ULT tools and libraries for Intel platforms

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

Patel Nilaykumar Kantibhai
(13MECE12)

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

May 2015

ULT tools and libraries for Intel platforms

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

Patel Nilaykumar Kantibhai
(13MECE12)

Under the guidance of

External Project Guide:

Indranil Mukherjee
Engineering Manager,
Intel Technology India Pvt. Ltd.
Bangalore.

Internal Project Guide:

Prof.Sachin Gajjar
Assistant Professor,
EC Branch, EE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

May 2015

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgment has been made in the text to all other material used.

- Patel Nilaykumar Kantibhai

13MECE12

Disclaimer

"The content of this thesis does not represent the technology,opinions,beliefs, or positions of Intel Technology India Pvt. Ltd.,its employees,vendors, customers, or associates."

Certificate

This is to certify that the Major Project entitled **"ULT tools and libraries for Intel platforms"** submitted by **Patel Nilaykumar Kantibhai A.(13MECE12)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge,haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Prof. Sachin Gajjar
Internal Guide

Dr. N.P. Gajjar
Program Coordinator

Dr. D.K.Kothari
Section Head, EC

Dr. P.N.Tekwani
Head of EE Dept.

Dr. K. Kotecha
Director, IT

Certificate

This is to certify that the Major Project (Phase- II) entitled **"ULT tools and libraries for Intel platforms"** submitted by **Patel Nilaykumar K. (13MECE12)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.

Indranil Mukherjee
Engineering Manager,
Intel Technology India Pvt. Ltd.,
Bangalore

Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr. P.N.Tekwani**, Head of Electrical Engineering Department, and **Dr. N.P.Gajjar**, PG Coordinator of M.Tech Embedded Systems program for allowing me to undertake this thesis work and for his guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Prof.Sachin Gajjar**, guide of my major project for his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I would take this opportunity to express a deep sense of gratitude to **Indranil Mukherjee** and **Rakshmi Bhatia** , Project Managers, Intel Technology India Pvt. Ltd.. for their cordial support, constant supervision as well as for providing valuable information regarding the project and guidance, which helped me in completing this task through various stages.I would also thank to **Vandana Kannan** and **Sangani Suryanarayana**,my Project Mentors for always helping,give good suggestions and solving my doubts and guide me to complete my project in better way.

I am obliged to **R, Durgadoss** member of Android Display Driver Development team, Intel Technology India Pvt. Ltd. for the valuable information provided by him in his respective field. I am grateful for his cooperation during the period of my assignment.

Lastly, I thank almighty, my parents, brother, sisters and friends for their constant encouragement without which this assignment would not be possible.

- Patel Nilaykumar Kantibhai

13MECE12

Abstract

Unit level testing in driver development is one of the most crucial part. Through unit level testing, developer can know basic faults in his code. Because of this testing, developer can avoid small but disastrous mistakes. On the other hand, if this faulty (without doing unit level testing) code is passed for validation cycle, lots of man hours and resources are wasted. The other part of this project is doing register operations effectively. Register operations (like reading, writing and comparing) in driver development are small but most crucial parts. In lower level driver in linux, one of developers job is to configure specific hardware registers in particular way to make hardware working properly. On the other side, validation or debug engineer needs to check register status in order to validate or debug various scenarios. In some scenarios, engineer needs to check whole platform dump so that he can find root cause of problem or understand that whether driver is working in proper manner or not.

In this project, main focus is on how to design and implement unit level testing tools which can detect maximum faults in code before going to validation cycle. Two tools are designed to check whether BIOS is properly parsed and values are properly applied to registers. Backlight control is designed to check proper kernel programming for backlight and Split screen checks for flickers and proper FIFO buffer programming. For effective register operations, main focus is on how to do these operations through application layer so that user can have better experience. Also, register parser tool is developed in such a manner that it can be used on various hardware platforms without extra efforts. Using this tool, various operations like reading, writing, comparing can be performed by the user. Register dump (all-important register of platform) is one of the important feature of this tool which

gives debug information of all important registers of the platform. By this, user does not need to read what every bit does from platform specific document. This tool also gives facility of saving all the results so that they can be used for future reference or can be sent to code owner for corrections.

Contents

List of Figures

Abbreviations

SoC ..... System on chip
EDID ..... Extended display identification data
OpenGl ..... Open Graphics Library
SDK ..... Software Development Kit
IDE ..... Integrated Development Environment
XML ..... Extensible Markup Language
I/O ..... Input-Output
MMIO ..... Memory Mapped Input-Output
CPU ..... Central Processing Unit
ASIC ..... Application specific Integrated Circuits

Chapter 1

Introduction

1.1 Background

A driver developer, In Linux, whenever writes driver for a particular device, He writes functions in C to configure particular set of registers to make hardware work in particular way. During the design of driver, developer sometimes needs to check the value of register to verify his design. Register writing is also needed to configure hardware manually to test its functionality in various scenarios.Apart from developer engineers, validation and debug engineers also need to read and write registers to test various scenarios and find root cause of problem. For reading and writing registers, Engineers use command line tools which can read and write single registers at a time. Sometimes developer makes code changes or writes new codes which need initial level testing to find common but disastrous mistakes. No tools are available for such testing, developer is forced to do this testing manually.

1.2 Motivation

Command line tools can do only single operation at a time so user needs to type command for each operation. Also, these command line tools does not give any debug information regarding registers. Sometimes debug and validation engineer

need to save register values of various scenarios(in which driver is behaving normally) and send to owner of that code for corrections in driver.In some driver related problems, debugger need to take snippet of all registers of platform to understand problem which is not possible using command line tools. For code modifications, manual testing in time consuming and tedious task which has lead to development of unit level tests

1.3 Problem Statement

Main objective of this project is, to deliver library and tools support to developer, validation and debug engineer to ease their work and save resources. Following are the parts of this tool.

- Developing C shared library which contains functions for reading and writing single register and register dump.
- Android tool which calls these files and save results
- Write shell script to install application and give required permissions
- Write python script to parse valid register addresses
- Developing unit level testing tools for early validation of code.

1.4 Thesis Outline

The rest of the thesis is organized is as follow :Chapter 2 gives details regarding architecture of Android operating system. It explains about basic building blocks of developing android application like activity, content, service, manifest.xml etc. Further in this chapter sysfs file system is explained. In chapter 3, all the C functions which are included in shared library are explained with flowcharts. Last section of this chapter elaborates scripts written in Python and Shell. In chapter 4, all the

tools including ULT and register parser are elaborated with help of flow charts. At last, chapter 5 and chapter 6, concludes the thesis and gives idea about future work.

Chapter 2

Literature Survey

2.1 Android OS

In the figure 2.1 the diagram of Android operating system architecture is shown. The Android operating system is a software stack of various layers, where each layer is a group of various program components. All together it does consist operating system, middleware and applications. Each layer in the architecture provides different services to the layer just above it. In next sections, the features of each layer are explained in detail.[1]

2.1.1 Linux Kernel

The first layer is the linux kernel. The whole Android operating system is developed on top of the linux kernel with some extra changes introduced by Google. This linux kernel talks with the hardware and have all the required hardware drivers. Drivers are programs that are written in C which control and communicate with the underlying hardware. For example, consider the display. All devices has a display hardware in it. Therefore the kernel must include a display driver to handle display hardware. This layer also plays role as an abstraction layer between other software layers and hardware.

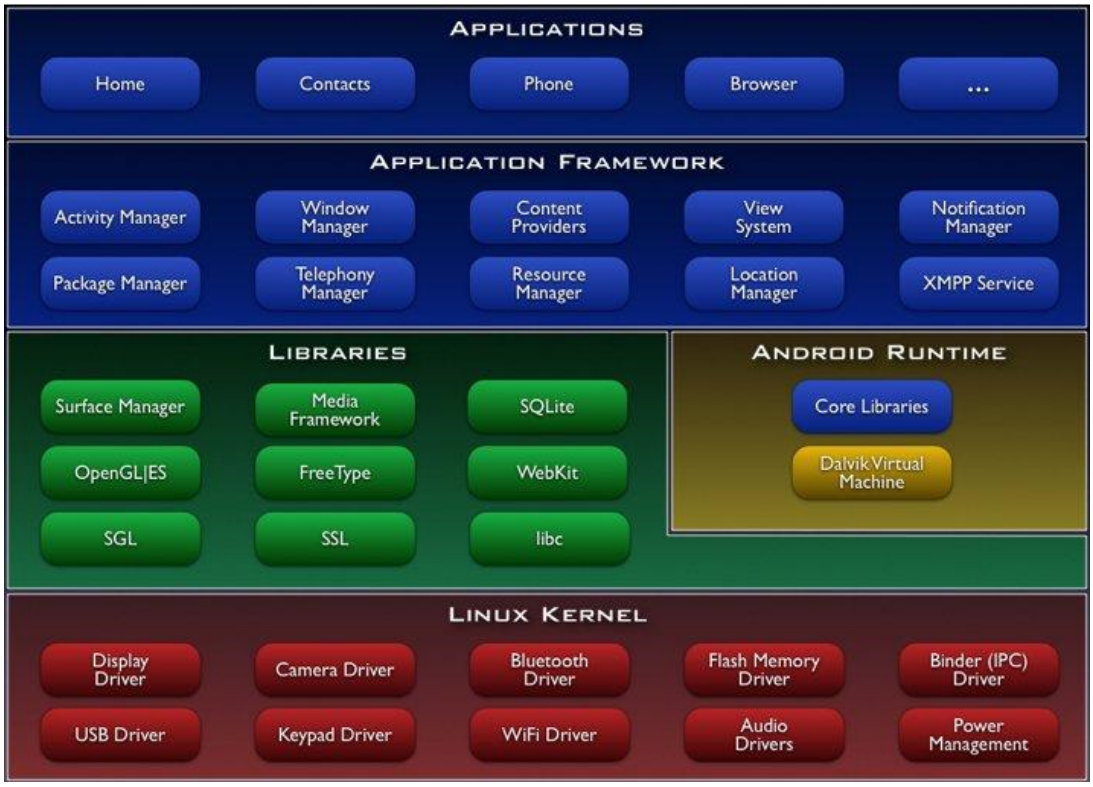

Figure 2.1: Android OS Architecture

All the core functionality of Android, like memory management, networking, security settings, process management etc. uses Linux kernel. Because the Android is developed on a proven and popular foundation, it is easy to the port of Android to various hardware [1][2].

2.1.2 Libraries

In a next level there are bunch of native libraries developed in C or C++, which offer consistent performance for number of components. For instance, surface Manager is mainly responsible for combining various drawing surfaces on a display. It is responsible for the access for various processes to compose 3D and 2D graphics layers. OpenGL designs core of various graphic libraries and are being used accordingly for 2D and 3D hardware acceleration. Also, it is very much possible to use 3D and 2D graphics in the single application in Android. A PacketVideo provides the media

framework. It offers libraries for a recording and playing support for all of the media and images. FreeType libraries are useful for rendering the bitmap and vector font. For data maintainance and storage, SQLite is used in Android. As mentioned earlier, it is very light rational management system, which does locate a file for all of the operations connected to database. WebKit, the browser is being used by Apple Safari, was modified by Android to fit better in a small size screens[1][3][7].

2.1.3 Android Runtime

At the same level Android runtime is there, where the important component dalvik virtual Machine resides. It was designed specially for Android running in resource constrained environment, where the limited CPU, battery, data storage and memory are major issues. Android does offer an integrated tool named as dx, which translates generated binary code to .dex form file .jar, after this binary code does become much more efficient to be executed on small processors.

As a result, it becomes possible to have various instances of dalvik VM which are executingng on a single device at a time. The Core libraries are designed in Java and does have the collection of the utilitie, IO, classess and other tools.Figue 2.2 shows the conversion of .dex to .java[2][3].

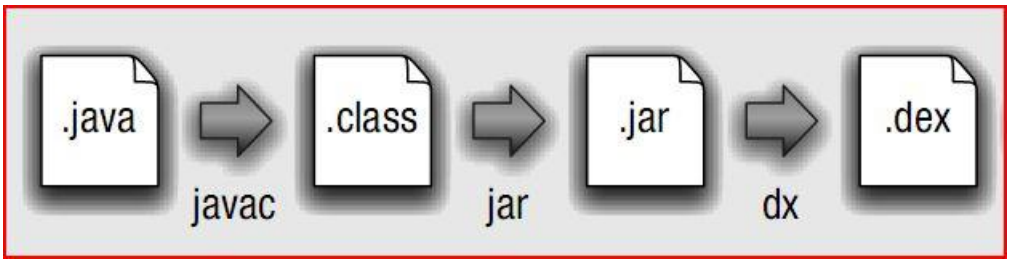

Figure 2.2: Conversion from .java to .dex file

2.1.4 Application Framework

As a very next, there is an application framework, developed in Java. It is the toolkit which is used by all of the applications, ones which are shipped with mobile device like SMS or contact box, or applications developed by the Android developer or Google. It has various components like content providers, Package manager, activity manager.

The Activity manager is responsible for managing the life cycle of the applications and gives a common platform of navigation backstack for all the applications, which are being executed in various processes. A Package Manager maintains track of all of the applications, which user has installed in device. A Windows Manager is Java programming language abstraction on top of lower level services which are offered by the Surface Manager.

Content Providers were built for Android operating system to share the data with another applications, for example, the contacts of device in address book can be exported in other application too. A Resource Manager is responsible for saving localized bitmaps, layout file, strings descriptions and other parts of an application. A View System generates a set of lists and buttons that can be used in UI. Other component like Notification manager is useful for customizing display alerts and various other functions.[4]

2.1.5 Application Layer

Top layer of Android architecture consists all the applications, which are used by the end user. By installing various different applications, user can make his device optimized, unique and smart phone. All applications are developed in the Java programming language[1].

2.2 Developing Android

2.2.1 Application Building Blocks

Android application can be considered as collection of various components, of different kinds. These components are mostly quite loosely coupled, to a degree where one can precisely describe them as the bunch of elements rather than an isolated cohesive application. In general, these elements execute in a single system process. It is possible (and very common) to invoke multiple threads in that same process, and it is also possible to invoke completely different child process if one need to. Such possibilities are very uncommon though, as Android tries pretty hard to make various processes transparent to the code[4].

Google gives three different versions of the SDK for Mac OSX, for windows and one for the Linux flavoured os. The developer can make use of Android plugin in Eclipse IDE or other IDE like as intelliJ. the very First step for the developer is to segregate the particular application into the various components, which can be supported by particular platform. The main building blocks are:

- Activity
- Intent Receiver
- Service
- Content Provider

Activity: It is user interface component and corresponds to single screen at a time. It means that for a simple application like contacts, the developer needs to have one particular activity for displaying information about chosen contact, another activity component for showing contact details of selected name and etc [4][5].

Intent Receiver: It invokes a predefined action via an external event. As an example,for an application such as Email Inbox, a developer needs to have an intent

receiver and have to register his code via XML to invoke the alarm notification, when there is an email in inbox of user [4][5].

Service: It is a task, which is being executed in background. It does mean the end user can invoke the application from an activity window and can keep a service working, while browsing through any other application. For example, he can browse Google browser application while listening music or holding the call. [4][5].

Content Provider: It is a component, which can allow user to share some of the data with an other applications and processes. It is a best way for the applications to talk with each other.Android generally, is shipped with the bunch of applications like an email, calender, SMS, alarm, browser, contacts, maps and others. Java is used to develope all the applications [4][5].

2.2.2 Android Manifest File

The AndroidManifest.xml file is a control file which informs the system what is to be done with every top-level components (specifically services, activities, content providers and intent receivers as explained below) one has created. For example, this is a "glue" which tells which Intents of your Activities are going to receive. The developer need to define and make list of every component, which he does want to use in the particular AndroidManifest.xml file. It is the must file for all of the applications and can be found in a root folder. It can also be possible to specify every global value for particular package, all the components and their classes used, intent filter, which tell when and where the certain activity is supposed to start, permissions and instrumentation like testing and security concerns. AndroidManifest.xml file is described in following example[6]:

1. $\langle ?xmlversion = "1.0" encoding = "utf - 8" ?\rangle$
2. $\langle manifestxmlns : android = "http : //schemas.android.com/apk/res/android"\rangle$
3. $\langle package = "dk.mdev.android.hello"\rangle$

4. $\langle applicationandroid : icon = "@drawable/icon" \rangle$
5. $\langle activityclass = ".HelloAndroid" android : label = "@string/app\_name" \rangle$
6. $\langle intent - filter \rangle$
7. $\langle actionandroid : value = "android.intent.action.MAIN" / \rangle$
8. $\langle categoryandroid : value = "android.intent.category.LAUNCHER" / \rangle$
9. $\langle / intent - filter \rangle$
10. $\langle / activity \rangle$
11. $\langle / application \rangle$
12. $\langle / manifest \rangle$

The line 2 is the declaration of name space, which does make a standard Android attributes visible for that application. In the line number 4 there is one ¡application¿ element, where the developer can specify all of the application level components and their properties that are being used by that package. Activity class in a line 5 tells the initial screen which is visible to the user and it may have more than one ¡intent-filter¿ elements which describes the actions that particular activity supports [3] [6].

2.2.3 Application Lifecycle

In Android, every application is executed in their own method, which supplies higher performance protected memory, in security and alternative advantages. So humanoid is accountable to execute and stop working properly those processes once it's required. It's vital that application developers perceive however completely separate application elements (in explicit BroadcastReceiver, Activity and Service) impact the lifespan of the application's method. Not victimization these elements properly may result within the system killing the application's method whereas it's

doing vital work. to work out that processes ought to be killed once low on memory, humanoid places every method into AN "importance hierarchy" supported the elements running in them and also the state of these elements. These method sorts ar (in order of importance).

1. A foreground method is one that's needed for what the user is presently doing. numerous application parts will cause its containing method to be thought of foreground in several ways. A method is taken into account to be within the foreground if any of the subsequent conditions hold:

   - It is running associate Activity at the highest of the screen that the user is interacting with (its Resume() process has been called).
   - It has a BroadcastReceiver that's presently running (its BroadcastReceiver.onReceive() instance is executing).
   - It has a Service that is currently running code in one of its own callbacks (Service.onCreate(), Service.onStart(), or Service.onDestroy()).

   There will solely ever be a couple of such processes within the system, and these can solely be killed as a final resort if memory is thus low that not even these processes will still run. Generally, at now, the device has reached a memory paging state, thus this action is needed so as to stay the programme responsive.

2. A visible method is one holding an Activity that's visible to the user on-screen however not within the foreground (its onPause() methodology has been called). this could occur, as an example, if the foreground Activity is displayed as a dialog that permits the previous Activity to be seen behind it. Such a method is taken into account very vital and can not be killed unless doing thus is needed to stay all foreground processes running.

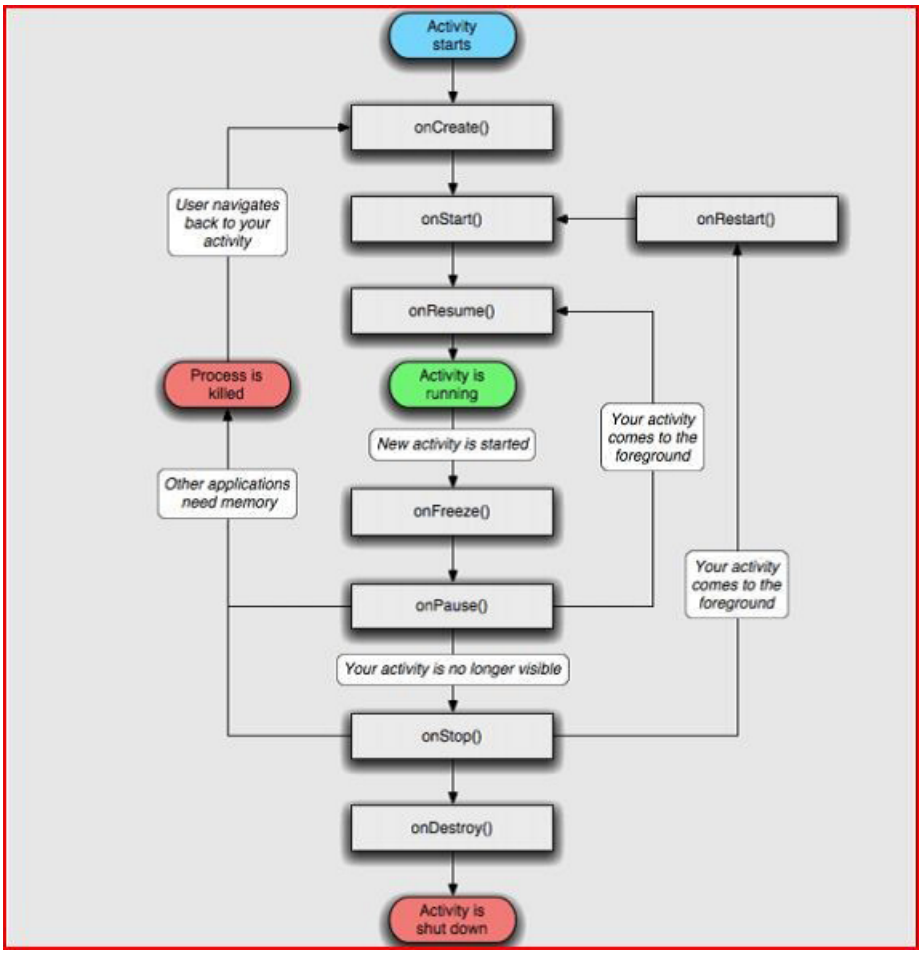

Figure 2.3: Activity lifecycle of Android application

3. A service method is one holding service that has been invoked with the start-Service() technique. tho' these processes don't seem to be directly visible to the end user, they're usually doing things that end user cares regarding (such as background network information transfer or download), that the system can invariably keep such methodes running unless there's not enough memory to retain all of foreground and visual process.

4. Background method is the one which is holding an Activity that's not presently visible to the user (its onStop() technique is been called). These processes don't have any direct impact on end user expertise. Provided they design their Activity life-cycle properly, the system will kill such processes at any time to reclaim

memory for one amongst the 3 previous processes varieties.

Usually there are several of those processes running, in order that they are generally placed in LRU list to make sure the method that was last seen by the user is that the last to be killed once running low on memory.

5. An empty method is one that does not hold any application program parts. the sole reason to stay such a method around is as a cache to enhance startup time following time a element of its application has to run. As such, the system can usually kill these processes so as to balance overall system resources between these empty cached processes and therefore the underlying kernel caches[4][5].

2.3 sysfs file system

sysfs could be a approach for presenting kernel objects, their relationships with each other and their attributes. It gives 2 components: kernel programming interface for mercantilism these things via sysfs, and user interfaces to look at and change these things that are mapped back to kernel objects that they present. sysfs is just a group of files, symbolic links and directories. The that means of every scheme in sysfs and their contents is as following:

2.3.1 Block directory

The block directory holds subdirectories for every block device that can be found within the system. In every block devices directory, there are some attributes that describe several things, as well as the size of a device and the dev_t variety. there's a symbolic link which indicates the physical device which the block device indicates to (in the physical device tree, which is described later in this chapter). Also, there's a directory which expose interface to an I/O scheduler[8].

2.3.2 Bus directory

The bus directory holds sub-directories for every physical bus kind that has support registered within the kernel. every bus kind that can be represented has 2 sub-directories: drivers and devices. The devices directory holds a listing of each device found thereon variety of bus within the whole system. The devices found are literally symbolic links which time to the devices directory within a global device tree. The drivers directory holds directories for every utility that has been registered with the bus kind. at intervals every of the drivers directories are attributes that permit viewing and manipulation of driver parameters, and symbolic links that direct to the physical devices (in the device tree) that the driver is bound to [8][8].

2.3.3 Class directory

Class directory holds representations of each device class which is registered with a kernel. Sub directories are contained by every device for every class object which are registered and alloted therewith device category. The directories of every class device objects hold symbolic links to driver directories and device (in the global device hierarchy and therefore the bus hierarchy accordingly) that are related to particular class object[9].

2.3.4 Devices directory

The global device hierarchy is held by device directory. Device directory contains every physical device which is found by the bus types registered with kernel. They are represented in ancestrally correct way. Every device is represented as a subordinate device of device for which it is physically subordinate .

Mainy, two type of devices are exceptions to this kind of representation: system devices and Platform devices. Platform devices are mainly peripheral devices which are inherent to the specific platform. Generally they have MMIO, some I/O ports which resides at a fixed location that is known. For instance, legacy X86 devices such

as the embedded devices of a SoC solution, a floppy controller or a serial controller. system devices, in various ways, are different than any other device. System devices are likely to have hardware register read-write accesses for configuring system but they are not eligible to transfer data.Generally, system devices are not likely to have drivers that are bound them.Though, at least for them who are represented via sysfs, are likely to have architecture-specific code which configures them, treats them as the objects to whom they can export. Examples of system devices are ASICs, timers and CPUs[9].

Chapter 3

Library and scripts Design and Implementation

Mainly two libraries are implemented, one is linux shared library and the other is Java native interface shared library(used in register parser tool). There are mainly four C functions that are included in these libraries. while using run time it is compiled and pushed on the device to basic perform operations like checking platform type, reading register, writing register.

- Devid : Checks on which platform tool is running.
- Read: Reads register value.
- Write: Writes to register.
- Dumper: Reads Essential registers of whole platform and parse them to get debug information.

3.1 Device Id

1. As shown in flow chart, first through get_pci_device() various functions from shared library libpciaccess are called to get device related data in pci_device

structure.

2. In Second step, condition is placed to check whether device is x86 based or not.

3. If it is, its pci_device-⟩ devid will be checked for platform. According to platform, particular string will be returned to java layer to further process.

4. If pci_device-⟩ devid does not match with any known platform, other will be sent. On reception of other java will invoke one default routine.

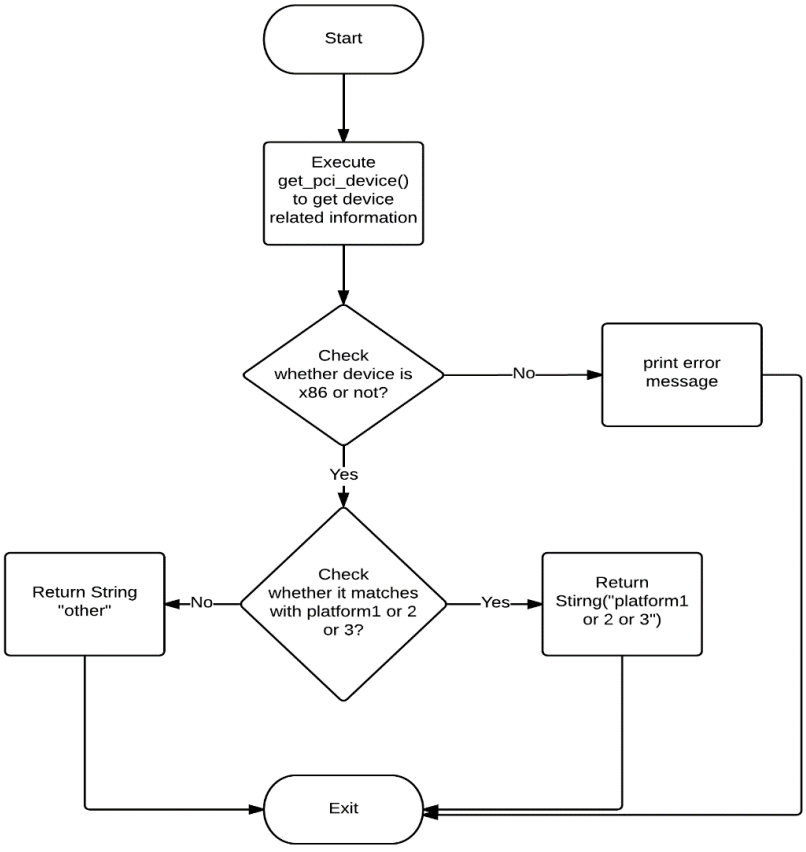

Figure 3.1: Device ID

3.2 Register reading

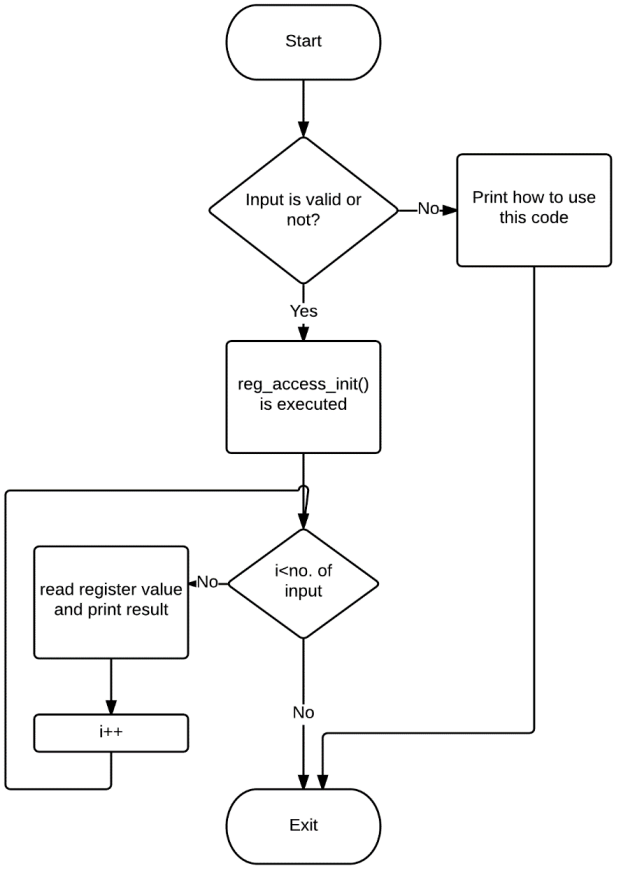

Figure 3.2: Register Read

1. In first step, It is checked that whether valid set of inputs are passed or not. If not, One function will be executed which tells how to use.

2. In next step, reg_access_init() is called which in turn calls several intel library functions to initialize platform specific environment to read registers. These functions are as follows

    - Get_mmio: According to device id, it will execute pci_device_map_range from libpciaccess. On successful execution, it will return 0 else error number.
    - Get_register_map : It will give base address of platform from which reading can be started.

3. Now one for lop is invoked and it will take register offsets one by one and read their value and prints result on console (From where java layer can read).

3.3 Register Write

1. In first step, It is checked that whether valid set of inputs are passed or not. If not, One function will be executed which tells how to use.

2. In next step, reg_access_init() is called which in turn calls several intel library functions to initialize platform specific environment to read registers. These functions are as follows

    - Get_mmio: According to device id, it will execute pci_device_map_range from libpciaccess. On successful execution, it will return 0 else error number.
    - Get_register_map : It will give top address of platform from which reading can be started.

3. One pointer is initialized which will point to address passed by user. The data at that address is read and thrown on shell.

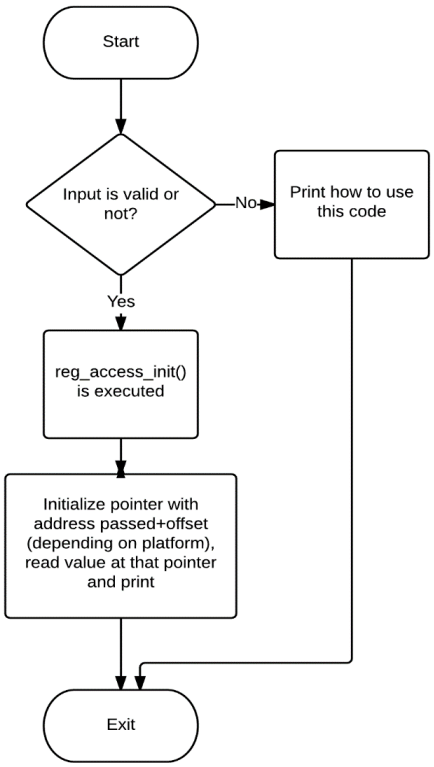

Figure 3.3: Register Write

3.4 Register Dump

1. Through get_pci_device various functions from shared library libpciaccess are called to get device related data in pci_device structure.
2. In next step, reg_access_init is called.
3. Various if conditions will check which type of platform in running this code. According to platform array of structure will be passed to a function which takes every register one by one and read register value.
4. Next, register value is parsed and debug information will be stored in a string.
5. The result will be printed on console from where java layer can read.

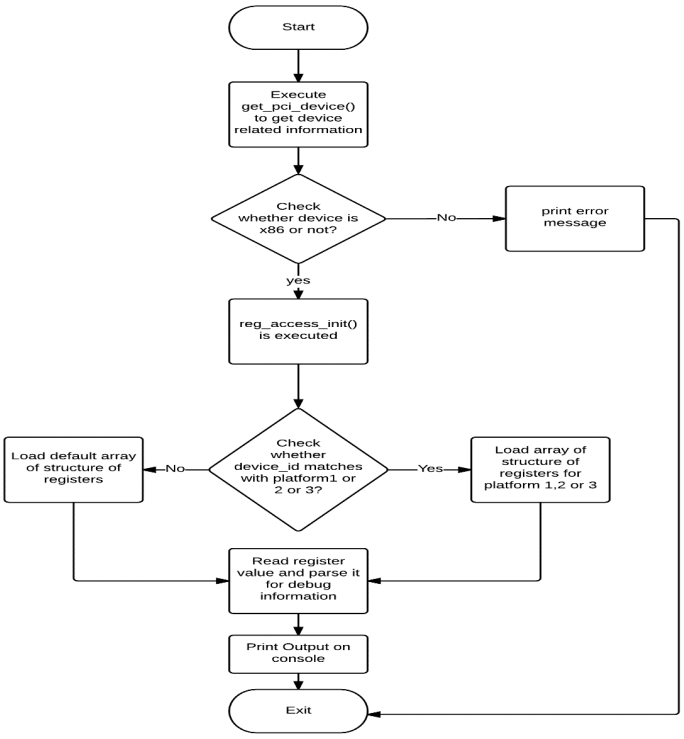

Figure 3.4: Register Dump

3.5 Shell and Python Scripts

A Shell script is written to install android tool on device and give necessary permissions for sysfs entry (to where kernel throws register value asked by tool). It also changes permission to 777 of reg_read, reg_write, reg_dump so that tool directly can execute them from user space.
Python Script is used to get valid set of registers from a word file of specific platform. Flow of Script is as following.

1. First word file is opened in read mode. We read each line of file one by one in for loop.

2. Line is split in array of words by split() method in python. Whole array is checked for words like Address offset, memory offset, I/O offset and if found copy next word which is likely to be address in an array.

3. The resultant array is converted to hexadecimal from string and arranged in ascending order.

4. The properly arranged array is again converted to string so that whole string of array is called by java.

Chapter 4

Tools Design and Implementation

4.1 Register parser

The C files explained before runs at back end by invoking process from java layer. The output generated by that files are printed on console form where java layer takes that output in input buffer. Further process is done by java layer on that buffer. The figure 4.1 explains 2 buttons (Add and Show).

4.1.1 Add button implementation

Using this button user can enter various register addresses to see their corresponding values at a time. Following is flow of actions on clicking this button.

1. On click of this button first, it will be checked whether there is any value in the edit text (Enter Register). Here user is allowed to enter up to 8 characters only.

2. If value is found, it will be taken in form of string and appended in Array list. If not found, No action will be taken.

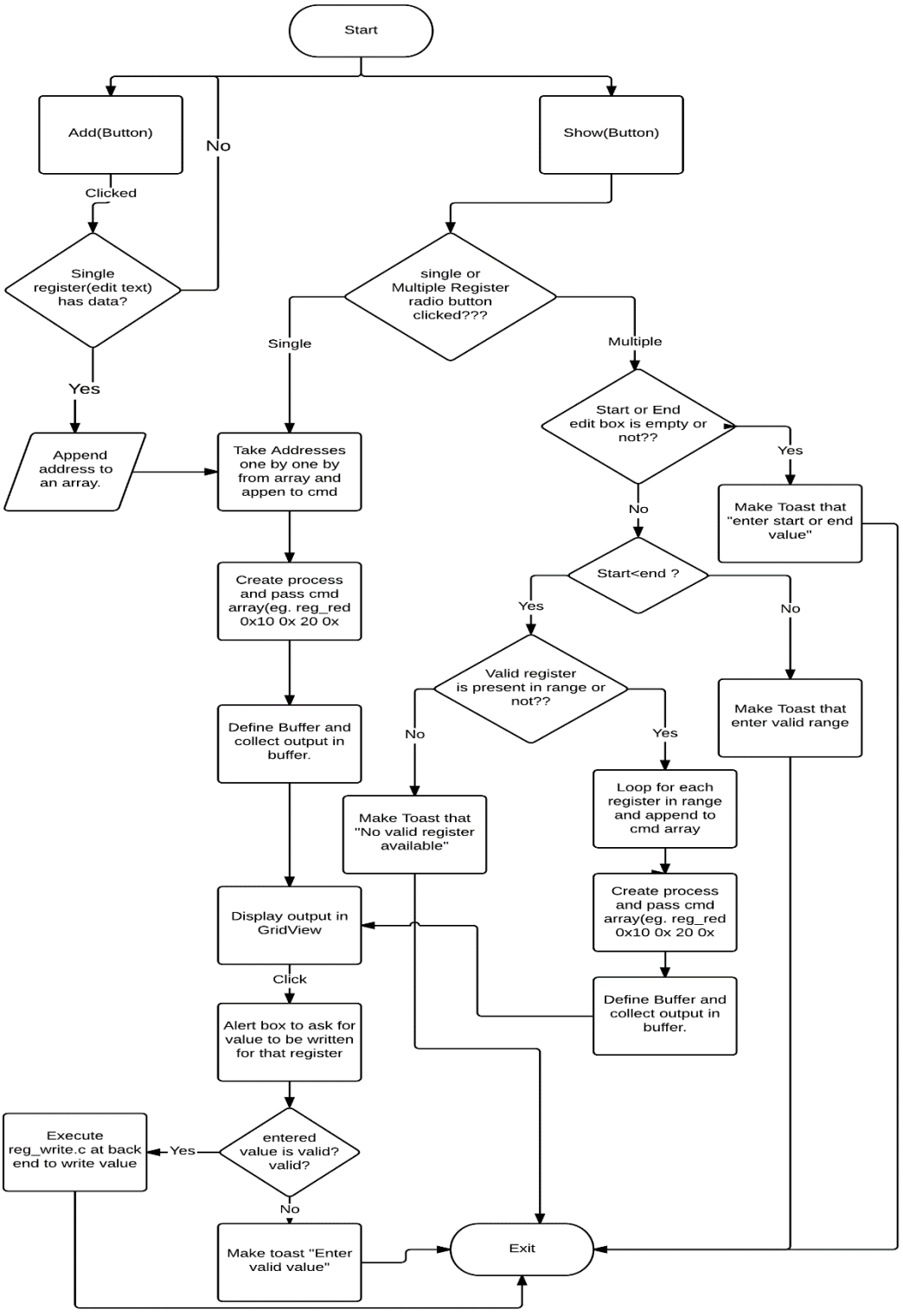

Figure 4.1: Add and Show Button

4.1.2 Show button implementation

This button is used to display register values in Grid View. Following is working flow of this button.

1. First, It is checked whether user wants to read register for multiple continuous or discrete.

2. If single register read is clicked then it will take elements from array list and append to a cmd (e.g. reg_read 0x10 0x20 0x30 .) and make one String to pass process.

3. If user selects for multiple register, first it is checked that start and end register values are written or not. If not, Make toast that enter start or end value.

4. Next conditions is that whether start value is less then end or not. If yes, it is checked that there is any valid registers in that range or not. If not, make toast to display message.

5. All the valid registers are appended to cmd string to pass reg_read.c.

6. One process is created and the above command is passed to process to execute and output is collected in one buffer.

7. The output will be displayed in GridView. User can click any output from GridView to write that register. When user enters valid value and click on write, in back end reg_write.c is invoked and writes to that register.

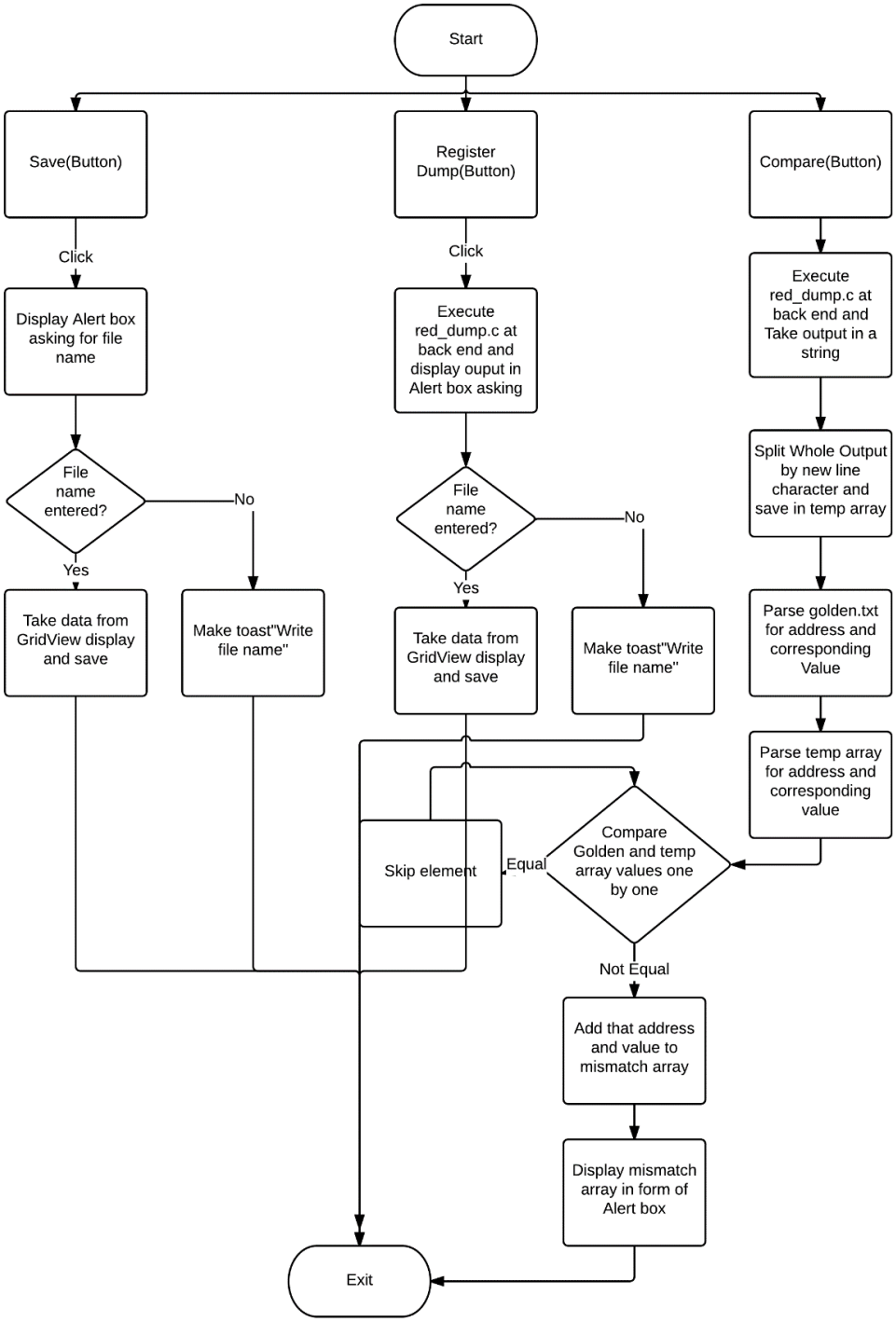

Figure 4.2: Save,Register Dump and Compare Buttons

Figure 4.2 Shows remaining three buttons(Save, Register Dump and Compare). Following is explanation for each button.

4.1.3 Save button implementation

Sometimes Debug or validation engineer needs to save register value to send Developer for possible fault in code. Following is the working procedure of this button.

1. When clicked on this button, an alert box in displayed for giving file name to which displayed data can be saved.

2. If file name is give, data is saved in filename.txt at location /data /data /com.example /register_parser. User can pull file from device by adb pull command.

4.1.4 Register Dump button implementation

1. On clicking this button reg_dumper.c will be executed and output will be printed on console. From console, java layer will get that data and display in form of Alert box.

2. In Alert box option is provided for user to save register dump.

4.1.5 Compare button implementation

This button compare register dump with already saved register dump. Following is working procedure of this button

1. In back end reg_dumper.c will be executed and output is taken in java layer.

2. Whole dump will be saved in form of array by splitting it from new line character. Same will be done for golden.txt.

3. Now for same address, data is compared and if data is differ that value and debug information is stored in separate array.

4. The mismatched values will be displayed as output in form of alert box.

4.2 MIPI BIOS Configurations Checker

At the time of device boot up, When BIOS hands over to Android Operating system, Display driver takes some of the configurations from bios like color format, device reset timer. These taken configurations are then programmed in display registers. The aim of these tools is to parse bios binary and cross check that whether registers are programmed correctly or not. Following is the algorithm of this tool.

1. First step is to copy bios from the memory location at which it is saved to the location where we can save and manipulate it easily.
2. Next, it is checked that whether the file which is read is not corrupt and contains Intel's BIOS signature.
3. MIPI Panel Id is abstracted from a panel specification block and saved in panel id.
4. A pointer is defined which points header of required block (here configuration block).
5. Size of total block is taken from size block.
6. According to Panel id, pointer will be over written to point data in block.
7. Now Data is read and saved in packed structure for future comparisons.
8. Registers are read through shared library (explained in Library section) and compared to data in structure. Failure log is printed on screen and failure counter increases on each failure.
9. At last, total number of failures are returned.

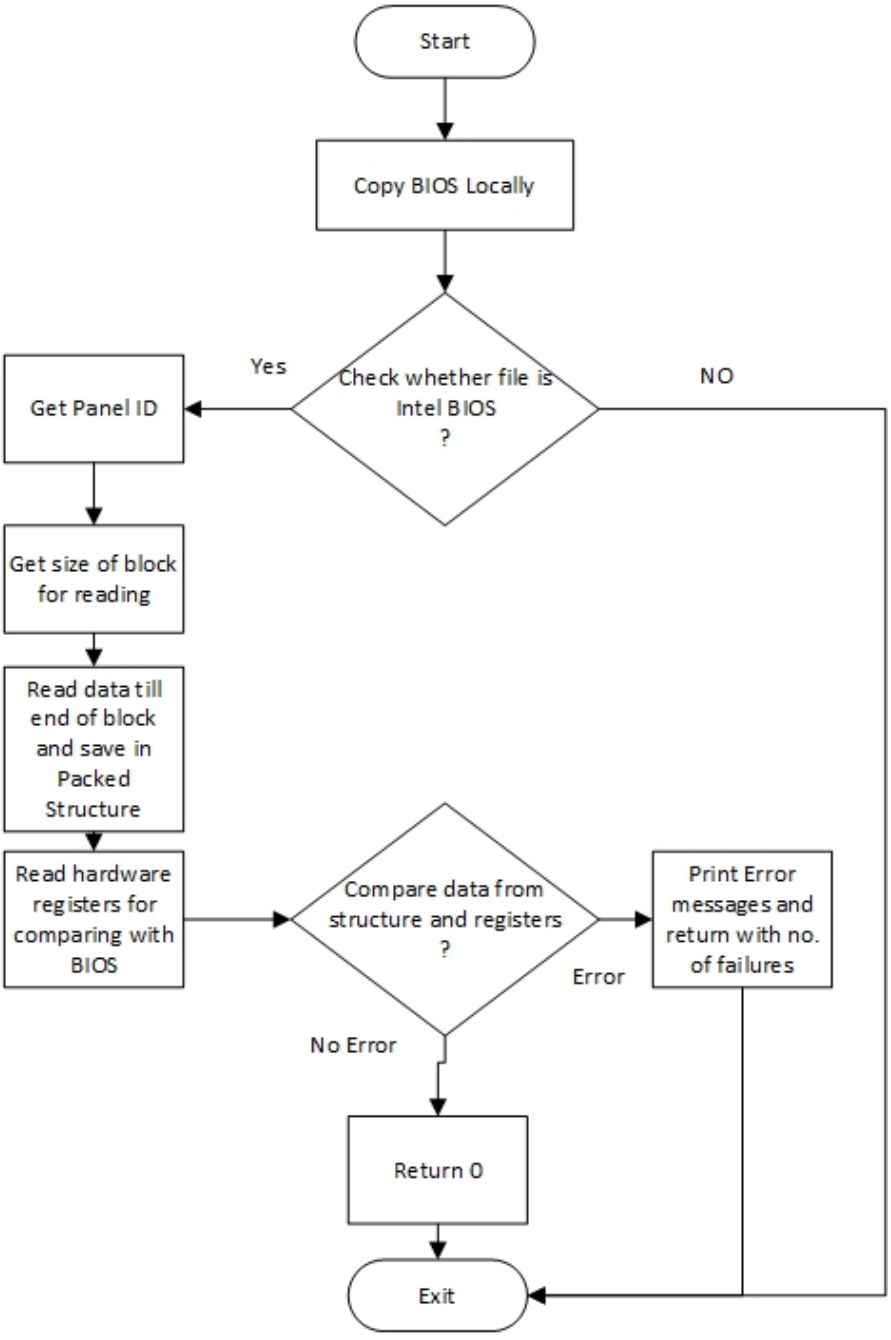

Figure 4.3: MIPI BIOS configuration checker

4.3 MIPI BIOS Sequence block Checker

Once device is booted up, to enable MIPI panel, based on panel (whether it is video mode panel or command mode panel), some command sequences are needed to be executed in particular order. These commands are written in one of the BIOS block. The aim of these tools is to parse bios binary to verify whether correct sequences are being followed or not. Following is the algorithm of this tool.

1. First 6 steps are as explained in MIPI BIOS Configuration Checker part of these Chapter.
2. In every sequence first byte indicates the type of sequence.
3. Next step is to read sequence in form of 1 byte till two consecutive 0s.
4. If 2 consecutive 0s are detected then from next byte onwards next sequence starts.
5. All the sequences are stored in form of array and then checked whether mandatory sequences for particular panel are there in the array in proper order or not.
6. If mandatory sequences are found in array then test results 0 and if not then returns -1.

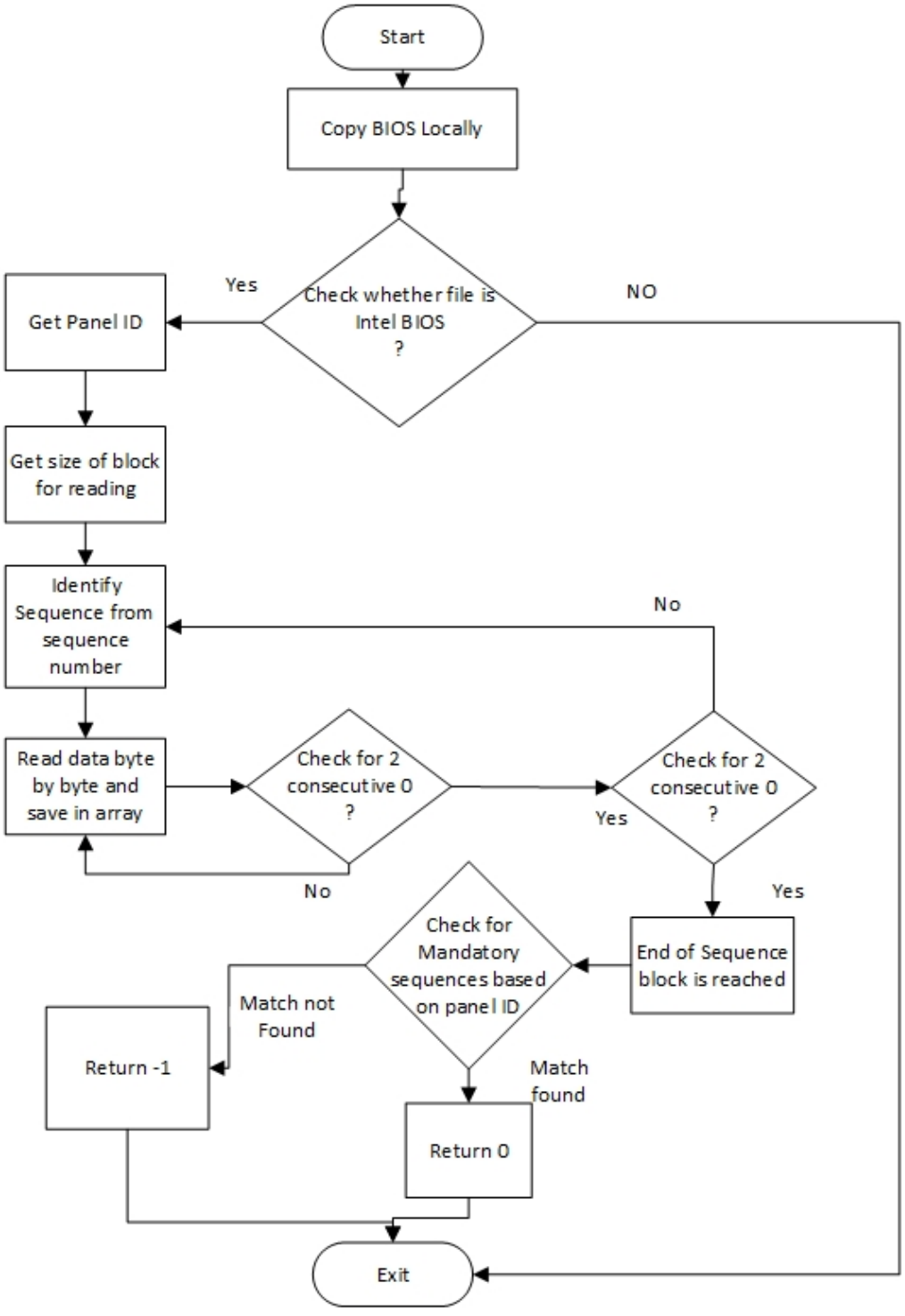

Figure 4.4: MIPI BIOS sequence checker

4.4 Backlight Control tool

Whether backlight is working properly or not is being tested with tool. Following is the algorithm to implement this tool.

1. First, Device under test is set for not to sleep for 30 minutes through various Android system calls which includes touch events, opening application, selecting an option among various options in menu.

2. First condition to be checked is whether display is set to some minimum level of backlight or not. If not, message is printed and tools returns -1.

3. Next, backlight is set through writing sysfs entry from 0 to 255 in increasing order with some predefined dely. After writing each value in sysfs, some hardware register are read to check whether written value are getting affected.

4. If for any value kernel is not programming backlight properly, error message will be printed with written value and expected values of registers.

5. Once reached to maximum level, values will start to reduce back to 0 and same procedure will be followed for assurance of perfect kernel programming.

6. If no error is detected then backlight will be set with the value at staring of execution and exit tool exists with 0 status.

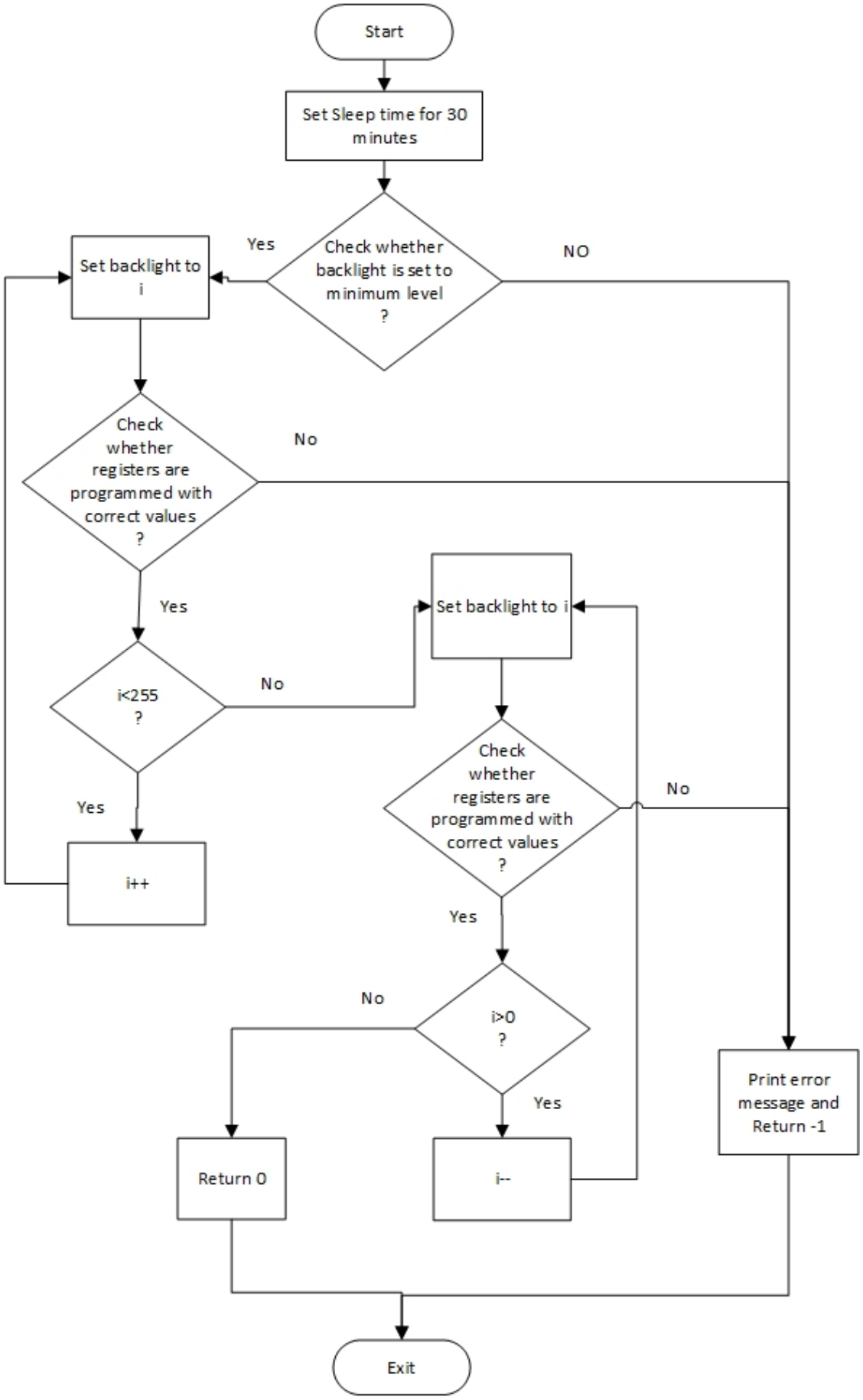

Figure 4.5: Backlight control

4.5 Split Screen Detection tool

At hardware layer, frames to be displayed are sent in FIFO buffer. From this buffer the frame is taken and displayed. If buffer is not properly emptied before the data is taken, the frame displayed will contain previous frame's data. This scenario in turn creates split screen. This tool detects split screen by two different ways. It continuously checks CRC value of red, green and blue elements of frame and compares these CRCs with golden values. Another way is it continuously checks hardware register for overflow. If any of these condition is detected, error status gets generated.

1. First, some binaries are pushed on the device for basic register operations like reading and writing registers.

2. Now as we need to compare current CRC with golden CRC, push an image to Gallery whose CRC is known.

3. Device under test is set for not to sleep for 30 minutes through various Android system calls which includes touch events, opening activity, selecting an option among various options in menu.

4. Disable status and navigation bar from display so that when we calculate CRC of screen they do not add their CRCs.

5. Launch Gallery application and open the pushed image through commands.

6. Next, enable the CRC calculations at hardware layer through writing 1 in particular register and clear overflow register bit just to make sure that it does not give wrong detection.

7. Now start loop for 0 to 200 comparing CRC and overflow bit for split screen detection. If both condition are detected then print status of CRCs and overflow register and exit with return value -1.

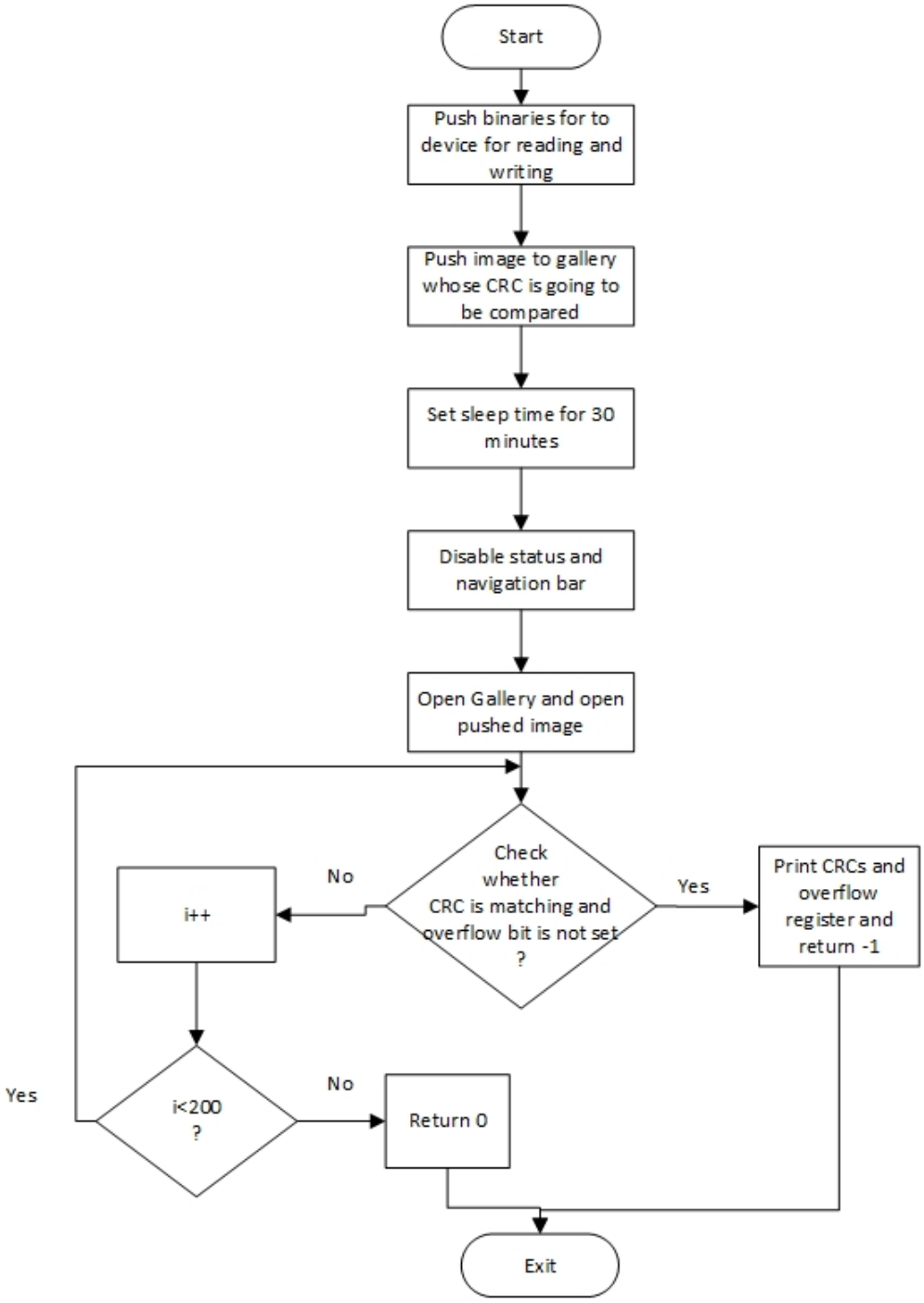

Figure 4.6: Split Screen Detection

Chapter 5

Conclusion

Unit level testing is perhaps one of the most important part of checking driver code before submitting it for validation cycle. After submitting this tools, developer had caught number of issues with code before submitting it for validation. This in turn, has saved considerable resources.

On the other side, for libraries, the purpose of making them is to give user better experience for this kind of operations. Apart from basic operations like reading and writing single registers, these libraries can read simultaneously number of registers. One advantage of using these libraries for validation and debug engineer is that they can save the data and send it to owner of code so that he can identify problem easily and fix it. Also, These libraries not only gives all the important registers of platform in single click but also it parses that registers and by this user is spared from peeping in whole platform specific document for registers.

Chapter 6

Future Work

There are number of possibilities for expanding unit level testing. Some possibilities are as follows.

- Tests based on panel specs can be designed.
- The major challenge and future focus area is to make these tools to run without human intervention.
- Scenario based tests can be added.

For libraries, there are various possibilities for expansion based on requirements. Following are the expansion area for this tool.

- Various functions can be added to check sprite plane, to check edid, etc.
- Functions implementation can be made in a manner that these libraries can become platform independent.
- Libraries can be made for other os platforms like linux, windows, etc.

References

[1] "Introduction to Android" [Online]. Available: http://developer.android.com/guide/index.html.

[2] "ART and Dalvik" [Online]. Available: https://source.android.com/devices/tech/dalvik/index.html.

[3] "Android A to Z: What is Dalvik" January 2013 [Online]. Available: http://www.androidcentral.com/android-z-what-dalvik.

[4] "Application Fundamentals" [Online]. Available: http://developer.android.com/guide/components/fundamentals.html

[5] "Maven: The Complete Reference" [Online]. Available: http://books.sonatype.com/mvnref-book/reference/android-dev.html

[6] "The AndroidManifest.xml File" [Online]. Available: http://lyle.smu.edu/ coyle/cse7392mobile/handouts/s01.The

[7] "Run native executable in Android" July 2014 [Online]. Available: http://gimite.net/en/index.php

[8] "sysfs -_The_ filesystem for exporting kernel objects" August 2011 [Online]. Available: https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt

[9] Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman, Linux device drivers. Third edition, o'reilly, 2005.