL ES 2.0 for direct 2D rendering.(As there are no direct APIs for rendering 2D graphics in OpenGL ES 2.0)
- Explore the source code itself for some inbuilt libraries if the 2nd Approach doesn't works.

All the three approaches have there advantages and disadvantages. In first approach

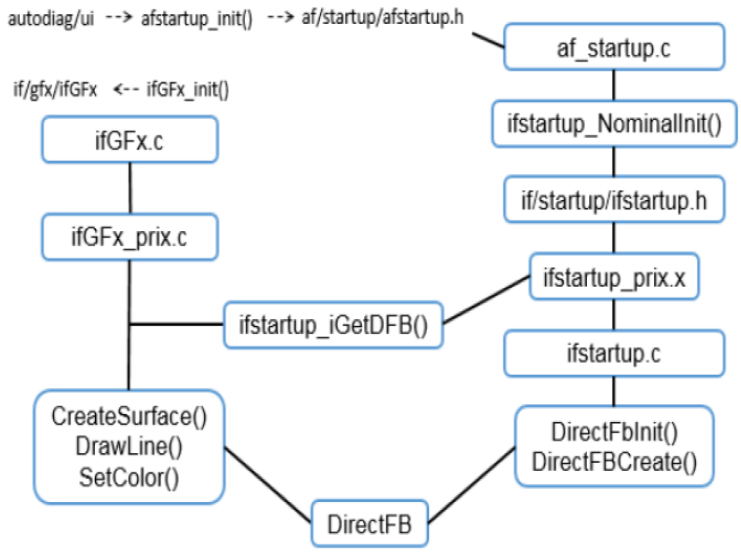

Figure 1.2: Dependency of DirectFB in Auto-Diag

the 2D drawing is done with the help of Nexus so it was not of much use, because Auto-Diag App UI needs text, splash-screen and some other 2D renderings, which were not possible with the help of Nexus APIs. Also in case of DirectFB, BCM is providing glue layer which is present in the form of 'AF' and 'IF', and in this approach, the glue layer is the egl which is the embedded glue layer, which is not widely used and also it hard to use. Also before the first approach the V3D core was enabled, as because OpenGL is using V3D core in background, and a list was prepared to with all the DirectFB APIs used in the codebase.

In the second approach, more options was searched to replace egl and also as OpenGL does not have any direct 2D rendering APIs, so options was searched for some open-source libraries to use OpenGL ES in the backend to do 2D rendering.With options like QT, Skia, Nanovg etc., GLFW was preferred to use in replacement of egl, because GLFW can be easily downloaded from the internet, also it is not as difficult to use as egl. GLFW was used for window creation and context creation. But one more library is needed with GLFW, because all GLFW can do is window and context creation so one more library is needed for 2D rendering which can use GLFW and OpenGL in the backend. So Nanovg is used above GLFW, as Nanovg has direct APIs to render 2D, also the dependencies it has was GLFW. The dependencies for GLFW is only x-org package.This second approach was rejected due to the dependencies of GLFW which was the x-org package which was

not in the codebase.

Figure 1.3: Text Rendering with BSG

In third approach, framework3d or BSG was used. BSG was there in the codebase itself, so there was no need to find dependencies because it was provided by BCM as a part of package in the trellis folder which contains all the open source soft-wares. BSG was able to draw all the 2D and 3D drawings. Some of the APIs that were used in DIrectFB was also mapped in BSG. But the disadvantage with BSG was that it was in C++ and DirectFB was in C so we need to write wrapper classes to implement the other APIs of DirectFB. 2D text rendering and one splash screen rendering was showed in demo with help of BSG.

At last a comparison table was made between the pros and cons of DirectFB, OpenGL, Play-assure and Skia.

Chapter 2

Phase II - Feature Addition in Android TV

2.1 Motivation

Android is normally known as a portable working framework created by Google. Android, in any case, began as an autonomous organization in 2003 and was acquired by Google in 2005. The same number of other inserted working frameworks it depends on Linux and acquires a great deal of the Linux qualities, obviously likewise includes a considerable measure of gadget and industry particular highlights, and what you see on an Android gadget is entirely different from what you see on a work area Linux PC.

There are essentially 3 things that describe Android :

- The multi screen or multi gadget bolster, with a steady end-client encounter. More than 80 percent of cell phones shipped today accompanies Android and more than half of the tablets.
- The broad access to applications, approximately 3M applications are accessible through the Google Play Store today.
- Thirdly, Google. Google is available in a large number of the parts of Android. This opens up bunches of potential outcomes and gives loads of highlights and administrations with no permit costs, yet in addition makes a few questions and vulnerabilities.

Since the presentation on the Smart Phone showcase in 2008, Android has now assumed control over the scene. It has ventured into tablets, wearable (smart-watches), autos, Internet of Things and as of late onto the TV screen through Android TV. The street to enter the TV showcase has been somewhat unpleasant and Google has fizzled two or three times (for example with Google TV). Be that as it may, Google has possessed the capacity to adjust to the administrators needs and with Android TV and the most recent Operator Tier design, Android is certainly making a major passageway as a working framework and structure on set-top boxes. Late insights demonstrates that there are in excess of 2B dynamic Android clients in 190 nations. What's more, on the TV side, there are around 20 pay TV administrators who as of now has propelled or are going to dispatch Android based set-top boxes.

From a product point of view, Android begins with AOSP, which is an open source working framework, much the same as Linux (which AOSP really expands upon). AOSP isn't intended for a particular industry or reason, yet shapes the establishment for all the business specific arrangements. With AOSP, we needn't bother with any business or specialized connection to Google, however then again we are "all alone". We don't gain admittance to Google's restrictive administrations, the Google TV Services (or GTVS for short). That implies no YouTube, no Cast usefulness and no Play Store. We additionally need to connect with outsider application suppliers (like Netflix) our-self and secure our own particular licenses for all the applications that we need to make accessible to our end-clients. As a contrasting option to AOSP, we as an administrator, can utilize some other Linux based TV working framework, similar to the ARRIS KreaTV IPTV software stage.

2.2 Android TV

Android TV then again is an entire structure for pay TV administrations. It expands on AOSP, yet includes a system and an arrangement of pre-incorporated TV highlights. Most noticeable is of-course that we get full access to the GTVS. This incorporates YouTube and the as of late propelled YouTube for Kids applications. We additionally get Cast usefulness with an indistinguishable usefulness from the Google Chrome-cast gadget. Through the Google Play Store we gain admittance to in excess of 3000 applica-

tions particularly intended for Android TV and the TV screen, with no compelling reason to specifically arrange rights with the distinctive application suppliers.

In this way, in synopsis, Android comes in for the most part two flavors, AOSP and Android TV. Android TV gives us a full TV system which by and large means a shorter time to advertise. We likewise gain admittance to some fundamental Google includes that are not accessible in the event that we utilize AOSP. With Android TV, ARRIS as a provider and we as an administrator need to go along to a few necessities from Google. We can not utilize an outsider application store for example and we can not keep outsider applications from appearing in the Play Store. We likewise need to take after the Google refresh and update plans. However as Google sees the TV, the STB and the administrator condition as a magnificent stage for connecting with their own particular administrations, they have found a way to limit this threshold and will no doubt keep on doing so.

With a specific end goal to make it easier for administrators to choose Android TV, Google has as of late characterized a variation of Android TV called Android TV Operator Tier. With this variation, we as an administrator gain substantially more prominent power of the client experience and look and feel. We likewise get the likelihood to get Revenue Share through Direct Carrier Billing. For ARRIS, who is one of the universes biggest supplier of Pay TV administrations and hardware, the regular concentration is Android TV Operator Tier.

2.3 Why Android TV?

2.3.1 Android TV Benefits

Presently how about we plunge into the primary advantages of utilizing Android TV from ARRIS. In short we will see that Android TV can help putting up down the Time For sale to the public, increment Flexibility and Control and in addition the pace of Innovation. It likewise gives best in class security and an unmatched access to TV related applications.

- Android TV accompanies a system planned with the motivation behind abbreviate Time To Market for different TV administrations.

- It offers an unified and bland system, the TV Input Framework (or TIF for short) for coordination of all TV related administrations.

- At the back end of TIF, ARRIS incorporates DVB tuners, SI parsers and multicast IPTV arrangements. ARRIS additionally performs combination of CA and DRM frameworks and also DVR and time move.

- At the opposite end, from the UI side, ARRIS works with Google standard applications and different accomplices keeping in mind the end goal to have the capacity to offer an assortment of client experience and look and feel. On account of TIF, distinctive blends of media circulation and UIs can be immediately consolidated and conveyed.

- At the opposite end, from the UI side, ARRIS works with Google standard applications and different accomplices keeping in mind the end goal to have the capacity to offer an assortment of client experience and look and feel. On account of TIF, distinctive blends of media circulation and UIs can be immediately consolidated and conveyed.

- To additionally enhance Time to Market, ARRIS has built up a wide arrangement of equipment stages that are prepared to be conveyed inside segment lead times.

- On the product side, ARRIS is applying a product offering approach which implies that we have full reuse of improvement and highlights over all equipment stages and over every one of our clients, all adding to a Faster Time to Market.

- Development might be one of the primary attributes of Android. For each new Android discharge new highlights and upgrades are propelled.

- We have as of late observed YouTube for Kids and another inquiry/aide include being discharged. Moreover effectively existing highlights gets expanded. This is particularly the case for the cast highlight, which now is bolstered in excess of 1000 Android and iOS advanced mobile phone and tablet applications.

- The same is valid for the Google Play Store, where TV related applications are constantly included and has now achieved more than 3000. Esteem included applications are increasingly observed as an approach to keep control over HDMI 1

input and keep end clients from moving without end to another, non administrator controlled, condition.

- Receptiveness does not really mean less secure. Android acquires security components from the basic Linux working framework. Notwithstanding this Google has found a way to expand the security through, for example, application sand-boxing, marking and secure intercommunication.

- Updates are done in a secure path through Google Over The Air (or GOTA) refreshes or through administrator or provider restrictive arrangements with a similar level of security. Google likewise discharges month to month security updates and there is an all around characterized process for getting these updates sent in your system.

ARRIS commitment to Innovation inside Android TV gets through the consistent guide work, where we expand media bolster through TIF and different administrations applicable for the TV business.

2.4 Outline Of Thesis

Chapter 3 (*Technical Specifications*) Technical specifications of servers used and soft wares used are included in this chapter.

Chapter 4 (*Platforms*) Various Platforms, remotes used and their specifications are included in this chapter.

Chapter 5 (*Objectives of Feature Addition*) The objectives of Feature Addition are described in this chapter.

Chapter 6 (*ArrisCustomizer Application*) This chapter describes approach used to integrate Auto-Pairing Application with `DEV_Cory` branch, its POC with various platforms and with various remotes, and the porting of Auto-Pairing Application on Android O.

Chapter 7 (*Configuration Of GPIOs wrt BLE RCU*) This chapter describes how GPIOs are configured with respect to the BLE RCU behavior.

Chapter 8 (*Conclusion And Future Scope*) Conclusion of the work is described in this chapter, and the future scope and maintenance of the work done.

Chapter 3

Technical Specifications

3.1 Configurations Of Android Servers

CPU i7, 16 GB RAM, Harddisk > 2TB
Ubuntu LTS 14.04.
Python 2.6 – 2.7 from python.org
GNU Make 3.81 – 3.82 from gnu.org
Git 2.9 or newer from git-scm.com
Ubuntu - OpenJDK 8
Java 1.8 or newer
Access/permission for Android repos in Devhub.

Table 3.1: Android Server Specifications

3.2 Tools

Source Insight
Putty
Notepad++
WinScp
Tera Term
BCompare
BroadBand Studio
Android Studio

3.3 Other Servers, Platforms and Remotes

3.3.1 Sweden Server

One of the Build server in Linkoping which is used is:

- benromach.lab.swelin.arrisi.com

3.3.2 Platforms

The platforms used in the Process are:

- Askival
- VIP61n2
- zh4515 - F070(7268), F079(7271), F072(72604)

3.3.3 Remotes

BLE RCUs used in the process are:

- Ruwido
- UEI Remote Control

Chapter 4

Platforms

4.1 List of Platforms

The platforms used in the process are:

- **Askival**
- **VIP61n2**
- **zh4515 - F070(7268),F079(7271),F072(72604)**

4.2 Askival

Askival Board has been shown in Figure 4.2.
Following steps should be taken to load the HW platform with Android Image:

- Update BOLT:
  - In ./out/target/product/askival/ ofthe workspace, we should find some bootloader images called bolt-<X>.bin, where <X> indicates the different hardware/chip variants. Depending on the target variant, we will need to pick the corresponding image when upgrading bootloader on the platform. According to the current amount of memory, bolt-bb.bin has to be flashed on the platform.
  - Copy the right bootloader image to use on the TFTP server.

- In BOLT's command prompt, issue the following command to flash the bootloader image to teh target platform.The following command assumes that the platform is connected to some network via Ethernet that can communicate with the TFTP server.
  * BOLT> ifconfig eth0 -auto
  * BOLT> flash <tftp-server-ip>:<path-to-bolt-xx.bin> flashxxx
- Reboot the target platform after the bootloader is successfully upgraded.

- Check Box Mode (RTS):
  - To check which mode the platform is in, type "rts" at BOLT's command prompt. Here is a sample of what the command returns from BOLT:
    * BOLT> rts
    * rts 00 [20160116022007_Box0_box0]
    * BOX MODE: 1
    * *** command status = 0
  - If for whatever reasons we need to override the default box mode to some other values, we can use the rts -set = <mode number> command from BOLTs command prompt.

- Update Android's partition table in the eMMC:
  - Locate the Android BSU file in out/target/product/askival/android_bsu.elf of the workspace and copy it over to TFTP server.
  - From BOLT command prompt, issue the following command to load the compiled Android BSU file from TFTP server:
    * BOLT ifconfig eth0 -auto
    * BOLT> boot -elf -noclose -bsu <tftp-server-ip>:<android_bsu.elf>
  - Once the BSU is successfully loaded, it should return to the BOLT command prompt. Note the IP address assigned to the platform.
  - Verify that BSU is loaded and that the 'android boot' command is available as shown in Figure 4.3:

```
BOLT> help android boot
SUMMARY
Load an Android boot image into memory and boot it
USAGE
android boot [options] [arg]
This command loads and boots and Android image. The
optional arg can be used to override any builtin
kernel command line arguments.
The file or partition to use for booting can be
specified with environment variable or explicitly
in the command with -i option.
The format can be:
dev:filename (e.g. usbdisk0:boot.img)
dev (e.g. flash1.partition)
If -rawfs option is enabled, then it must be:
usbdisk0:partition (e.g. usbdisk0:boot)
The following environment variables can be used:
ANDROID_BOOT_IMG - for normal boot image
ANDROID_RECOVERY_IMG - for recovery boot image
OPTIONS
-rawfs ........... Load the file from an unformatted file system
*** command status = 0
```

Figure 4.1: BSU loaded on HW platform

- Once it is confirmed that Android BSU is properly loaded, the next step is to use "android fastboot" to update the partition table of the platform.
- Put the target platform into fast boot mode:
  - BOLT> android fastboot -device=emmcflash0
- Use fastboot to flash "gpt.bin" to target platform. In workspace, run the following command from Linux build machine:<verbatim> as shown in Figure 4.4. If we have a platform that has never been used for Android before, it is possible that there isnt a valid GPT pre-loaded to eMMC and we would see a log message complaining that GPT Header signature is wrong. This is harmless as long as we only update the 'gpt' partition as per the instructions:
  - croot
  - sudo ./out/host/linux-x86/bin/fastboot flash gpt ./out/target/product/askival/gpt.bin
- Use fastboot to reboot target platform from the host machine: sudo ./out/host/linux-x86/bin/fastboot reboot
- The partition table should have been updated by now. Run the "show de-

vices" command in BOLT to confirm the newly created partitions to show up properly.

  - Lastly, check if the BOLT environment variables still carry a valid MAC address. If we don't see a MAC address listed from the "printenv" command in BOLT, then program the board with the MAC address we recorded in earlier step or get BOLT to generate one for automatically based on the board serial number as illustrated below: BOLT> macprog <board-type> <serial-num> <board-rev>

- Flash Android BSU

  - With the partition table properly created in eMMC, we can now flash Android BSU to the "bsu" partition from BOLT command prompt as follows. This allows us to boot from the bsu partition instead of loading the Android BSU from TFTP server everytime after reboot:

    * BOLT> ifconfig eth0 -auto
    * BOLT> flash <tftp-server-ip>:<path-to-`android_bsu`.elf> emmcflash0.bsu
    * BOLT> setenv -p FB`DEVICE_TYPE` "emmcflash0"

- Hardware configuration partition

  - The hwcfg partition in eMMC device is specifically used for storing hardware specific content such as DRM binaries (e.g. drm.bin and `drm_hdcp1x`.bin) and Wi-Fi configuration file (e.g. nvm.txt) that are unique to each platform. This section describes all the steps for preparing and generating the hwcfg.img image that is unique to HW platform:

    * Create hwcfg.img
    * Flash hwcfg.img

- Image Flashing

  The following instructions outlines the steps we should use to flash compiled images to the rest of system partitions:

  - Load Android BSU from your previously flashed location:

        * BOLT> boot -elf -noclose -bsu emmcflash0.bsu Once BSU is successfully loaded, it should return to the BOLT command prompt.
        * Put target platform in Fastboot mode:
            · BOLT> android fastboot -transport=tcp -device=emmcflash0
    - From the host machine, flash all the images (boot.img, hwcfg.bin, system.img; userdata.img, cache.img and recovery.img). (Some images have to be flashed twice, such as boot.img in `boot_i` and `boot_e` partitions.)
    - Once you have flashed all the images, reboot the target from host machine:
        * sudo ./out/host/linux-x86/bin/fastboot -s tcp:<target_ip>:<port> reboot

Booting up your image

- From BOLT command prompt, issue the following command to load Android BSU that was previously flashed to your system:
    - BOLT> boot -elf -noclose -bsu emmcflash0.bsu

  Once BSU is successfully loaded, it should return to the BOLT command prompt.
- Boot up Android.
    - BOLT> android boot
- To start automatically Android, we can set STARTUP variable as follows:
    - BOLT> setenv -p STARTUP="boot -elf -noclose -bsu emmcflash0.bsu; android boot"

4.3 VIP61n2

VIP61n2 board has been shown in Figure 4.3. Following steps should be taken to load the HW platform with Android Image:

- Update BOLT the steps for updating BOLT are same as Updating BOLT in Askival Board.
- Install and run Android images from eMMC flash

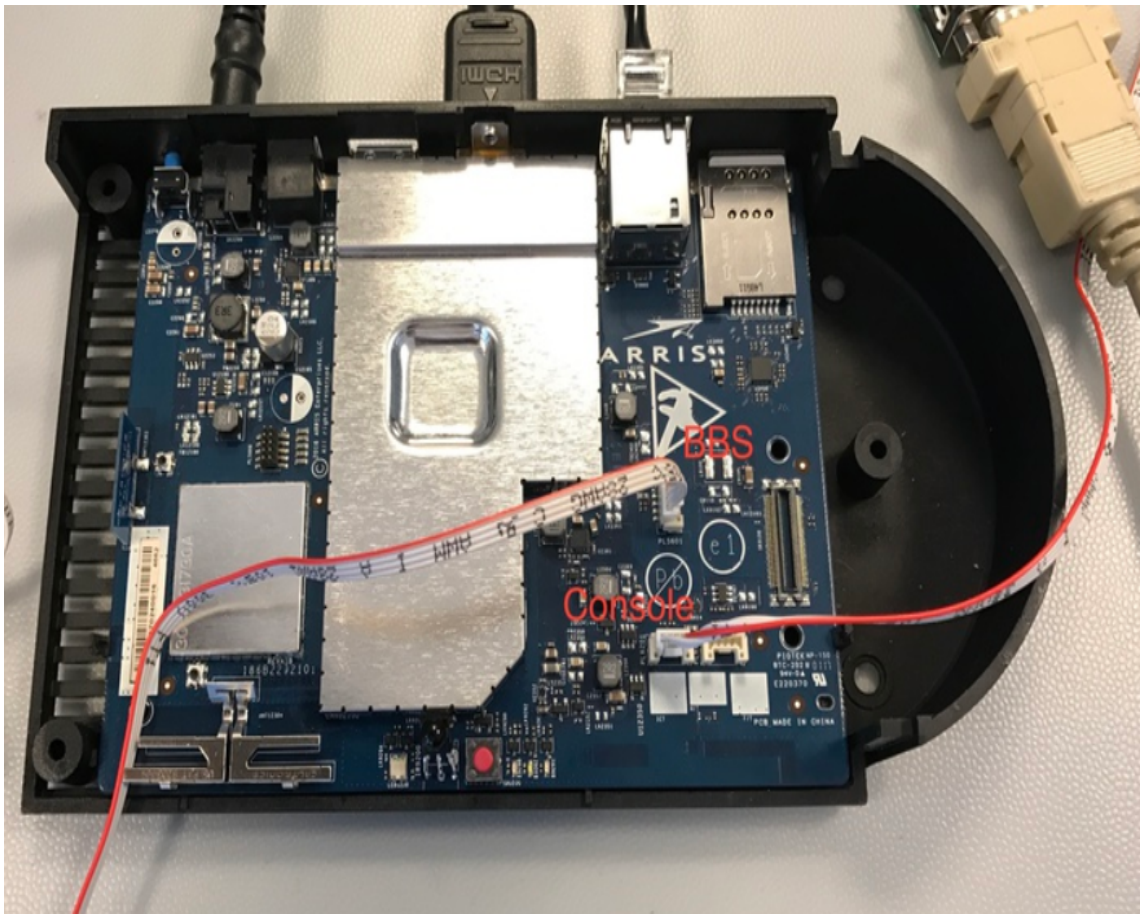

Figure 4.2: Askival Board

- Build Android from source
  - Android N: repo init -b `DEV_barracuda` -u ssh://git@git.arrisi.com/android/arris/manifest
  - Android O: repo init -b `DEV_barracuda_o` -u ssh://git@git.arrisi.com/android/arris/manifest
- Now lets sync and build it:
  - repo sync -j8
  - . build/envsetup.sh lunch `vip6102w_bell`-userdebug
  - make -j16

  Once the branch has been build the images can be found in out/target/product/vip61n2.

- Install on eMMC flash
  - Copy out/target/product/vip61n2/`android_bsu`.elf to TFTP root and run the following in the BOLT prompt:
    - BOLT> ifconfig eth0 -auto
    - BOLT> boot -kelf -noclose -bsu <tftp-server-ip>:`android_bsu`.elf
    - BOLT> android fastboot -transport=tcp -device=flash0
  - Now device is ready to receive fastboot flash commands from computer via TCP.
  - In the out/target/product/vip61n2 directory, type the following command:
    - ./arris-provision-device -i <stb-ip>

    The script above would call fastboot utility with the required args. The actual fast boot command would look something like this:
    - fastboot -s tcp:<IP of the STB>
    - flash gpt gpt.bin
    - flash boot boot.img
    - flash bsu `android_bsu`.elf
    - flash system system.img

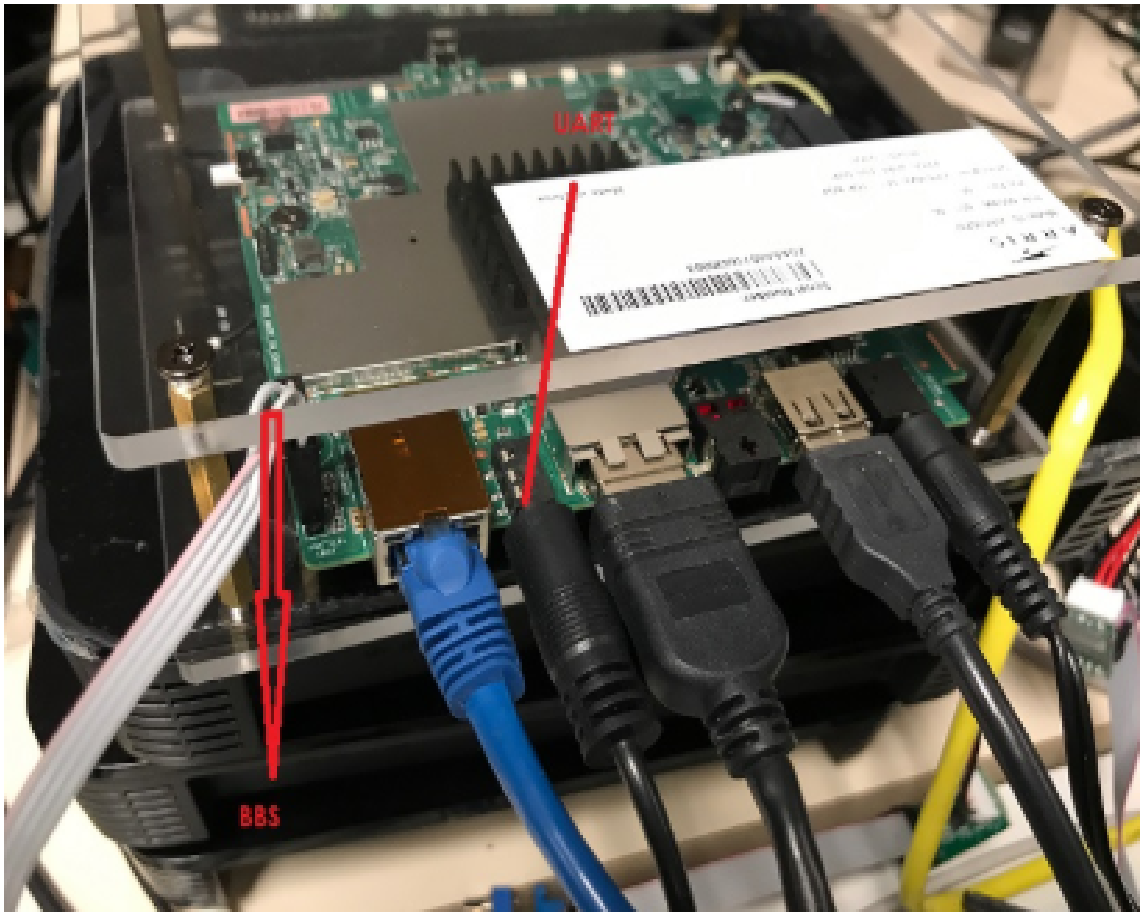

Figure 4.3: VIP61n2 Board

* flash userdata userdata.img
* flash cache cache.img
* flash hwcfg hwcfg.img
* flash recovery recovery.img

4.4 zh4515

zh4515 boards have been shown in Figure 4.4 and Figure 4.5.

The steps for loading the Android Image on these HW platforms are same as Askival Board. Just at every place where askival is there in the steps, we have to replace it with zh4515.

Steps to checkout the code and build image for zh4515:

- repo init -b `DEV_cory` -u ssh://git@git.arrisi.com/android/platform/vendor/arris/manifest

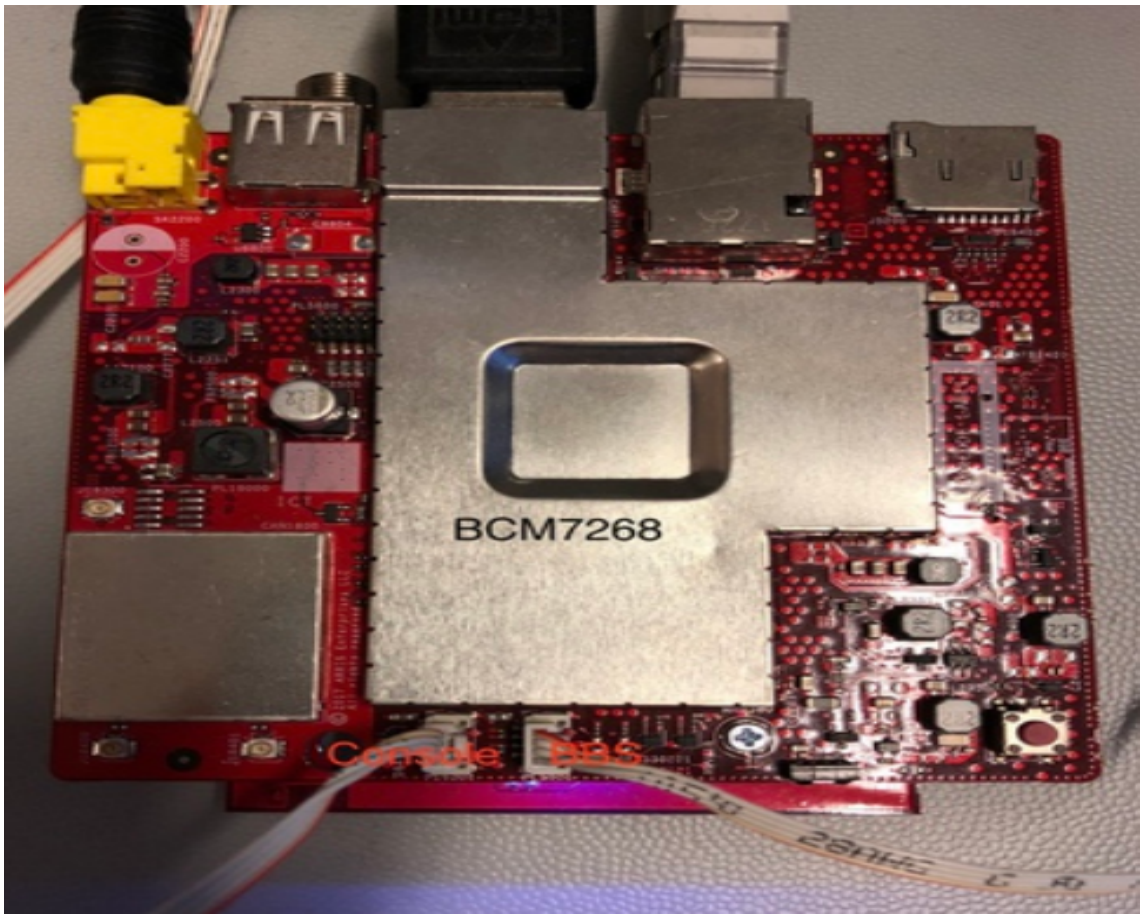

Figure 4.4: F070(7268) Board

Figure 4.5: F072(72604) Board

- repo sync -j8
- . build/envsetup.sh lunch elements-userdebug
- make -j8 — tee build.log

4.4.1 Updating The STB With BCM OTA Update App

From the top of workspace, the OTA package can be built with the following command:

- source ./build/envsetup.sh
- lunch zh4515-userdebug
- make otapackage

After the build completes, an OTA package should be generated at the out/target/product/zh4515 directory.
There are two options to apply this zip file:

- Option 1:
  - Push the zip file to cache partition and trigger the update through console as follows:
    - On the host computer:
      adb push <my ota package>.zip /cache/update.zip
    - On the box:
      - mkdir -pv /cache/recovery
      - echo `update_package`=/cache/update.zip > /cache/recovery/command
      - reboot recovery

      After the "reboot recovery" command, system will reboot into recovery mode, and apply the update.zip file. If cache partition is NOT large enough to hold the entire OTA package, we can use the sdcard instead of cache partition in this method.
  - Option 2:

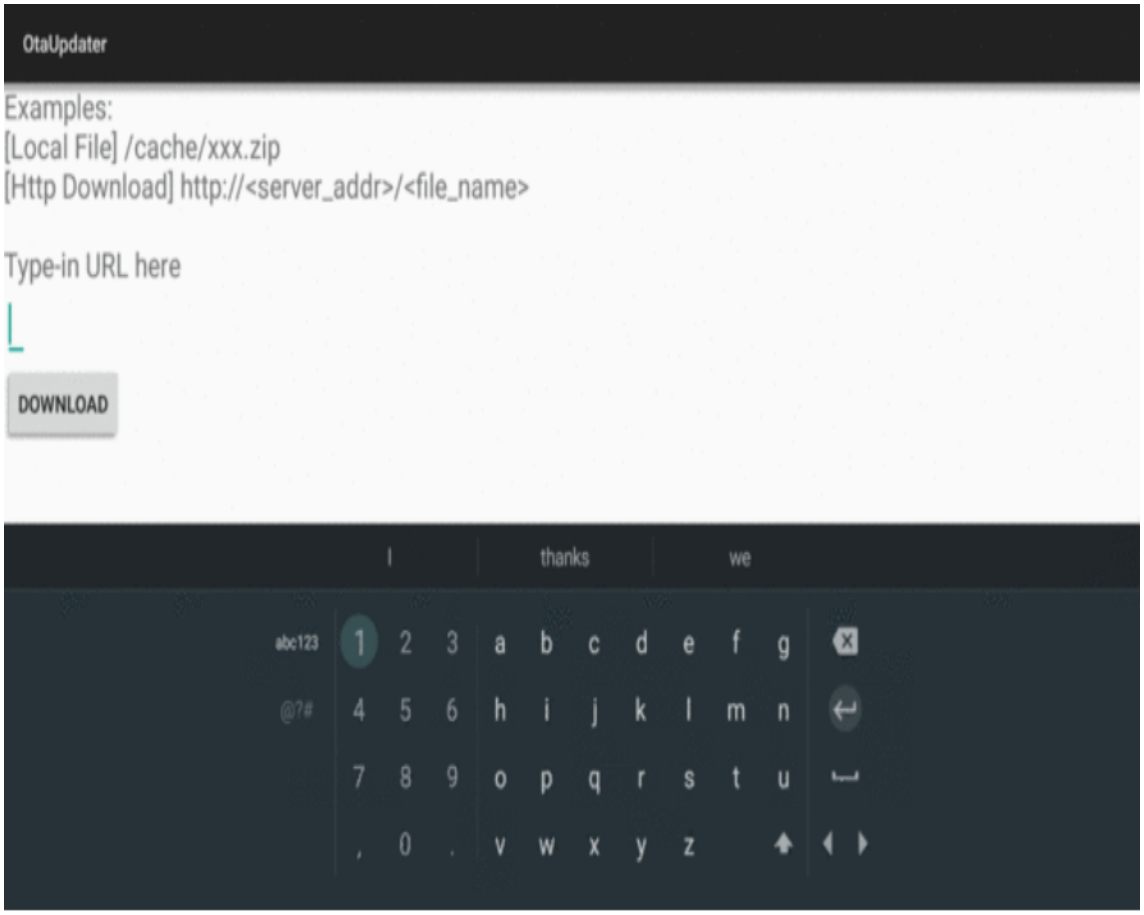

Figure 4.6: BCM OTA Update App

* Use the built-in BcmOtaUpdate app. The app is in vendor/broadcom/`bcm_platform`/apks/BcmOtaUpdater. If we don't have this application installed on your STB, we can build the latter with the "mm" command in the directory of this application. Once we have checked the application is available, we need to setup an HTTP Server to host the update file.

– On the box:

* Navigate to "Settings" -> "Apps" -> "System apps" (or "Downloaded apps" if installed manually)
* Look for "OtaUpdater" apk, and open it
* Input the URL to the http server and update file:
  ie : http://`<http_server_address>`/<my ota package>.zip
* Then click on Download Button
* The Apk will download the package, and when finished, it will prompt for

installation confirmation.

* The STB will reboot in "recovery" mode to update the different partitions with the appropriate images.

Chapter 5

Objectives Of Feature Addition

5.1 Objectives

- This Feature Addition task has been divided into various sub-tasks. But the main task, was to add Auto-Paring feature in Android TV.

- " What is Auto-Pairing ?
  Auto-Pairing: Connect and pair the Bluetooth device manually and then just turn off and on Bluetooth and it will auto connect.

- " What are the use-cases for this feature in Android TV ?
  One of the use-case can be: We can connect our Bluetooth headphones with STB, and there is no need to pair it every time whenever we want to use headphones.
  Use-case decided, keeping business in mind - To connect a Bluetooth remote as soon as it comes in vicinity of STB.

- Everytime when the STB is powered on/wakes up, it should automatically pair with the Bluetooth remote, so that the end user dont have to use an IR Remote to go to Settings and then pair the box with Bluetooth remote.

- Subtasks decided for this end goal:
  - Give a demo with Pre-Built images.
  - Checkout code and build the code for different-different platforms.

  - Platforms decided were under the product name zh4515 and the boards are F070 (7268), F072 (72604), and F079 (7271).
  - Checkout Auto-pairing app from DEV barracuda and integrate it with DEV cory.
  - Update the Jiras.
  - POC for different remotes on different platforms.
  - Update the Jiras for comparison and performance of different remotes on Different platforms.
  - Currently the bluetooth remotes available are: Ruwido BLE RCU, UEI Remote Control.
  - Push the code for Auto-Pairing App in DEV cory and fix any issues after that.
  - For beginning, all this work will be done with Android N, as soon as Android O is available, try to port that app on Android O.

- After Auto-Pairing is finished, the next task was to configure the GPIOs with behavior of BLE RCU with the STB.

- After configuration of GPIOs the expected behavior was that until the booting animation is there the LED should blink and as soon as the android logo comes up the LED should still be on, now as we start pairing the BLE RCU with the STB, led should start blinking on the board and as soon as the RCU connects the LED on the board should be on. If the BLE RCU disconnects from the board the LED should be off.

- Configuration of GPIOs should be done on two platforms namely - F070(7268) and F072(72604).

5.1.1 BLE RCU

Two BLE RCUs used for the POC were:

- Ruwido

- UEI Remote Control

5.1.1.1 Ruwido

LED behavior:

- Red LED will blink when the remote is in pairing mode.
- Green LED will blink when the remote is in IR mode.

Ruwido can be put in pairing mode by pressing "play/pause" and "home" buttons. For deleting the pairing "4" and "6" should be pressed.

5.1.1.2 UEI Remote Control

LED behavior :

- When the remote is in pairing mode Green LED will blink, if the remote is connected to STB then by pressing any key GREEN LED will be there as an indiaction.

For putting the remote in pairing mode press "OK" and "O" buttons and for deleting the pairing press "OK" and "back" buttons.

Figure 5.1: Ruwido BLE

Figure 5.2: UEI Remote Control

Chapter 6

ArrisCustomizer Application

6.1 ArrisCustomizer Application

ArrisCustomizer is an system application which runs in the backend of the system and provides auto-pairing feature to Android.For cloning the app the following command was used : git clone -b `DEV_barracuda` ssh://git@git.arrisi.com/android/vendor/arris/hals. This command will make a folder in the vendor/arris/ location named hals with apks in it. One of the apks will be ArrisCustomizer.

Before cloning the app, public key should be generated and copied to the devhub profile account.Initially ArrisCustomizer was containing many java files like for LED, IR mode, Auto-pairing feature etc. But as the requirement was only to port Auto-Pairing on `DEV_cory` branch so only some features was cherrypicked from the app. Rest all for beginning was deleted from the app.

But for first whole app was ported on the platform. For this the starting was done with testing the bluetooth of the box.Initially bluetooth of zh4515 was not enabled because it was enabled for BCM bluetooth chip and zh4515 has CSR Bluetooth so bluetooth was not working for STB, it was enabled by enabling the CSR bluetooth chip of STB.

For changing the platform, we have to change the export `NEXUS_PLATFORM` variable of zh4515.mk. If the platform is 72604, change it to 9760, if the platform is 7268, change it to 97268.The changes for enabling CSR bluetooth was also done in zh4515.mk. Now the

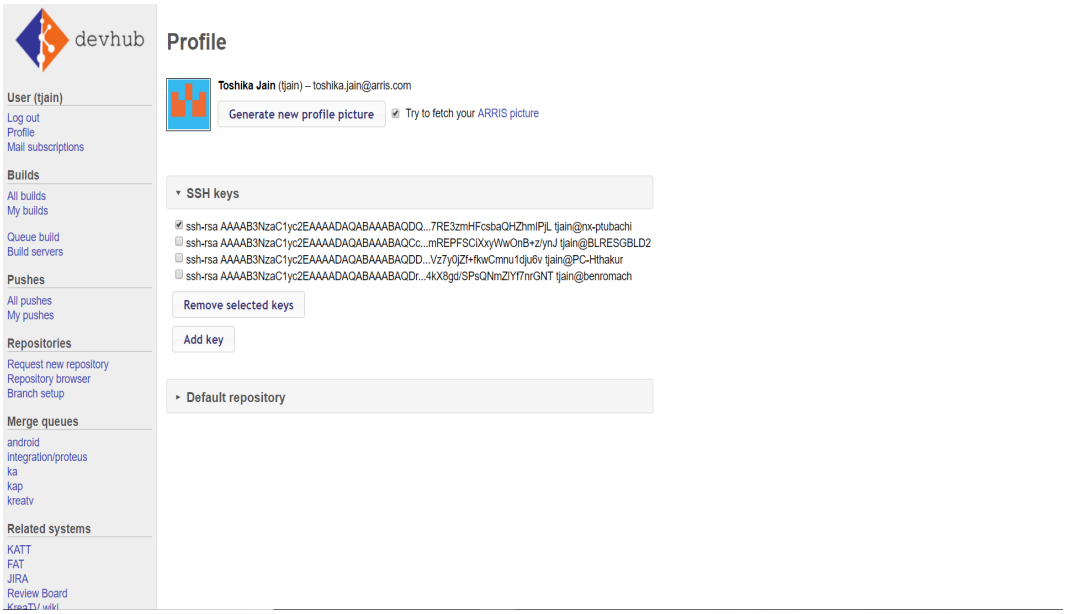

Figure 6.1: Devhub

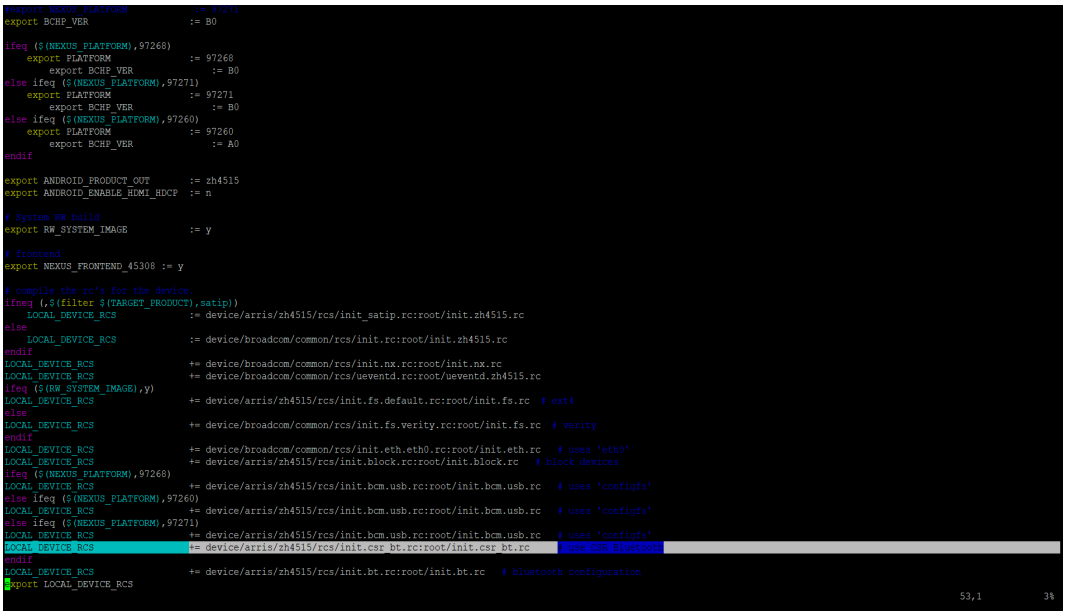

Figure 6.2: CSR Bluetooth Enabling

next task was to check whether the bluetooth for STB is discoverable or not. So it was checked with the help of phone, as before STB was not coming up in the list of bluetooth devices, now it was showing in the list of bluetooth devices.

Now as the STB is showing in the list of bluetooth devices.The next task was to add ArrisCustomizer in the code and to add it in zh4515.mk as `PRODUCT_PACKAGES` and then build again the code and flash the images on the board.

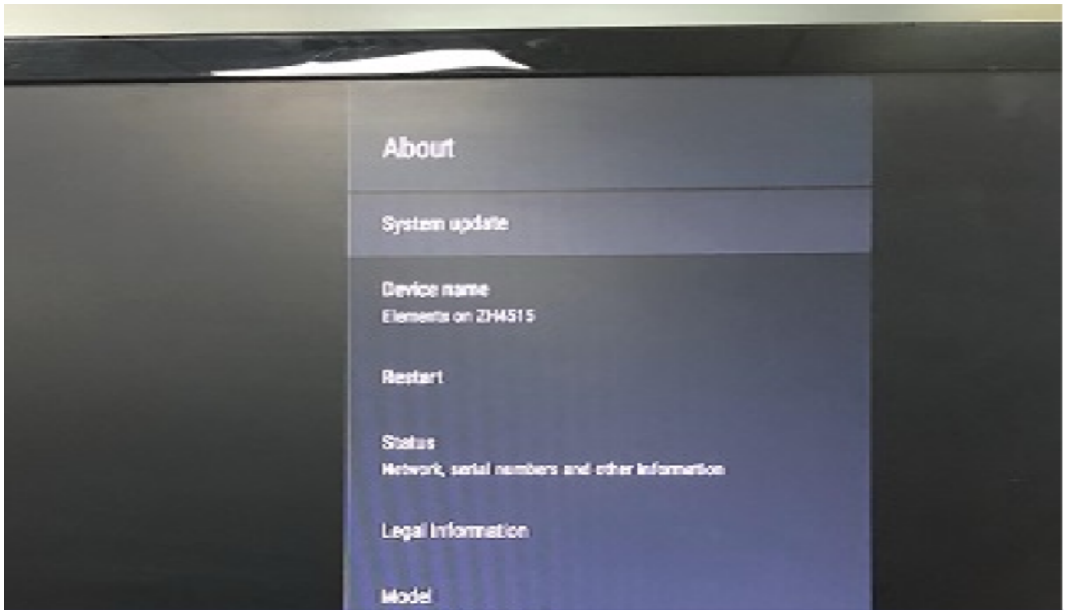

Figure 6.3: Device name of STB

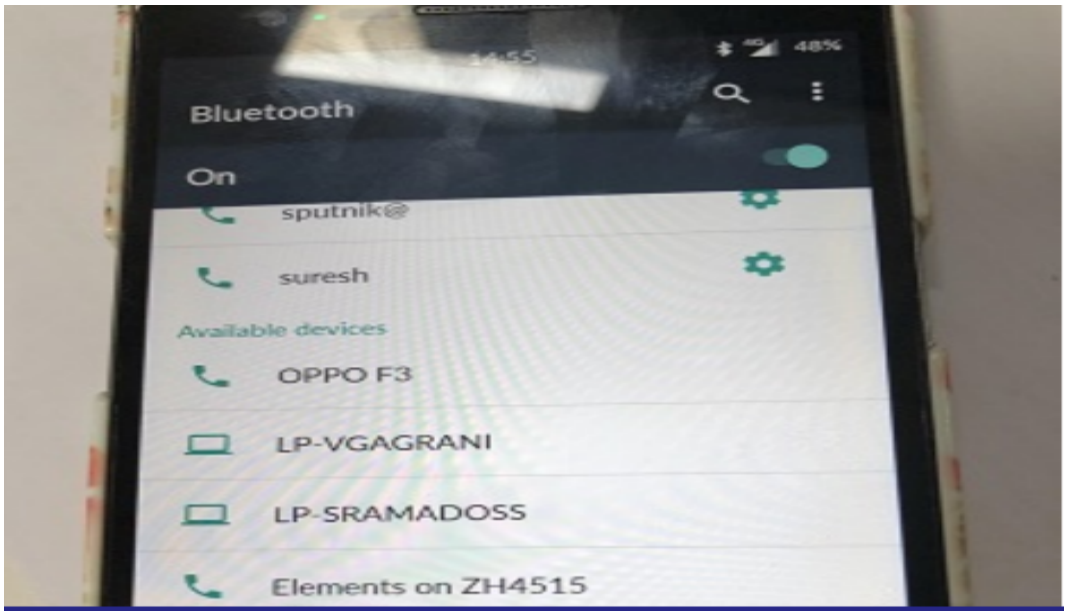

Figure 6.4: STB in list of bluetooth devices

But as soon as the image was flashed on the board, the app was getting crashed. So after this I started taking logs as soon as the box boots up. The box is then connected through adb and the logs are taken with the help of adb logcat. After taking logs the first keyword to search was ArrisCustomizer, and then noting down the process number of this app and then again find the logs with this process number.After going through the logs it was found that as platform is in search of input device that was device4 and as it was not able to find it so the app was crashing as soon as the box boots up. This crashing issue was solved with the help of a hack in which as soon as the code starts to find that

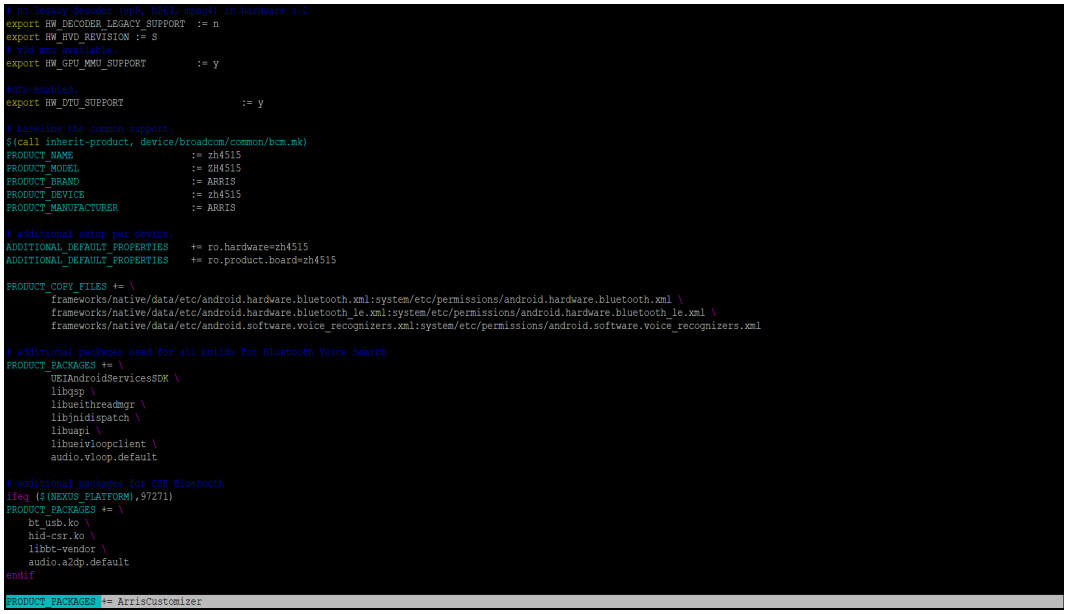

Figure 6.5: Addition of ArrisCustomizer in zh4515

particular input device it comes out from the loop, as because this input device was last in the list so as code was not able to find this it comes out of the loop. Now again after this fix, the code was again build and then flashed on the board to test if it was working or not.

This time the fix worked, but now we got to know that there was an update to the app so the app also and the full codebase was synced with git repository, now again the code was build and flashed on to the board to check if the new version of the app is working fine, this time as soon as the box boots up the App was crashed, so after taking the logs with the adb logcat we got to know that as the new version of the app has some extra features like wifi thread, network examination and led support. Among these features the led support feature was dependent on the codebase itself and it was talking to the platform with the help of nexus with setLed API. So setLed API was calling in one of the java files in the ArrisCustomizer app and from where it was going to jni layer and from there as it was a native call it was going to nexus to find the defination of setLed API. As `DEV_cory` nexus layer does not have setLed API so the ArrisCustomizer app was crashing.[1]

So the solution was this was to keep the app as minimum as possible and to cherrypick the features we need. So the setLed API was removed from the app, all the java files that was calling the setLed API was removed, jni layed was removed because at that time led

support was not required. Now again the code was build after this fix and reflashed on box and this time ArrisCustomizer app didn't crashed.

So now when the ruwido was put in the pairing mode then as soon as the box boots up ruwido was connected to the STB, this task was done with platform F079(7271).Some of the errors faced during this:

- Pre-Built images not working with 7268. As here 7268 is Askival board so there is hardware difference between Askival and F070 (7268).
- Not able to discover STB bluetooth with any other android device. As bluetooth was not configured for F079 (7271) because 7271 has different bluetooth chipset from 7268 and 72604, so they fixed the issue by configuring CSR bluetooth chip for F079 (7271).
- While building sandbox for 7271, facing some unexpected error due to Fedora version, later the build was transferred to Ubuntu 14 version and the build worked well.
- Changing the jack-port numbers, as default jack-port numbers for all were 8076 and 8077, the jack-port numbers were changed by going in jack-settings and then changing the jack-port numbers[2].
- Bash was not reflecting due to which build was getting disrupted, this was fixed by giving bash command to the console and exporting the `CCACHE_DIR` and `USE_CCACHE` to the bashrc.

The next task was to add support for UEI Remote control and to test it with F070 and F072. After adding the support for it was observed that F070 and F072 cant be paired and connected with ruwido. Before this one JIRA was created to follow this issue. After the observation of both the remotes with all the platforms one Comparison table was created.

6.2 Git Commit

After the demo on all the three platforms with both remotes, code was pushed in devhub, but one new issue was found with UEI in both F070and F072. The issue was after

```
      mRegistered = false;
      Log.d(TAG, "Receiver unregistered to listen to events: ACTION_FOUND and ACTION_DISCOVERY_FINISHED.");
    }
  }

  @Override
  public void onReceive(Context context, Intent intent) {
    final String action = intent.getAction();

    Log.d(TAG, "Inside onReceive with action: "+action);
    Log.d(TAG, "Waiting for action: ACTION_FOUND");
    if (BluetoothDevice.ACTION_FOUND.equals(action)) {
      // When discovery finds a device
      // Get the BluetoothDevice object from the Intent
      BluetoothDevice btDevice = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
      int rssi = intent.getShortExtra(BluetoothDevice.EXTRA_RSSI,Short.MIN_VALUE);
      final String address = btDevice.getAddress();
      String name = btDevice.getName();
      Log.d(TAG, "Device found, address: " + address + " name: \"" + name + "\"");

      if (address == null || name == null) {
        Log.d(TAG, "We don't accept strange device");
        return;
      }

      if ((!name.startsWith("UEI")) && (!name.toLowerCase().startsWith("ruwido"))) {
        Log.d(TAG, "We support UEI and ruwido only for the time being, hence "+name+" ignored.");
        return;
      }
        Log.d(TAG, "Remote accepted");
      notifyDeviceFound(btDevice, rssi);
    } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
      notifyFinished();
    }
  }

  private void notifyDeviceFound(BluetoothDevice device, int rssi) {
    Log.d(TAG, "Notifying device found.");
    if (mListener != null)
      mListener.onDeviceFound(device, rssi);
  }

  private void notifyFinished() {
    Log.d(TAG, "Notifying scanning finished.");
    if (mListener != null)
      mListener.onScanFinished();
  }
}
                                                                    89,30         Bot
```

Figure 6.6: Support Addition for UEI Remote Control

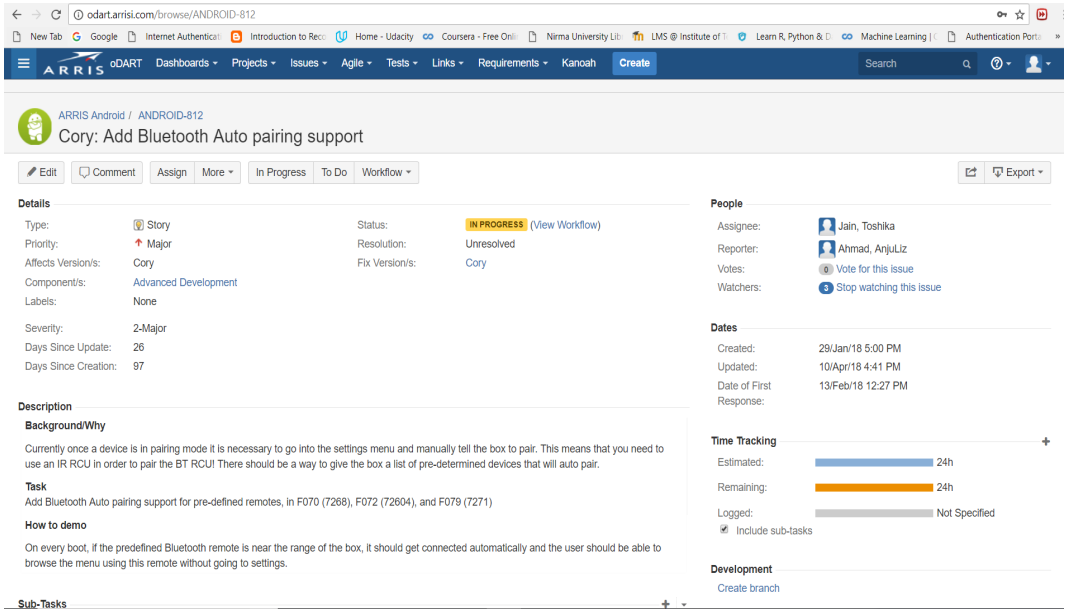

Figure 6.7: JIRA for adding Auto-Pairing support

Boards/Remote	Ruwido RCU	UEI remote
7271	Able to scan, pair and connect	Able to scan,but not able to pair and connect
7268	Able to scan,but not able to pair and connect	Able to scan, pair and connect
72604	Able to scan,but not able to pair and connect	Able to scan, pair and connect

Figure 6.8: Comparison of performance between HW platforms

connection of UEI in both the platforms, the remote was getting itself disconnected after some time.The reason behind this was, UEI was going in dead mode after some time and UEI comes only in pairing mode after you press both keys for putting it in pairing mode, so auto-pairing app was not able to connect to UEI again, once it is connected from the STB. Now this was the main function of auto-pairing app, even if the remote is disconnected the app will again try to connect with the remote if it is pairing mode. For ruwido, this was not the problem, because ruwido remains in pairing mode all the time, so even if it is disconnected it can again be connected with the help of auto-pairing app by itself without any manual settings.but for UEI, as after disconnection it was not going in pairing mode so auto-pairing app was not able to connect with UEI.

This issue was fixed by removing the part in the ArrisCustomizer which was removing the dead remotes and again a git commit was done to push the code in devhub.

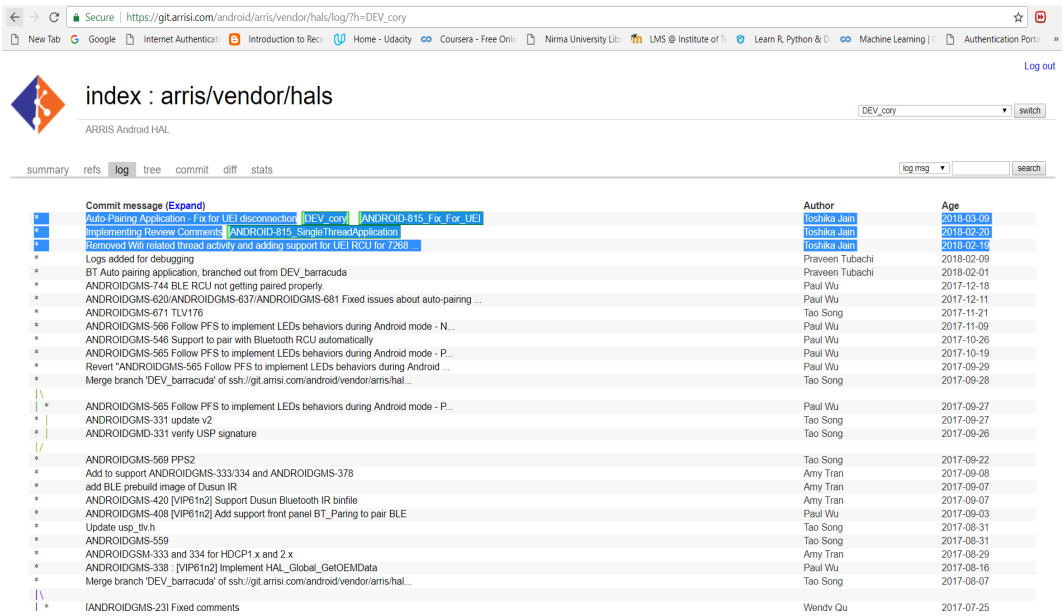

Figure 6.9: Git Commit

In between all this procedure logs were also added to ArrisCustomizer app to understand the flow of the code. After the App was successfully tested on Android N it was ported to Android O.In Android O, one issue came in auto-pairing app, that after UEI is disconnected, another UEI remote was anot able to connect withour reboot,also one more scenario was there that after UEI was disconnected, ruwido was not able to connect. Although this issue was not fixed but these scenario was also tested with manual settings

and there also happens to be same thing, so this was not the problem with auto-pairing app.In Android O, there was no problem like one remote is working with a specific board and the remote was working with other board. In Android O, both the remotes were working with all the three boards, and also auto-pairing app was working fine.

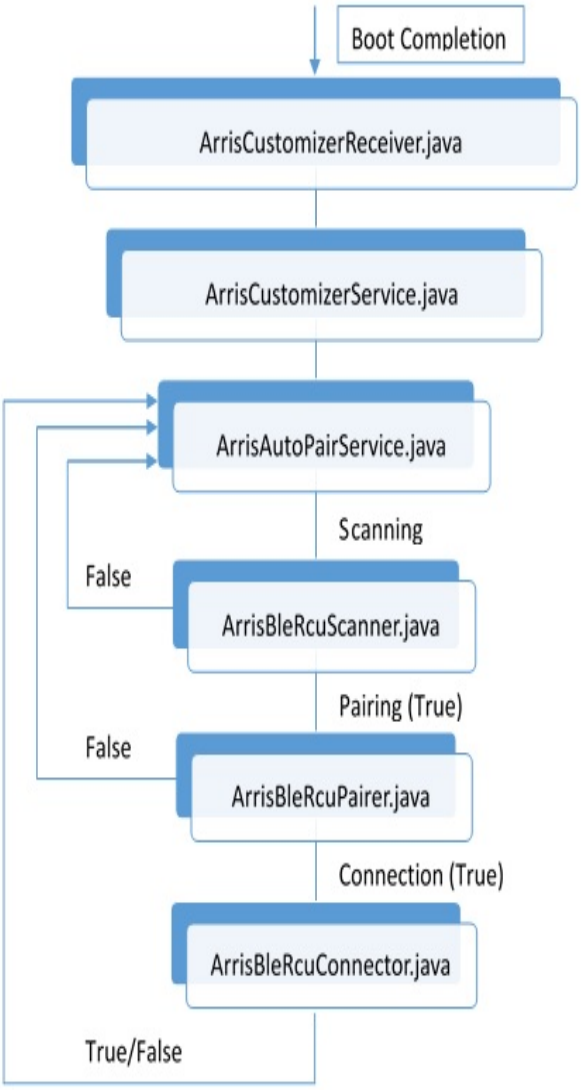

Figure 6.10: ArrisCustomizer Application

Chapter 7

Configuration of GPIOs wrt BLE RCU

An open point of discussion was left in the end of previos task that although the remote was connected, there was no sign on UI also there was no signal on STB that the remote is connected. So this task comes into consideration, starting with the signal on STB with remote connection, following end goal was designed for this task:

- The LED on STB should blink once remote and STB are in pairing mode.
- The LED should be in 'ON' state, once remote is connected to STB.
- The LED should be in 'OFF' state, once remote is disconnected from the STB.

This task started with following one JIRA and porting the setLed API from `DEV_barracuda` branch to `DEV_cory` branch. For porting the setLed API, setLed API was added to ArrisCustomizerReceiver.java, ArrisCustomizerService.java. Two new files were added to ArrisCustomizer App - ArrisLedType.java and ArrisLedColour.java. The jni layer which was initially deleted from the code was also restored back.

After figuring the flow of the code given in the JIRA, the next task was to figure out the pin mapping of the GPIOs in 72604 board. The pin mapping was figured out, and the changes were made and the final end goal was completed for this task.

The GPIO for 72604 which is used in this configuration of GPIOs was GPIO 48. Images were sent and demo was given and now pin mapping is being done on 7268 platform.

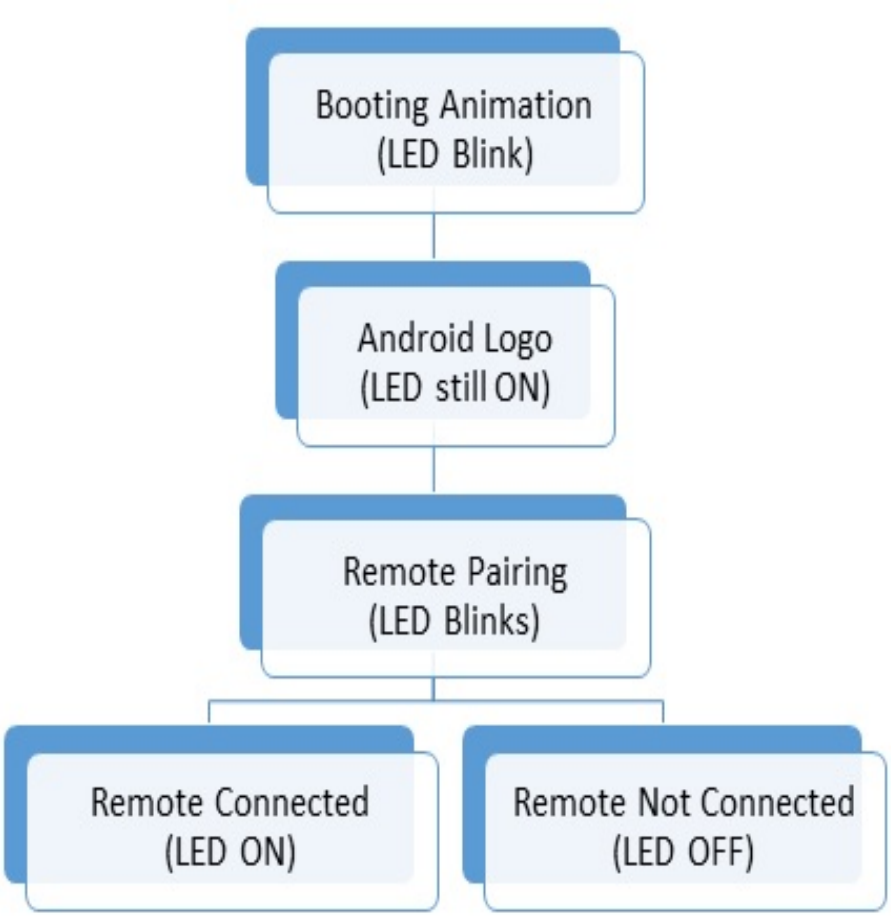

Figure 7.1: Flow diagram for GPIOs configuration

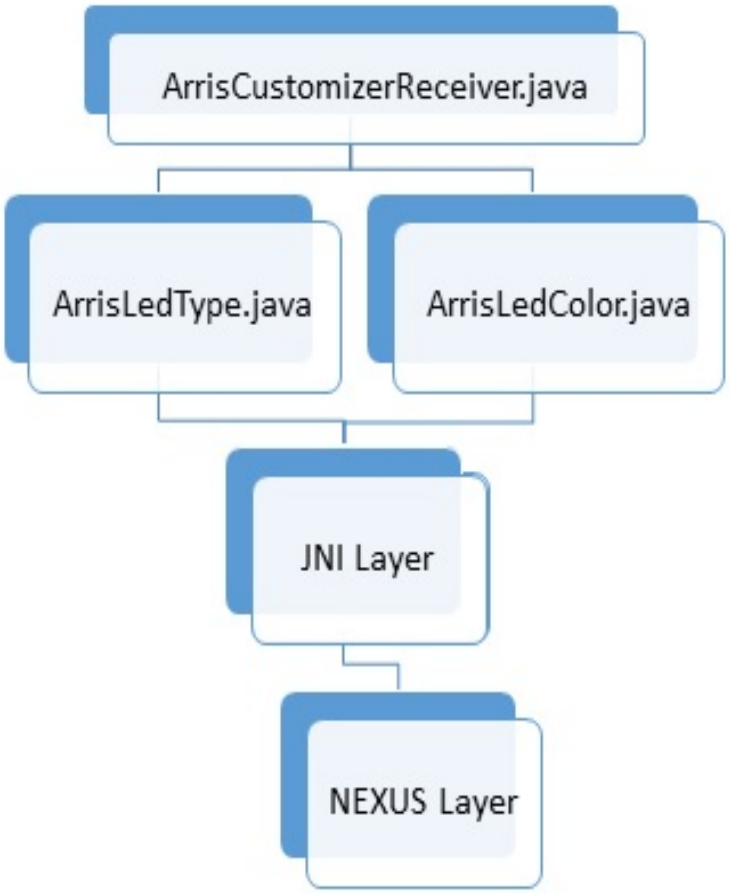

Figure 7.2: Changes done in ArrisCustomizer for GPIOs configuration

Chapter 8

Conclusion

8.1 Conclusion

The conclusion for this Task is that Auto-Paing app is working fine with both the remotes and with all the three platforms, also LED indication has also been added to application so that there is some indication on the platform side that the remote is connected, paired or disconnected.

Some future work is also needed in the direction that there should be some notification on the UI that the remotes has been connected or disconnected. Also for now, some of the remotes are working with some of the platforms only, although ideally both the remotes should work with all three platforms.

Bibliography

[1] "Installing Repo Tool." https://source.android.com/setup/build/downloading.

[2] "Jack server." https://android.googlesource.com/platform/prebuilts/sdk/+/master/tools/README-jack-server.md.