Software Device Driver Development & Automation of Test Framework

Major Project Report

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology

in

Electronics and Communication Engineering

(Communication Engineering)

By

Patel Bhavin M.

(12MECC18)

Electronics and Communication Engineering Branch

Electrical Engineering Department

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2014

Software Device Driver Development & Automation of Test Framework

Major Project Report

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology

in

Electronics and Communication Engineering

(Communication Engineering)

By

Patel Bhavin M.

(12MECC18)

Under the guidance of

Assi. Prof. Sachin Gajjar

Electronics and Communication Engineering Branch

Electrical Engineering Department

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2014

Declaration

This is to certify that

i) The thesis comprises my original work towards the degree of Master of Technology in Electronics and Communication Engineering (COMMUNICATION) at Nirma University and has not been submitted else where for a degree.

ii) Due Acknowledgment has been made in the text to all other material used.

Patel Bhavin M.
(12MECC18)

Certificate

This is to certify that the Major Project entitled**" Software Device Driver Development & Automation of Test Framework "** submitted by **Patel Bhavin M.(12MECC18)**, towards the partial fulfillment of the requirement for the degree of the degree of Master of Technology in Electronics and Communication Engineering (COMMUNICATION) of Institute of Technology, Nirma University, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The result embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: **Place:** Ahmedabad

Internal Guide **Program Coordinator**

Mr. Sachin H Gajjar (Assi. Prof.,EC)

Dr. D K Kothari (Professor,EC)

HOD **Director**

Dr. P N Tekwani (Professor,EE)

Dr. K Kotecha (Director,IT-NU)

Acknowledgement

With immense pleasure, I would like to show my gratitude to those who have helped me directly and indirectly for the successful completion of the dissertation and for providing valuable guidance throughout the project work.

First and foremost,I would like to thank my institution guide **Assi. Prof. Sachin Gajjar** for his continuous support, encouragement and stimulating suggestions for this project.I am deeply indebted to **Mr. Nabrun Dasgupta**, Project Manager and **Mr. Amitkumar Pathak**,Mentor at **STMicroelectronics Pvt Ltd.**,Greater Noida for their constant guidance and motivation. I would also like to thank Head of EE Dept **Prof. P N Tekwani** and our PG Co-ordinator **Dr. D K Kothari** for his valuable guidance during the review process.

I would also like to thank our Director **Dr. K Kotecha** for encouragement for project work and also to the Nirma University for providing excellent infrastructure and facilities whenever and wherever required.

I also wish to thank my team members **Hemendra Singh, Subrata Chatterjee, Manu Sharma, Ankur Tyagi , Ankush Sabharwal, Shantanu Dey, Sanket Parmar, Dipak Goel** and **Rupesh Shrivastav** for their valuable help and support. Without their experience and insights, it would have been very difficult to do quality work.

Lastly I would like to thank **GOD, my Family** especially **my Mother** for support and encouragement. I would also like to thank all my **Friends** who have directly or indirectly helped.

Patel Bhavin M.

12MECC18

Abstract

Digital TV is becoming an emerging consumer electronics appliance. Set-Top-Box (STB) is the migration from analog to digital broadcasting. Set top Box is an instrument which converts the transmitted digital video signals to the data or the signals which can be displayed on the standard LCD, LED, and Analog TVs. In this report, basic overview and essential fundamentals of Set-Top-Box are discussed. For successful launch of any product and application, development and validation has to follow designing. Hence, validation approach is dealt in detail and it is following by testing of real time embedded systems like STB development board.

This report includes the generalize development procedure of the frontend device drivers of STB like Tuner, Demodulator and Forward error correction (FEC),Device driver testing and debugging techniques have been explained, as a part of embedded software development. Finally Automation of test framework in Set-Top-Box integration has been included.

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Set Top Box

1.1 Background

Digital television is a completely new way of broadcasting and is the future of television. It is a medium that requires new thinking and new revenue-generating business models. Digital TV is the successor to analog TV and eventually all broadcasting will be done in digital format. Around the globe, satellite, cable and Terrestrial operators are moving to a digital environment. The digital age will improve the customer viewing experience through cinema-quality pictures, CD-quality sound, hundreds of new channels. Television will become more fun and powerful to use, yet at the same time simpler and friendlier. Digital TV also opens up new world of opportunities for companies who want to develop content and applications for the new paradigm. This includes the creative communities within the TV and film industry, internet content providers and software development houses. Finally, the new medium will allow viewer from the comfort of their homes to use a simple remote control to electronically purchase goods and service offered by various content providers. Digital TV uses the same language as computers a long stream of binary digits, each of which is either 0 or 1. With digital TV, the signal is compressed and only the updated data is transmitted. As a result, it is possible to squeeze 6 or 8 channels into a frequency range

that was previously occupied by only one analog TV channel. The digital TV cycle begins by recording a particular event or program with digital equipment and is relay to a redistribution centre. In most cases, the redistribution centre will be a satellite, cable or terrestrial operator. The operator uses specific transmission techniques to broadcast the digital signal to viewers on their network.

The development of the digital TV and interactive services is already underway in various locations around the globe. During this transition period, TV operators will continue to broadcast analog signal in parallel to the new digital transmissions. And now a day's Indian government has announced to shutdown analog transmission and migrates to on digital transmission only. Central to this migration from analog to digital broadcasting is a small black box called Set-Top-Box. This Set-Top-Box (STB) sites on the top of a standard TV set which enable consumers to use their existing analog TV to participate in the digital revolution of the 21st century. These STB will provide consumers with a much better picture and sound quality. Experts are predicting that these STB will become a gateway to the much-hyped digital information superhighway.

Set top Box is an instrument which converts the transmitted digital video signals to the data or the signals which can be displayed on the standard LCD, LED, and Analog TVs. Front end in STB consists of tuner and demodulator.Digital video broadcasting of TV signals can be done over three different mediums viz., terrestrial, cable, and satellite. TV signals use DVB-C as the standard for transmission over the cable, DVB-T for terrestrial transmission; DVB-S for the Satellite transmission. The compression of TV signal can be done by standard MPEG-2 compression in all cases. But the modulation scheme used in each case is different[1].

1.2 Motivation

As Digital TV is becoming an emerging consumer electronics appliance and Set Top Box is the migration from analog to digital broadcasting. Development of Linux based hardware devices is on the centre point of the current industry trends. Developing software drivers of set top box front-end systems under Linux kernel results in low cost solution to the customers. For successful launch of any application, validation and testing has to follow designing development. So it's a challenging task to develop software drivers for Tuner,Demodulator and FEC.

1.3 Objective

- To understand the System and its work-functionality.
- To provide an understanding of the essentials of device drivers.
- To achieve practical experience in developing device drivers.
- The steps necessary to add devices to a system
- How to determine what hardware is present on a system
- The purpose and functionality of device drivers
- Compiling and linking device drivers
- Generate the test cases to automate the testing.
- Integrate, build up the functionality and stability of the software.

1.4 Organization of Thesis

- **Chapter 3** Describes the basic overview of linux device drivers and generic methodology to develop drivers. Driver development and run procedure is explained by writing a sample code of drivers. It also includes the role of device

driver development in Digital Set Top Box system. A brief view on features of set top box device drivers is also included.

- **Chapter: 4** This chapters provides all the details about set top box hardware setup, Application programmable interface SDK kit, software configuration environment, frontend driver's compilation and run procedure on two different STB oprating systems OS21 and STLinux.

- **Chapter: 5** This chapter includes Testing procedure of all STB frontend devices like tuner, demodulator and Forward Error Correction(FEC) with hardware arrangement and results.

- **Chapter: 6** How automation in software test framework results in low time consumption, less human error and less human interaction, reduction in development cost and increment in end product quality.

- **Chapter: 7** Finally the title Device Driver development and Automation of test framework has been concluded with future scope.

- Document ends with **Publication list** and **References** which includes some extra work while training.

Chapter 2

Literature Survey

2.1 Device Architecture

As shown in Figure 2.1 Set Top Box architecture is separated in two sub-systems.

2.1.1 Front-End

The front-end block is comprised of a tuner and a demodulator. It translates a RF signal into a digital-corrected stream, also called the transport stream. The digital demodulator in the cable is QAM, QPSK for satellite, and COFDM for terrestrial connection. Front-end devices are now embedded into the tuner can. The integration of complete tuner functions on silicon is now available for satellites, while silicon tuner for cables is still under development[1]. The front-end components are:

1. Tuners: Tuner receives a digital signal from a network and tunes to a particular channel in the corresponding frequency range. Basic purpose of a tuner is to amplify, detect, select and convert a desired RF signal from an antenna (or cable) to a demodulator while keeping the signal quality as much as possible. Main features and qualities of a tuner are:

   a. Sensitivity: defines the lowest RF signal power that can be received.

b. Selectivity: capability of the tuner to discriminate the desired signal out of all others received signals.

c. Dynamic range: span of acceptable of input desired signal power from the lowest to the highest.

d. Fidelity: capability to keep the desired signal characteristics.

Three types of tuners are generally being used in STBs:

- In-band tuners: The tuner is generally controlled by an I2C bus to select the required channel in the cable band (VHF/UHF from 50MHz to 860MHz), converts it into an IF, then feeds to a QAM demodulator.
- Out-of-band tuners: They facilitate the transfer of data between the head-end systems and the STB. They are typically used in a cable box when providing interactive services, and operate within the 100MHz to 350MHz frequency band.
- Return-path tuners: These tuners allow for activation of the return path and send data back to the head-end station. The frequency band allocated to the upstream is located between 5MHz to 65MHz frequency band.

2. Demodulators, modulators: The baseband output signal from the tuner continues on to the QAM demodulator and FEC, which performs sampled IF to bit streams that are fully compliant with ITU-T J83 annexes A/B/C or DVB-C specification. These bit streams contain A/V and data in the backend demultiplexer for MPEG-2 block processing. The modulator is to reverse the demodulator's actions and use the STB to deliver a signal to the return-path tuner. Type of demodulation depends on the type of tuner being used as given below.
QPSK demodulator is used in case of satellite transmission.
OFDM demodulator is used in case of terrestrial transmission.
QAM demodulator is used in case of Cable transmission.

Digital demodulator embedded into tuner is called Network Interface Module (NIM). NIM is controlled using I2C bus to select required channel, converts it into IF signal and feeds it to the demodulator.

2.1.2 Back-End

Back-end basically comprises of TS demultiplexer, A/V decoder, digital video encoder, DAC and CPU with on-site memory. These components are described as follows:

1. Demultiplexer: Demultiplexer (demux) extracts all the useful information from the TS, since MPEG-2 data streams consist of a number of unique data packets and uses packet ID to identify each packet that contains data, A/V and interactive services. The demux examines every packet ID, selects packets, decrypts them, and then forwards them to their specific decoder.

2. Decoders: Decoders are required to convert the digital bit stream back into the format accessible by the subscriber. The video decoder converts video packets into a sequence of pictures. Next, the audio bit stream is sent to the audio decoder for decompression so it can be sent to the speakers. Table formats are decompressed using data decoders. Then, the decoded data is presented to the set-top processor. Sometimes, JPEG and MP3 decoder are also embedded to process digital still picture and compressed music.

3. CPUs and memory: CPUs initialize various STB hardware components, process Internet and interactive TV applications, manage hardware interruptions, pull data from memory and run programs. CPU in STB is typically a 32bit processor with speed ranges from 50MHz to 300MHz. It always contains an arithmetic logic unit, a control unit and a clock.

Set top Box (STB) is instrument which converts the transmitted digital video signals to the data or the signals which can be displayed on the standard LCD, LED,

Standard	Modulation
DVB - S	QAM(Quadrature Amplitude modulation) and QPSK (Quadrature Phase shift keying)
DVB - C	QAM(Quadrature Amplitude modulation) and QPSK (Quadrature Phase shift keying)
DVB - T	OFDM (Orthogonal Frequency Division Multiplexing)

Table I: DVB Transmission Modulation Schemes

and Analog TVs. Front end in STB consists of tuner and demodulator and FEC. The hardware configuration of all DVB-receivers is same except for demodulator, which is different for different DVB receivers as their transmission schemes are different. This leads to the cost reduction in the set top boxes which can be used to receive all the three broadcasting signals.

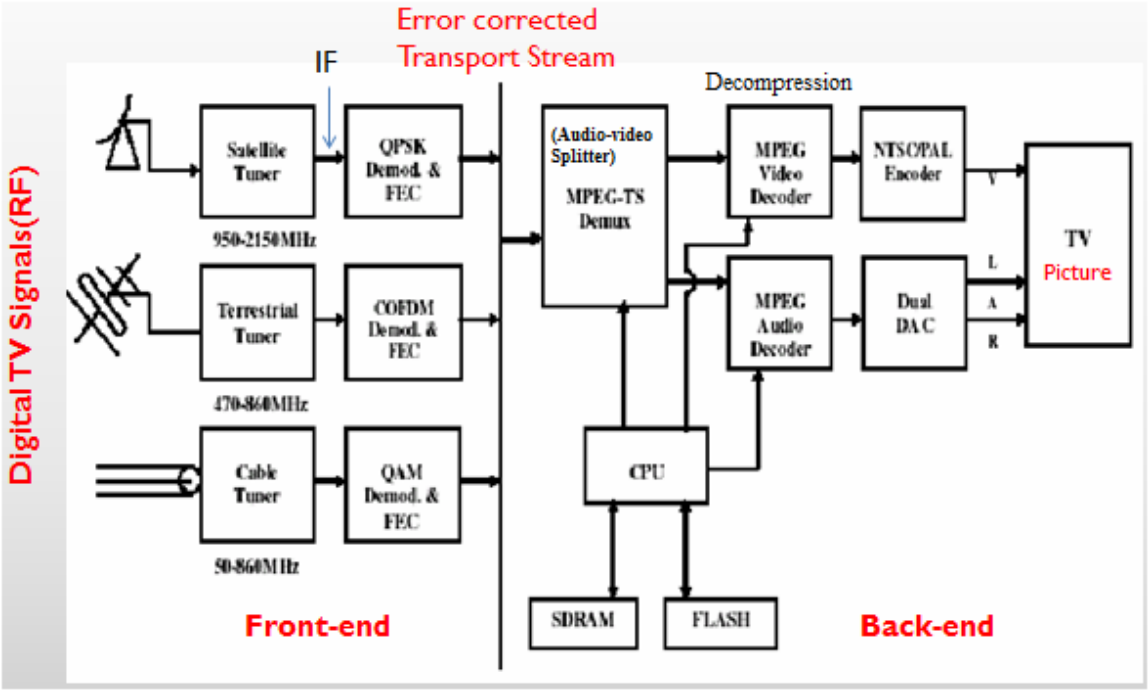

Figure 2.1: Digital Set-Top-Box

Digital demodulator embedded into tuner is called Network Interface Module (NIM). NIM is controlled using I2C bus to select required channel, converts into IF and feed to demodulator.

2.2 Broadcast Systems

2.2.1 Satellite Systems(DVB-S)

Satellite-based systems deliver programs and multimedia content from broadcasters, who use a number of geostationary satellites to relay their signals to customers back on earth. Customers must be within the "footprint" of a given satellite in order to receive the transmission. The set-top-boxes designed for receiving broadcasts from satellite-based systems were the first to be deployed. It is a system which will deliver a system which would improve reception quality and allow them to bring in services which would not be possible with conventional TV systems. A set-top-box decodes the incoming data from the satellite transponder. Each manufacturer has specified the requirements for their own particular system, and to a large extent this has governed the choice of as soon as possible used in the set-top-box designed for their system.
The audio and video data from the studio is compressed using MPEG-1 and MPEG-2 and up linked to the satellite. This data is the transmitted by the satellite. The transmission frequency of the communication satellites ranges from 10-17 GHZ (KU-Band)[2].

2.2.2 Cable Systems(DVB-C)

In cable systems, broadcasts are sent to the home via coaxial or optical fiber based cable. In the near future, DSL systems will also be able to deliver these services over normal twisted pair telephone wire. Cable-based systems are beginning to ramp up significantly in volume. The rapid deployment of satellite systems is seen as an

obvious threat by the cable companies. They are meeting the challenge, and are also looking to the future to how their set-top-boxes may evolve into the primary means for accessing the internet. This will open up the market and allow them to compete in areas normally associated with the PC systems. In the long run some industry analysts predict that cable based set-top-boxes could become the hub of a media-centric system, connecting systems and appliances within the home to their own network and acting as a bridge to the internet[2].

2.2.3 Terrestrial Systems(DVB-T)

In a terrestrial system, digital broadcast signals are transmitted via ground based transmitters in exactly the same way as analog television signals are transmitted. In fact, in the majority of cases exactly the same aerial can be used. Systems designed for terrestrial systems are limited in terms of the number of channels they can offer compared to both satellite and cable based systems. The modulation scheme required is more complex than that required for cable or satellite. In this system the data is spread over a number of frequency channels. A concatenated error correction system is used, and other techniques such as the use of "guard intervals" are also employed in order to ensure as robust a scheme as is practical. The system needs to be practical in order to overcome the effects of multi-path echo and noise effects which occur when the signals transmitted by a terrestrial system are reflected around objects[2].

2.3 Modulation Techniques for different broadcast systems

2.3.1 Satellite Transmission-QPSK Modulation

Satellite transmissions have a few unique characteristics viz.:

- The signal has to travel an extremely large distance (36,000 kilometres) from

the ground to the satellite and then another similar distance back to the earth.

- The satellite transmission is subjected to a broadband noise which is practically uniform at all frequencies.

- Since multiple channels are broadcast from the same satellite, the modulation technique should not be prone to Inter Channel interference.

- A satellite transponder has a fairly large bandwidth. Full transponders often have a bandwidth of 72 MHz This is fairly a wide bandwidth, particularly when compared with the 7 or 8 MHz allotted to a channel on a cable systems.

Hence a Digital Modulation technique used for Satellite Broadcasting (DVB-S) can use a fairly large bandwidth but should be capable of preserving the signal and maintaining a low Bit Error Rate (BER) even for very low signal strength. The QPSK Modulation system provides an ideal solution for this.

The word Quadrature simply means - Out of Phase by 90 Degrees. QPSK provides for 4 different states or possibilities for encoding a Digital Bit. This is because 2 components are used - one In Phase (I) & the other Out of phase or Quadrature (Q). This doubles the number of possible variations, from 2 to 4, that simple PSK offers. The QPSK system is now universally used, for all satellite DVB broadcasts.

2.3.2 CATV Transmission - QAM

- Quadrature Amplitude Modulation (QAM) systems utilize changes of both, Phase Shift Keying and Amplitude Shift Keying to increase the number of states per symbol.

- Each state is defined with a specific variation of both - Amplitude AND Phase.

- This means that the generation and detection of symbols is more complex than simple phase detection as in QPSK employed for Satellite Transmissions (DVB-S) because in QAM. The Amplitude changes have also to be detected.

QAM modulation is ideal for use in CATV networks. A cable system provides different transmission characteristics compared to satellite transmissions.

- The bandwidth allocated per channel is restricted - just 6 to 8 MHz Hence the Digital Modulation system must densely pack the digital data in a small bandwidth (unlike a satellite based transmission).

- The signal levels are significantly higher than for satellite transmissions. Since the Carrier (signal strength) is larger, the Carrier to Noise (C/N) ratio is always fairly good in a CATV network.

- A large number of channels are modulated and carried simultaneously on the same cable. Hence the modulation scheme should provide good Inter Channel Interference suppression.

QAM comfortably meets all these requirements:

- Since the Phase and Amplitude are varied in QAM Modulation, a large number of states or possible discreet values can be created to provide dense Digital Modulation.

- Each time the number of states or options per symbol is increased, the bandwidth efficiency also increases. This bandwidth efficiency is measured in bits per second/Hz. As higher density modulation schemes are adopted, the Decoder or Demodulator gets progressively more complex.

2.3.3 Terrestrial Transmission - OFDM.

- The biggest concern for proper reception of terrestrial broadcast is multi path distortion, or " Ghosts ".
- This happens when a signal arrives at the receiving antenna from multiple paths or direction.
- These multiple signals add up at the antenna, creating multiple images or "Ghosts" on the TV screen.
- Analog transmissions cannot prevent "Ghosts".
- Terrestrially transmitted Television signal should preferably not interfere with other terrestrial transmissions such as those for wireless radio etc.

Orthogonal Frequency Division Multiplexing (OFDM) is a type of Frequency Multiplexing.

- In Frequency Multiplexing, multiple carriers are used at different frequencies; each carrier is separated by an unused band of frequencies called a "Guard Band".
- A Digital Terrestrial transmission (DVB-T) for a single television channel can utilize up to 8000 separate carriers.
- Orthogonal here refers to a phase difference of 90 Degrees between two adjacent carriers. Using Orthogonal Frequency Division Multiplexing (OFDM) Modulation, 2 Adjacent Carriers will overlap without causing any interference because the two carriers are out of phase by 90 degrees.
- The overlapping of carriers avoids wastage of frequency bandwidth.

- OFDM causes less interference to analog transmissions than an analog signal would, because it doesn't have the same strong carrier and subcarrier elements. Also, because there is a specific spacing between carriers of the same phase (guard interval), the signal is immune to multi path reflections or "Ghosts ".

Chapter 3

Device Driver Development for STB Application

3.1 Overview of Device Driver Development

A driver drives, manages, controls, directs and monitors the entity under its command. What a bus driver does with a bus, a device driver does with a computer device (any piece of hardware connected to a computer) like a mouse, keyboard, monitor, hard disk, Web-camera, clock, and more.A specific piece of hardware could be controlled by a piece of software (a device driver), or could be controlled by another hardware device, which in turn could be managed by a software device driver. In the latter case, such a controlling device is commonly called a device controller. This, being a device itself, often also needs a driver, which is commonly referred to as a bus driver[3].

Place of a Device Driver in Linux

Figure 3.1. shows the place of a device driver in linux relative to the device:

- User program or utility

  A user program, or utility, makes calls on the kernel but never directly calls a device driver.

- Kernel

  The kernel runs in supervisor mode and does not communicate with a device except through calls to a device driver.

- Device driver

  A device driver communicates with a device by reading and writing through a bus to peripheral device registers.

- Bus

  The bus is the data path between the main processor and the device controller.

- Controller

  A controller is a physical interface for controlling one or more devices. A controller connects to a bus.

- Peripheral device

  A peripheral device is a device that can be connected to a controller, for example, a disk or tape drive. Other devices (for example, the network) may be integral to the controller.

3.1.1 Device Drivers

- **Device Specific:** The device-specific portion of a device driver remains the same across all operating systems, and is more about understanding and decoding the device data sheets than software programming. A data sheet for a device is a document with technical details of the device, including its operation, performance, programming, etc.

- **Operating system Specific:** OS-specific portion is the one that is tightly coupled with the OS mechanisms of user interfaces, and thus differentiates a Linux device driver from a Windows device driver and from a MacOS device driver.

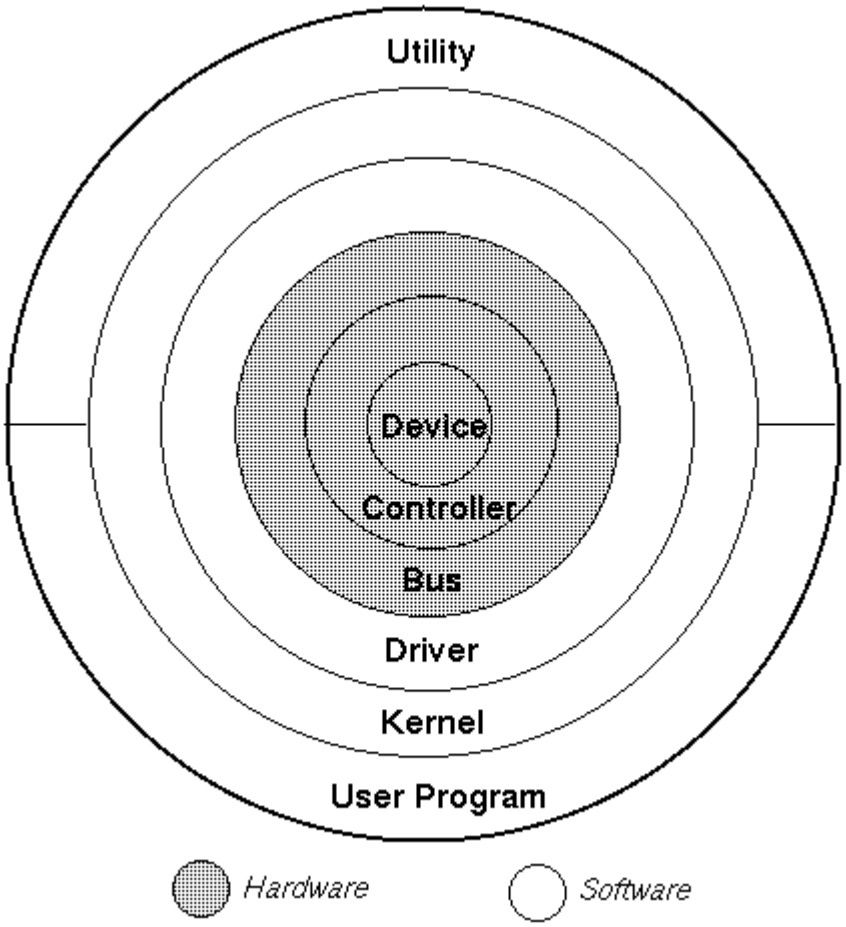

Figure 3.1: Place of Device Driver in Linux

Drivers are small programs that enable the linux kernel to communicate and handle hardware or protocols (rules and standards). Without a driver, the kernel does not know how to communicate with the hardware or handle protocols (the kernel actually hands the commands to the BIOS and the BIOS passes them on the the hardware). The Linux Kernel source code contains many drivers (in the form of source code) in the drivers folder. Each folder within the drivers folder will be explained. When configuring and compiling the kernel, it helps to understand the drivers.

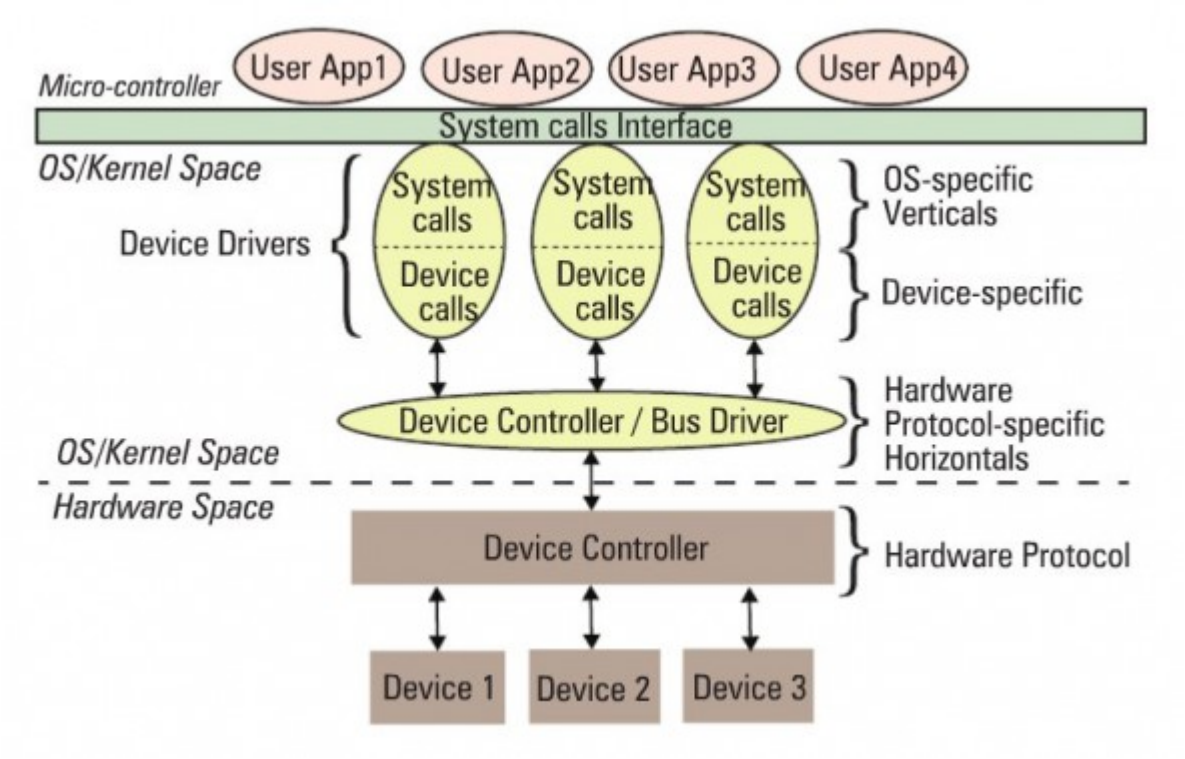

Figure 3.2: Linux device driver partition

3.1.2 Linux Kernel Overview

A kernel is the core of an operating system. The operating system is all of the programs that manages the hardware and allows users to run applications on a computer. The kernel controls the hardware and applications. Applications do not communicate with the hardware directly, instead they go to the kernel. In summary, software runs on the kernel and the kernel operates the hardware. Without a kernel, a computer is a useless object[2].

In Linux, a device driver provides a "system call" interface to the user; this is the boundary line between the so-called kernel space and user-space of Linux, as shown in Figure 3.3. Based on the OS-specific interface of a driver, in Linux, a driver is broadly classified into three types:

- **Network device drivers:** A network device driver attaches a network subsystem to a network interface, prepares the network interface for operation,

and governs the transmission and reception of network frames over the network interface. This book does not discuss network device drivers.

- **Block device drivers:** A block device driver is a driver that performs I/O by using file system block-sized buffers from a buffer cache supplied by the kernel. The kernel also provides for the device driver support interfaces that copy data between the buffer cache and the address space of a process.Block device drivers are particularly well-suited for disk drives, the most common block devices. For block devices, all I/O occurs through the buffer cache.

- **Character device drivers:** A character device driver does not handle I/O through the buffer cache, so it is not tied to a single approach for handling I/O. You can use a character device driver for a device such as a line printer that handles one character at a time. However, character drivers are not limited to performing I/O one character at a time (despite the name "character" driver). For example, tape drivers frequently perform I/O in 10K chunks. You can also use a character device driver when it is necessary to copy data directly to or from a user process.

  Because of their flexibility in handling I/O, many drivers are character drivers. Line printers, interactive terminals, and graphics displays are examples of devices that require character device drivers.

  A terminal device driver is actually a character device driver that handles I/O character processing for a variety of terminal devices. Like any character device, a terminal device can accept or supply a stream of data based on a request from a user process. It cannot be mounted as a file system and, therefore, does not use data caching.

User space and Kernel space

When you write device drivers, it's important to make the distinction between "user space" and "kernel space".

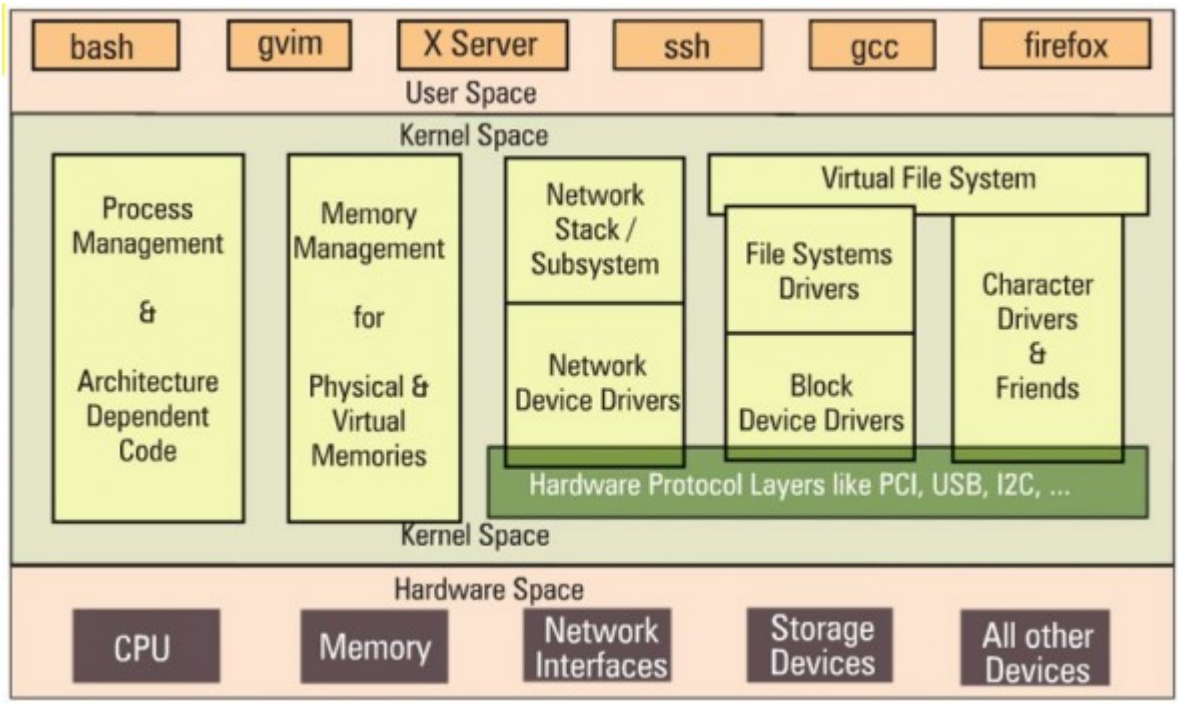

Figure 3.3: Linux Kernel Overview

- **Kernel space:** Linux (which is a kernel) manages the machine's hardware in a simple and efficient manner, offering the user a simple and uniform programming interface. In the same way, the kernel, and in particular its device drivers, form a bridge or interface between the end-user/programmer and the hardware. Any subroutines or functions forming part of the kernel (modules and device drivers, for example) are considered to be part of kernel space.

- **User space:** End-user programs, like the Linux shell or other GUI based applications are part of the user space. Obviously, these applications need to interact with the system's hardware . However, they don't do so directly, but through the kernel supported functions.

Interfacing functions between user space and kernel space

The kernel offers several subroutines or functions in user space, which allow the end-user application programmer to interact with the hardware. Usually, in UNIX or

Linux systems, this dialogue is performed through functions or subroutines in order to read and write files. The reason for this is that in Unix devices are seen, from the point of view of the user, as files.
On the other hand, in kernel space Linux also offers several functions or subroutines to perform the low level interactions directly with the hardware, and allow the transfer of information from kernel to user space.
Usually, for each function in user space (allowing the use of devices or files), there exists an equivalent in kernel space (allowing the transfer of information from the kernel to the user and vice-versa)
Device driver events and their associated interfacing functions in kernel space and user space are listed below.

- Load Module
- Open device
- Read device
- Write device
- Close device
- remove Module

Interfacing functions between kernel space and the hardware device

There are also functions in kernel space which control the device or exchange information between the kernel and the hardware. Which are illustrated below.

- Read data
- Write data

3.2 Writing Linux Driver

As we know, a typical driver installation on Windows needs a reboot for it to get activated. That is really not acceptable; suppose we need to do it on a server? That's where Linux wins. In Linux, we can load or unload a driver on the fly, and it is active for use instantly after loading. Also, it is instantly disabled when unloaded. This is called dynamic loading and unloading of drivers in Linux.

Dynamically loading drivers

These dynamically loadable drivers are more commonly called modules and built into individual files with a .ko (kernel object) extension. Every Linux system has a standard place under the root of the file system (/) for all the pre-built modules. They are organised similar to the kernel source tree structure, under /lib/modules/(kernel_version)/kernel, where(kernel_version) would be the output of the command (uname -r) on the system.

To dynamically load or unload a driver, use these commands, which reside in the /sbindirectory, and must be executed with root privileges:

- **lsmod** - lists currently loaded modules
- **insmod (module_file)** - inserts/loads the specified module file
- **modprobe (module)** - inserts/loads the module, along with any dependencies
- **rmmod (module)** - removes/unloads the module

Let's look at the FAT filesystem-related drivers as an example. The module files would be fat.ko, vfat.ko, etc., in the fat (vfat for older kernels) directory under /lib/modules/‘uname -r‘/kernel/fs. If they are in compressed .gz format, you need to uncompress them with gunzip, before you can insmod them.

The vfat module depends on the fat module, so fat.ko needs to be loaded first. To automatically perform decompression and dependency loading, use modprobe instead. Note that you shouldn't specify the .ko extension to the module's name, when using the modprobe command. rmmod is used to unload the modules.

Example of writing Linux driver

Before we write first driver, let's go over some concepts. A driver never runs by itself. It is similar to a library that is loaded for its functions to be invoked by a running application. It is written in C, but lacks a main() function. Moreover, it will be loaded/linked with the kernel, so it needs to be compiled in a similar way to the kernel, and the header files you can use are only those from the kernel sources, not from the standard /usr/include.
One interesting fact about the kernel is that it is an object-oriented implementation in C, as we will observe even with our first driver. Any Linux driver has a constructor and a destructor. The module's constructor is called when the module is successfully loaded into the kernel, and the destructor when rmmod succeeds in unloading the module. These two are like normal functions in the driver, except that they are specified as the init and exit functions, respectively, by the macros module_init() and module_exit(), which are defined in the kernel header module.h.

Figure 3.4 shows the complete code for our first driver; let's call it ofd.c. Note that there is no stdio.h (a user-space header); instead, we use the analogous kernel.h (a kernel space header). printk() is the equivalent of printf(). Additionally, version.h is included for the module version to be compatible with the kernel into which it is going to be loaded. The MODULE_* macros populate module-related information, which acts like the module's "signature".

```
/* ofd.c - Our First Driver code */
#include <linux/module.h>
#include <linux/version.h>
#include <linux/kernel.h>

static int __init ofd_init(void) /* Constructor */
{
    printk(KERN_INFO "Welcome: ofd registered");
    return 0;
}

static void __exit ofd_exit(void) /* Destructor */
{
    printk(KERN_INFO "Bye: ofd unregistered");
}

module_init(ofd_init);
module_exit(ofd_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Bhavin Patel <myemailid@yahoo.com>");
MODULE_DESCRIPTION("Our First Driver");
```

Figure 3.4: Sample Driver Program(ofc.c)

Building Linux driver

Once we have the C code, it is time to compile it and create the module file ofd.ko. We use the kernel build system to do this. The following Makefile invokes the kernel's build system from the kernel source, and the kernel's Makefile will, in turn, invoke our first driver's Makefile to build our first driver. To build a Linux driver, you need to have the kernel source (or, at least, the kernel headers) installed on your system. The kernel source is assumed to be installed at /usr/src/linux. If it's at any other location on your system, specify the location in the KERNEL_SOURCE variable in the Makefile shown in Figure 3.5.

```
# Makefile - makefile of our first driver

# if KERNELRELEASE is defined, we've been invoked from the
# Kernel builds system and can use its language.
ifneq (${KERNELRELEASE},)
    obj-m := ofd.o
# Otherwise we were called directly from the command line.
# Invoke the kernel build system.
else
    KERNEL_SOURCE := /usr/src/linux
    PWD := $(shell pwd)
default:
    ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules

clean:
    ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
Endif
```

Figure 3.5: Makefile

With the C code (ofd.c) and Makefile ready, all we need to do is invoke make to build our first driver (ofd.ko) as shown in Figure 3.6.

```
> make
make -C /usr/src/linux SUBDIRS=... modules
make[1]: Entering directory `/usr/src/linux'
  CC [M]  .../ofd.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      .../ofd.mod.o
  LD [M]  .../ofd.ko
make[1]: Leaving directory `/usr/src/linux'
```

Figure 3.6: Building Driver using Make Command

Loading Driver modules

Once we have the ofd.ko file, perform the usual steps as the root user, or with sudo.
su
insmod ofd.ko
lsmod — head -10

lsmod should show you the ofd driver loaded.
running dmesg — tail. We can found the printk output here.

dmesg — tail
Output:
KERN_INFO "Welcome: ofd registered
KERN_INFO "Bye: ofd unregistered

3.3 Features of Set Top Box Device Drivers

- Fully multi-instance -more than one set of Front End chipsets can be active at any time.
- Multi-standard support: Instances of devices receding signal different transmission media can co-exist e.g. record from satellite while watching terrestrial on appropriate HW platform.
- Extensible driver based model:
  - Higher level satellite, cable and terrestrial device (manager) code is driver independent.
  - Uniform interface with low level device specific drivers

  - Memory saving build options -Only those drivers required in a specific build can be selected.
  - Drivers are selected from a pool of those available during initialization(installation)
  - All platform/ application specific configuration is done in one file
  - Low level driver code has one to one correspondence with ST's PC based GUI.

- Uniform APIs across device and technology.
- I/O is handled for all device-drivers through an I/O manager enabling routing and easy debug.
- Debugging build flags are on a file by file basis for all files in STFROTNEND (including drivers).
- API function STFRONTEND_CustomiseControl enables low-level access to device managers and device-drivers enabling external intellectual property (IP)/ changes to be built on top of driver and device probe and register dump. This function also provides rarely used features. Minor changes can be accommodated without change in driver code.
- Extendibility: Template provided for easy addition and support of new tuners etc.
- Driver code is HW SOC platform is Independent. In addition, operating system calls are abstracted through STAPI driver STOS.

3.4 Summary

Having followed this chapter we have an understanding of linux kernel and its role to serve hardware resources to user demands via user space and driver functionality. we

Function	Description
STFRONTEND_Init	Initialise an instance of the named Frontend driver.
STFRONTEND_Term	Terminate instance of the Frontend
STFRONTEND_Open	Open a initialized instance and obtain a handle
STFRONTEND_Close	Close instance of the Frontend.
STFRONTEND_GetRevision	Returns the revision of the Frontend driver.
STFRONTEND_SetFrequency	Scan to an exact frequency
STFRONTEND_Unlock	Unlock or abort the current scan
STFRONTEND_GetTunerInfo	Get the current driver scan status and tuning information
STFRONTEND_GetStatus	Get the current driver scan status
STFRONTEND_Scan	Scan a Band, Stops at first valid signal and reports lock. Can be used for Blindscan features.
STFRONTEND_ScanContinue	Continue the previous STFRONTNED_Scan() till the next valid signal in the band.
STFRONTEND_DiSEqCSendReceive	Send Tone signals/ DiSEqC Command and receive corresponding reply if any.
STFRONTEND_SetLNBConfig	STB co-ax (LNB) output voltage and tone state
STFRONTEND_OutdoorUnitConfigure	To select DiSEqC/FSK in path 2
STFRONTEND_OutdoorUnitControl	To control FSK modulator
STFRONTEND_StandByMode	Invoke /terminate standby power mode
STFRONTEND_CustomiseControl	Low-level access to device. Register Dump; Non standard settings.

Table I: Functions to Develop Drivers

should now be capable of writing our own complete device driver for simple hardware or a minimal device driver for complex hardware. Learning to understand some of these simple concepts behind the Linux kernel allows you, in a quick and easy way, to get up to speed with respect to writing device drivers.

Chapter 4

Drivers Test Environment Setup with STAPI-SDK

4.1 STAPI-SDK (ST Application Programmable Interface-Software Development Kit)

The STAPI-SDK provides a unified software development platform for a range of set-top box(STB) devices and operating systems. In brief, the software stack consists of a set of low-level STAPI drivers and an application layer. The application layer supports an in-built diagnostic tool, testtool, that provides a wide range of testing and debugging functions.

In this section, we will see that instructions on how to install, compile and run the STAPI-SDK libraries and application

4.2 Generalize hardware setup

- A host PC (For compiling STAPI-SDK)

- A development board (For testing and debugging the compiled STAPI-SDK application) an ST Micro Connect 2 (which provides the physical interface between the host PC and the development board)

- A USB cable or Ethernet cable (To connect the ST Micro Connect 2 to the host PC directly or through a network)

- An LVDS target board connection cable (To connect the development board to the ST Micro Connect 2)

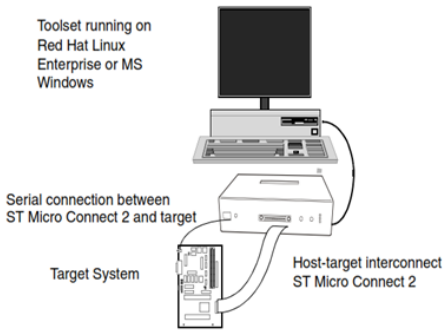

Figure 4.1: STAPI-SDK development environment

4.3 Configuring the ST Micro Connect 2 for serial data out

To configure the ST Micro Connect 2 to receive the serial data output from the target board:

1. For Windows, check that the PATH variable includes the correct STMC bin path. (For example: C:\ STM\ STMCR1.6.0\ bin.)

2. At command prompt, enter the following command:
   stmcconfig –ip <microconnect-ip> –serial-relay
   The command returns a response similar to the following:
   Starting serial relay : ip: <microconnect-ip> port: 5331

3. Run Telnet to receive serial output from the target board. Use the following command:
   syntax to specify the port number from the response at step 2.
   In this example, the port number is 5331.
   telnet <microconnect-ip> 5331

Toolsets and other software

To compile the STAPI-SDK for a given core, make sure that the appropriate toolset for that core is installed on the host PC. For example, the ST40 Micro Toolset is required to compile STAPI for an ST40 core running OS21, and the ST200 Micro Toolset is required to compile STAPI for an ST200 core.

4.4 STAPI installation

STAPI is supplied as a compressed archive file. This is a .ZIP file for MS Windows users, and a gzip-compressed tar file for Linux users. The release notes for each version of STAPI provide information on how to obtain the release, including the names of the archive files.

To install STAPI, uncompress the contents of the archive in a suitable location, using the archive utility that is appropriate to the host operating system.

aplib/	This directory contains the source code of all the STAPI drivers.
bin/	The directory contains the configuration file (setenv.bat or setenv.sh) and various other tools.
docs/	This directory contains the user documentation for STAPI-SDK, including this user manual
stapp/	This directory contains all the initialization and set up files for the STAPI drivers. The built STAPI executables are also located here. It also contains the makefile for compiling the STAPI-SDK tree.
stdebug/	This directory contains all the source files that define the testtool commands.

In this User manual, the root directory of the STAPI-SDK installation is <SDK_ROOT> in pathnames. This directory contains the following directories:

The STAPI-SDK is available for two different host operating systems: MS Windows or a standard Linux distribution, such as Red Hat Enterprise Linux, or Fedora. The STAPI-SDK can be compiled to run on target platforms running either OS21 (with OSPlus) or STLinux.

- On a Linux host environment, STAPI-SDK can be built for both OS21 and STLinux targets.
- On an MS Windows host environment, STAPI-SDK can be built for OS21 targets only

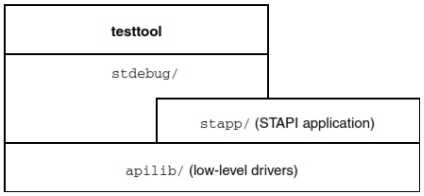

Figure 4.2: Schematic representation of the STAPI-SDK

4.5 Apilib

The apilib/ directory contains the low level drivers that provide an interface with specific devices. The library consists of a number of different modules. A full list of the modules is given in the release notes. Each module has its own API, which is documented separately.

The make apilib command generates a separate library (∗.a file) for each of the apilib/ modules. These are located in a sub directory of <SDK_ROOT>/apilib/lib which is named for the target board, SoC type and operating system for which it has been compiled.

4.6 Patches

For older versions of operating systems, toolsets or both, it may be necessary to apply patches before compiling STAPI-SDK. The release notes provide details of the patches that are required for any given release of operating system or toolset.

4.7 OS21

The patches for OS21 are provided as zip files, and are found in the <SDK_ROOT> \ bin\ patches\ os21\ directory. If a patch exists for a given version of toolset, then it should be applied to that toolset before building the SDK.

4.8 STLinux

STAPI-SDK is compatible with STLinux-2.3. However, the kernels of both versions of STLinux must be patched with the appropriate patches included with the STAPI-SDK distribution before being compiled. The patches for STLinux are provided as

.tgz tar balls and located in the <SDK_ROOT>/bin/patches/linux/ directory.

The release notes accompanying the STAPI-SDK distribution provides full details of the various versions of toolset and operating system that require patching. This is likely to change for different revisions of STAPI-SDK and different versions of each toolset and operating systems.

4.9 Compiling and running STAPI-SDK(OS21)

The STAPI-SDK is provided in the form of source files that must be compiled for the required target using the toolset that is appropriate for the platform. There are slight differences in the compilation procedure, depending on whether STAPI-SDK is being compiled on an MS Windows or Linux host. The two compilation procedures are described below.

Following installation of the STAPI-SDK package, the next step is to configure the compilation environment for a specific target and then compile the libraries and the STAPI application. This chapter provides instructions for compiling STAPI-SDK for an SoC running the OS21 real time operating system.

4.10 Board configuration for MS Windows machine

Before compilation, the STAPI-SDK environment must first be configured for a specific target platform. The STAPI-SDK installation includes a batch file for windows machine that performs the required configuration. The batch file for MS Windows is called <SDK_ROOT> \bin\setenv.bat.

Several of the environment variables in setenv.bat reference the paths of specific

directories on the host. Although the version of setenv.bat included in the STAPI-SDK distribution contains default values for the standard STAPI-SDK and toolset installation, it may be necessary to modify some of the environment variables to suit your particular installation. In particular, the variable STSDKROOT must be set to <SDK_ROOT>, the root directory of the STAPI-SDK installation. Before running the batch file, check that the SET commands listed in Figure 3.3 do correspond with the file paths of the various toolsets and other resources that are installed on the host system, and modify them if required.

```
SET STSDKROOT=C:\STAPI_SDK
SET STPPROOT=C:\STM\STMCR1.6.0
SET ST40ROOT=C:\STM\ST40R5.1.0
SET ST200ROOT=C:\STM\ST200R7.1.0
SET OSPLUSROOT=C:\STM\OSPLUSR3.2.4
SET WORKBENCHROOT=C:\STM\STWORKBENCHR6.0.0
```

Figure 4.3: Extract from setenv.bat

NOTE: If setenv.bat is run with incorrect file paths, the user must close the current DOS window and open a new one before running the updated setenv.bat again. Although environment variables are defined for the full range of toolsets, only the toolsets required for the platforms you are working with need to be installed.

NOTE:

1. In most cases, OSPlus is mandatory if compiling for OS21.

2. Before running setenv.bat, either make sure that the <SDK_ROOT>/bin/ directory is in the PATH, or run setenv directly from the bin directory.

To configure the STAPI-SDK environment for a specific platform, run the configuration file as follows:
setenv <platform> _ <SoC>

where <platform> is the name of the board and <SoC> is the name of the back-end SoC that is mounted on that board.

For example, to configure the environment for an MB680 board with an STb7105 SoC, enter the following:
setenv MB680_7105

The batch file responds with the following message:
MB680_7105 Configuration selected!

Run setenv without any arguments to display a list of all the platform and SoC combinations that it currently supports.

Note: The arguments passed to setenv.bat are case sensitive. Enter the arguments with exactly the same capitalization as in the list generated by the setenv.bat script.

If a platform and SoC are specified, but setenv.bat outputs a list of all the platform and SoC combinations it supports instead of giving the Configuration selected! Message, then either the configuration is not supported, or the platform and SoC names were entered wrongly.

4.11 Board configuration for Linux/UNIX machine

Before compilation, the STAPI-SDK environment must first be configured for a specific target platform. The STAPI-SDK installation includes a shell script file for UNIX or Linux machine that performs the required configuration. The shell script for UNIX or Linux machine is called <SDK_ROOT> \bin\setenv.sh

Several of the environment variables in setenv.sh reference the paths of specific direc-

tories on the host. Before running the batch file, edit the export commands listed in Figure 4 so that they correspond with the filepaths of the various toolsets and other resources that are installed on the host system.

Although environment variables are defined for the full range of toolsets, only the

```
export STSDKROOT=/opt/STM/STAPI_SDK
export STPPROOT=/opt/STM/STMCR1.6.0
export ST40ROOT=/opt/STM/ST40R5.1.0
export ST200ROOT=/opt/STM/ST200R7.1.0
export OSPLUSROOT=/opt/STM/OSPlusR3.2.4
export WORKBENCHROOT=/opt/STM/STWORKBENCHR6.0.0
```

Figure 4.4: Extract from setenv.bat

toolsets required for the platforms you are working with need to be installed.

Note:

1. In most cases, OSPlus is mandatory if compiling for OS21.
2. Before running setenv.sh, either make sure that the <SDK_ROOT>/bin/ directory is in the PATH, or run setenv directly from the bin directory.

To configure the STAPI-SDK environment for a specific platform, run the configuration file as follows:
source setenv.sh <platform> _ <SoC>
where <platform> is the name of the board and <SoC> is the name of the backend SoC that is mounted on that board. For example, to configure the environment for an MB680 board with an STb7105 SoC, enter the following:

source setenv.sh MB680_7105

The shell scripts responds with the following message:

MB680_7105 Configuration selected!

Run setenv without any arguments to display a list of all the platform and SoC combinations that it currently supports.

Note: The arguments passed to setenv.sh are case sensitive. Enter the arguments with exactly the same capitalization as in the list generated by the setenv.sh script.

To compile STAPI-SDK for OS21 on Linux or Unix Machine, ensure that all the required versions of tools as listed in Figure 3.4 are properly installed on the appropriate machine.

If a platform and SoC are specified, but setenv.sh outputs a list of all the platform and SoC combinations it supports instead of giving the Configuration selected! Message, then either the configuration is not supported, or the platform and SoC names were entered wrongly.

4.12 Compiling STAPI-SDK

Compiling the STAPI-SDK tree is a two stage operation. The first stage is to compile theSTAPI libraries, and the second stage is to compile the application itself and link it with the STAPI libraries. The reason for the two stage compilation is because compiling the STAPI drivers can be very time consuming.

For MS Windows, the STAPI distribution includes a copy of gmake.exe. The configuration script setenv.bat automatically adds this executable to the PATH. Use this

Target	**Description**
help	Display help text.
all	Compile stdebug -stapp and link it with the apilib library.
apilib	Compile apilib library only.
apilib MODULE=< *module* >	Compile the specified module in the apilib directory. The release notes provide full list of module names for the current release of STAPI-SDK.
Stapp	Compile contents of stapp directory only.
stdebug	Compile contents of stdebug directory only.
Purge_all	Clean the stapp and stdebug directories for the currently specified platform.
Purge	Clean stapp directory only
Purge_apilib	Clean apilib directory only
Purge_apilib MODULE=< *module* >	Clean specified module in the apilib directory.
run	Run the application without debug.

Table 4.1: Targets available with the STAPI-SDK make file

version of make in preference to any other make program that may already be installed on the host. All compilation activities are carried out from within the stapp/ directory. The STAPI-SDK makefile is located in this directory, so all make operations can be initiated from this directory by typing gmake followed by the relevant target. A list of the available targets is given in Table 3.1.

Note: Combinations of targets are permitted. The following example is an acceptable gmake command:

gmake purge_apilib purge_all apilib all run

gmake makes each of the named targets in the order in which they are given on the command line, reading from left to right.

The following commands are available if the $(MODULES) variable is set to 1. (The default is 0.)

4.13 Compiling the STAPI libraries

The gmake command for compiling the STAPI libraries is:
gmake apilib

This command builds all the modules in the <SDK_ROOT>/apilib/ directory and installs the resulting object files so that they can be linked to the STAPI application.

To clean the apilib/ directory by removing all the intermediary files generated by an earlier build, enter the command: gmake purge_apilib

To compile a single STAPI driver, use the MODULE command line argument. For example, to compile just the stvid driver, enter the command:
gmake apilib MODULE=stvid

To clean the files for a specific STAPI driver only, use the MODULE argument to the purge_apilib command. For example, to clean the stvid driver directory, enter the command:
gmake purge_apilib MODULE=stvid

Object directory
During the operation to build the STAPI libraries, make constructs a name for the directory in which object files and the libraries are placed which includes some or all of the following elements:

- Target platform name (for example, mb618 or mb680)
- Backend SoC name (for example, 7111 or 7105)
- Operating system (OS21 or Linux)

- Address mode (29 or 32)

For example, the library directory for the MB680 platform with 7105 backend has the path:
<SDK_ROOT> \ apilib\lib\mb680_7105_ST40_OS21_32BITS

This means that if an application is generated on the same machine for different platforms, the libraries and binary files for each platform are kept separate from one another. Recompiling any item for one platform has no effect on the software already compiled for a different platform. Also, the purge operation only removes files relating to the platform currently selected, and leaves all the files relating to other platforms untouched.

4.14 Compiling and linking the STAPI application

When the apilib libraries have been compiled, the next step is to compile the STAPI application itself.The source files for STAPI are located in the <SDK_ROOT> \ stapp directory. The gmakecommand for compiling the application and linking it with the pre-compiled libraries is:
gmake all

This generates an executable file in the stapp directory for the target with the name:
main_$(DVD_PLATFORM)_$(DVD_BACKEND)_(ARCHITECTURE)_(DVD_OS)
_[29—32]BITS.out
DVD_PLATFORM and DVD_BACKEND are the values of the %DVD_PLATFORM% and %DVD_BACKEND% environment variables, as set by the setenv.bat batch file. This naming convention ensures that is possible to build different applications for different platforms within the same build environment.

4.15 Running the STAPI application on the target

When the STAPI-SDK libraries have been compiled, the resulting executable can be run on the target. The executable file is first transferred to the target using an ST Micro Connect.

4.15.1 Configure the ST Micro Connect

STAPI-SDK supports both the ST Micro Connect 1 (STMC1) and the ST Micro Connect 2 (STMC2). The SoC determines which ST Micro Connect is selected as the default: generally, the default for older SoCs (such as STi7109) is the STMC1 and the default for newer SoCs (such as the STi7111) is the STMC2.

Note: If the environment variable USE_TARGETPACK is set to 1, then the system is forced to use the STMC2, irrespective of the default. This is because only the STMC2 supports ST Targetpacks.
Use the environment variable $TARGET to identify the ST Micro Connect to the system. If you are using an STMC1 through a USB connection, then set $TARGET as follows:
set TARGET=usb

Note: The gmake tool is case-sensitive, so make sure that you enter usb as written, in lower case.
For the STMC2, set $TARGET to the IP address of the ST Micro Connect, or to a name that can be translated into an IP address. For example:
set TARGET=192.168.1.101
or:
set TARGET= mystmc

4.15.2 Running the software

Having configured the ST Micro Connect, the next step is to run the STAPI application on the target. This is done by executing the following gmake command: gmake run

The ST Micro Connect loads the software onto the target and runs it. The makefile configures the connection.

Note:

1. When the application is running, it displays the testtool prompt.

2. If the Windows Data Execution Protection system (DEP) has been switched on for gdb, this may prevent gdb running executable code when gmake run is invoked. To prevent this, ensure that the Windows Data Execution Protection system is switched off for gdb. This is done in Control Panel > System > Advanced > Performance > Data Execution Prevention.

4.15.3 Debugging the Software

To run the application using the debug interface, enter the following command: gmake debug

To run the application using the console debugger interface, enter the following command:
gmake debug CONSOLE=1

To run the application using STWorkbench, enter the following command: gmake debug STWORKBENCH=1

This command launches STWorkbench with the appropriate command line parameters to create a new STWorkbench project containing the STAPI executables. The application stops at a breakpoint on the first line of the main() function.

Note: The CONSOLE and STWORKBENCH options are only available for ST40 based platforms. The STWORKBENCH option is only available if STWorkbench is installed.

4.16 Compiling and running STAPI-SDK (STLinux

Following installation of the STAPI-SDK package, the next step is to configure the compilation environment for a specific target and then compile the libraries and the STAPI application. This chapter describes these operations for a host machine running Linux compiling STAPI-SDK for an SoC running STLinux.

It is not possible to build STAPI-SDK for STLinux on a host that is running MS Windows XP.STAPI-SDK for STLinux can only be built on a host running a standard Linux distribution,such as Red Hat Enterprise Linux, or Fedora.

4.16.1 Clean current STLinux distribution

When installing STAPI-SDK on a board that already has STLinux installed, STMicroelectronics recommend that you first clean the entire installation to ensure that there is no extra software installed that may interfere with the operation of the STAPI-SDK.

SDK supports both STLinux 2.3 and STLinux 2.4. In the instructions given throughout this chapter, substitute x for either 3 or 4, as appropriate to the version of STLinux that you are using.

For STLinux 2. x, the command for removing any currently installed rpm packages is:
rpm -qa — grep stlinux2.x

Within a bash shell, enter the following for STLinux 2.x:
rpm -e –nodeps 'rpm -qa — grep stlinux2.x
rm -rf /opt/STM/STLinux-2.x

Under certain circumstances, the removal of some packages may fail due to script failures.To prevent this happening, add the option –noscripts to the rpm command.

The following command line provides a more "verbose" output of the process to remove packages:
rpm -qa — grep stlinux2x — xargs -i -t rpm -e –nodeps –noscripts

4.16.2 Installing STLinux

The installation package for STLinux 2.x is located at ftp://ftp.stlinux.com/pub/stlinux/2.x/. There is a README file at this location that gives current information concerning this release of STLinux2.x.

There are two different methods for completing the installation: either downloads the ISO image from the URL given above and then run the install script, or just

download the install script ftp://ftp.stlinux.com/pub/stlinux/2.x/install to a Linux host and complete the installation over the network.

You can download the files you require either through a web browser or using an FTP client.

Install script

To see a list of all the available installation options, including the current install profiles, run the install script as follows:

./install –help

Install the profile that you want by giving it as a parameter to the install script. For example, the following command installs all the packages for the ST40 architecture and glibc tool chain.

./install all-sh4-glibc

To install the uclib toolchain in place of glibc, use the following command:

./install all-sh4-uclibc

Verify the installation

Verify that the installation was successful by inspecting the name of the Linux kernel.

ls /opt/STM/STLinux-2.x/devkit/sources/kernel -l

- For STLinux version 2.3, the default name of the linux-sh4 kernel is linux-sh42.6.23.1_stm23_0102

- For STLinux version 2.4, the default name of the linux-sh4 kernel is linux-sh42.6.32.10_stm24_0201

The installation creates a root file system for the target, which includes device files and setuid programs.

4.17 Board configuration for STLinux

Before compilation, the STAPI-SDK environment must first be configured for a specific target platform. The STAPI-SDK installation includes a batch file (for MS Windows) and a shell script file (for Linux) that performs the required configuration. The shell script for Linux is called setenv.sh and is located in the directory <SDKROOT>/bin.

There are certain environment variables in setenv.sh that must be modified to correspond with your installation. These variables are:

- LINUX_TARGETIP (Target IP address)
- LINUX_SERVERIP (Server IP address)
- LINUX_GWIP (Network gateway IP address)
- LINUX_NETMASK (Network subnet mask)
- LINUX_NAME (The name of the host)
- LINUX_SERVERDIR (The path of the root of the target's file system)
- LINUX_PARAMETERS (The kernel parameters for the host)

Also, it is important to check that the definition of the variable KDIR (the path to kernel source code) is correct, as the default value: /opt/STM/STLinux-$LINUX_VERSION/devkit/sources/kernel/linux-sh4 may not be correct for all installations.

To configure the STAPI-SDK environment for a specific platform, ensure that <SDKROOT>/bin is on the path, and then run the configuration file as follows:
source setenv.sh ¡platform¿_<SoC>_<LINUX>

where <platform> is the name of the board and <SoC> is the name of the back-end SoC that is mounted on that board. For example, to configure the environment for an MB411 board with an STb7109 SoC, enter the following:
source setenv.sh MB411_7109_LINUX

The batch file responds with the following message:
MB411_7109_LINUX Configuration selected!

Run setenv without any arguments to display a list of all the platform and SoC combinations that it currently supports.

Note: The arguments passed to setenv.sh are case sensitive. Enter the arguments with exactly the same capitalization as in the list generated by the setenv.sh script. If a platform and SoC are specified, but setenv.sh outputs a list of all the platform and SoC combinations it supports instead of giving the Configuration selected! Message, then either the configuration is not supported, or the platform and SoC names were entered wrongly.

4.17.1 Customized configuration for STLinux

It is possible to create a customized environment by editing the values of these environment variables. This may be necessary, for example, when using STAPI-SDK for a platform not listed in the default configuration file.

Table 1: Environment variables in setenv.bat and setenv.sh gives a list of the environment variables in setenv.sh that are common to all operating systems (OS21 and ST Linux) and which may be edited in order to customize the configuration.

The Linux only version of the configuration file, setenv.sh, contains some additional environment variables that are only applicable when running ST Linux on the target. These variables are listed in Table 3.2.

Note: In some cases, the values of the modified environment variables may not be exported. In order to avoid this problem; run the modified version of setenv.sh in a new shell.

4.18 STAPI-SDK makefile

All compilation activities are carried out from within the stapp/ directory. The STAPI-SDK makefile is located in this directory, so all make operations can be initiated from this directory by typing make followed by the relevant target.

For STLinux, the makefile accepts a number of additional targets. These are listed in Table 3.3. The important targets are described in greater detail in the following sections.

Variable	Default Values	Comments
DVD_OS	LINUX	Use this variable to indicate that the SDK is to be compiled for Linux. The variable DVD_OS is not initialized for non-Linux configurations (such as OS21) because the makfiles for these operating systems already define the OS value used by default
PATH	/opt/STM/STLinux2.x/ devkit/sh4/bin:$PATH	Use this variable to add the the Linux cross compiler in the path (sh4-linux-*). Here the cross compiler is installed in /opt/STM/ STLinux-2.x/devkit/*.For uclibc support, set this variable to the default uclibc path:opt/STM/ STLinux-2.x/devkit/sh4 _uclibc/*
KDIR	/opt/STM/STLinux2.x/ devkit/sources/kernel /linux-sh4	Use this variable to define the root path of kernel source code.
KTARGET	/opt/STM/STLinux2.x/ devkit/sh4/target /root	Use this variable to define the path where all the STAPI kernel objects and scripts are to be installed during the SDK compilation process.For uclibc support, set this variable to:/opt/STM/STLinux-2.x/devkit /sh4_uclibc/target/root
LINUX_VERSION		Use this variable to define the version of the STLinux distribution.The value given to this variable is used to construct path names needed to compile or run the Linux distribution
LINUX_TARGETIP	192.168.1.50	This variable contains the IP address to be given to the platform.
LINUX_SERVERIP	192.168.1.51	This variable contains the IP address of the NFS server used by the platform. In most the cases, this is the IP address of the Linux Host machine.

Table 4.2: Additional Linux-specific environment variables in setenv.sh

Target	Description
Kernel	Compile the Linux kernel without the configuration menu. The kernel is compiled with the default configuration.
purge_kernel	Clean the Linux kernel directories.
run_kernel	Load the Linux kernel on the target (using the ST Micro Connect) and run the kernel without debug.
run_kernel DETACH=1	Load the Linux kernel on the target (using the ST Micro Connect) and run the kernel with debug. The command can also be used to perform the debug from the console with the following parameters : debug_kernel CONSOLE=1
debug_kernel	Load the Linux kernel on the target (using the ST Micro Connect) and run the kernel with debug. The command can also be used to perform the debug from the console with the following parameters : debug_kernel CONSOLE=1
Install	Copy apilib modules and main.out to the relevant installation directories

Table 4.3: Targets available with the STAPI-SDK makefile for STLinux

4.19 Building the kernel

To build a new kernel for the target system, use the command make kernel.

Note: The make kernel command must be executed as the root user. This operation builds the STLinux kernel using a kernel configuration file. The kernel configuration files are located in the stapp/platform directory. make constructs the path of the appropriate configuration file using the values of the DVD_PLATFORM and DVD_BACKEND environment variables, according to the following template:
$(DVD_PLATFORM)/$(DVD_BACKEND)/linux/$(DVD_PLATFORM)
_$(DVD_BACKEND)_kernel-2.x.cfg

For example, the configurations file for the kernel for the mb704 board with a STx5197 SoC has the following path: <SDK_ROOT>/stapp/platform/mb704

/5197/linux/mb704_5197_kernel-2.x.cfg

make builds the kernel using the configuration defined in the selected .cfg file.

To customize the configuration, invoke make with the command make kernel MENUCONFIG=1. This option displays the configuration menu; additional options for configuring the kernel can be selected from this menu.

The target purge_kernel purges all the object files created by make kernel and also cleans up the Multicom directories.

To debug the kernel, enter make debug_kernel. This command loads the kernel on the target (using the ST Micro Connect referenced by the $TARGET environment variable) and runs the kernel in the graphical debugger.

4.20 Compiling the STAPI-SDK tree for Linux

The process of compiling the STAPI-SDK tree on Linux is a three step process:

1. Compile Multicom with the command make multicom. This generates library files in the form of *.ko files. These are loaded at run time before the STAPI libraries.

2. Compile the apilib libraries with the command make apilib. This generates the library files in the form of *.ko files, which are loaded dynamically when the kernel is running.

3. Compile the STAPI application with the command make all. This operation generates the *.out file, and also performs the make install step, copying all the

relevant modules to the final target directory (as specified by the $KTARGET environment variable.

Note: Both the make apilib command and the make all command must be executed as the root user.

4.20.1 Compiling the STAPI-SDK tree for Linux with uclibc

Before compiling the STAPI-SDK tree for Linux with uclibc support, the uclibc toolchain for ST40 must first be installed When this toolchain is successfully installed, you will be able to see the path:
opt/STM/STLinux-$LINUX_VERSION/devkit/sh4_uclibc

Compile STAPI-SDK as follows:

1. Set an additional environment variable: CROSS_COMPILE=sh4-linux-uclibc-

2. Set the relevant environment variables PATH, KTARGET and SERVER_DIR to take account of the uclibc path.

3. Build STAPI-SDK.

4. The uclibc compliant library can be seen in the folder STAPI_SDK/STAPP/playrec/< *uclibc* >. This library replaces the default library delivered with the SDK (STAPI_SDK/STAPP/playrec/*) to make it uclibc compliant.

4.21 Running STAPI-SDK for STLinux

The following example assumes that the host PC is the host of the the STLinux root filesystem. Make sure that the filesystem of the host PC is visible to the target board on the network before attempting to run STAPI-SDK on the target. The path to the root of the visible directories on the host is defined by the environment variable $LINUX_SERVERDIR; this path must correspond with the host directory exported by NFS (that is, the path returned by the showmount -e command).

1. Load the kernel onto the target and run it. This is initiated with the command make run_kernel. The variable $TARGET provides the IP address of the ST Micro Connect to use.

2. Open a console on the target board. This can be done by connecting to the ST Micro Connect's telnet port. The first step is to configure telnet on the ST Micro Connect using stmconfig:
   /opt/STM/STMCR1.6.0/bin/stmcconfig –ip ¡MicroConnect_IP¿ –serial-relay 0115200 8 none 1 0 Starting serial relay : ip: ¡MicroConnect_IP¿ port: 5331

   Note: The ST Micro Connect telnet port must be reconfigured using the above command every time that the Micro Connect IP is restarted.

3. It is then possible to connect to the ST Micro Connect with telnet: telnet $< MicroConnect\_IP >$ 5331

4. When the kernel has booted without reporting any problems, it displays a login prompt. Enter the username root. There is no password.

5. The next step is to load the STAPI library modules. At the shell prompt, type: source /root/modules/load_modules.sh

As the modules are downloaded to the target using the ethernet port, this operation could take as much as ten seconds to complete.

If you want to run the STAPI-SDK application in a different telnet session, open and log in to that session, and then set up the environment with the command: source /root/modules/load_env.sh

Note: The location of the filesystem root on the host PC is determined by the environment variable KTARGET.

6. To run the STAPI application, execute the .out file that was created. For example, for an mb680 board with an STx710x SoC, this file is: /root/main_mb680 _710x_ST40_LINUX_32BITS.out When the application is running, it displays the testtool prompt.

An alternative method is to run STAPI in user space using gdbserver. In place of step 6. above, invoke gdbserver on the target with the following command: gdbserver $< HostMachineIP >$:5555 /root/main_mb411_710x_LINUX.out.

On the host machine, enter make debug. This attempts to connect to the gdbserver running on the platform.

Chapter 5

Testing of STB Frontend Drivers & Results

5.1 Hardware Setup

Figure 5.1 shows the general setup of the live streaming test where video stream has been fed to Set-Top-Box by satellite antenna through RF cable. User compiles the STB Frontend Drivers on Server PC remotely using PuTTy client where required environment for compilation and build has been setup. After compilation final .out file loaded to the Set-Top-Box development board through STmicroconnect which is a emulator for STB. UART connection has been used to see the ongoing activity log on STMicroconnect.

This chapter defines the test procedures carried out on **STFRONTEND** for testing the **FE** device functionality and verify performance of driver APIs. The test result obtained provides typical use case statistics for reference.

All the tests to be performed have a test harness within the stfrontend/tests sub-directory directories. Tests that are not applicable for particular devices are skipped

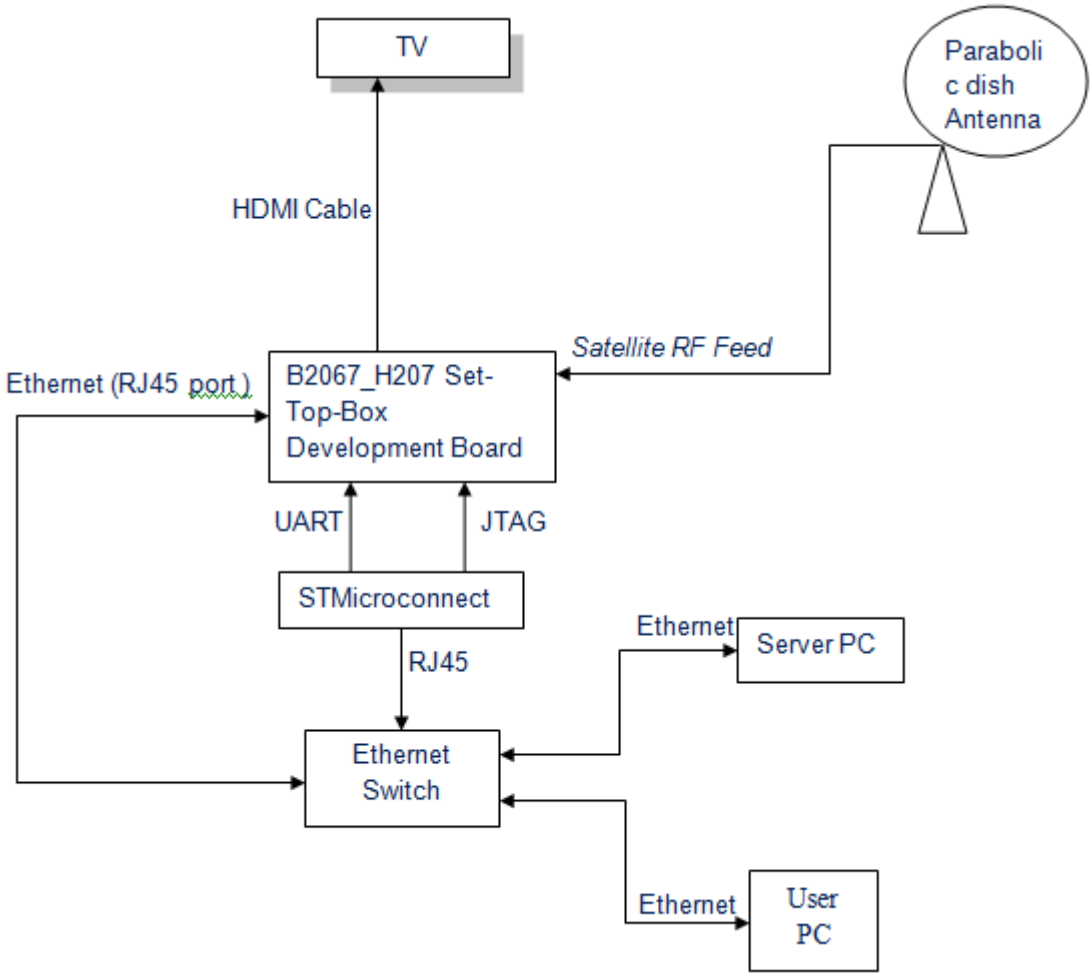

Figure 5.1: Live Streaming Setup

through inbuilt intelligence. The test harness also provides reference for typical use of APIs.

Target ID	Name of test function	Test Description	Acceptance Criteria
1	TunerAPI	Test the stability of STFRONTEND APIs and memory leak test	Test the stability of APIs by whether appropriate errors are passing invalid parametersand check returned.Memory pool to be unchanged after a series of function init,open etc..
2	TunerReLock	Relock channel test	Driver should relock afterthe input signal is reapplied after the short interval
3	TunerSetFrequency	Performs a typical run-through of the API calls such as initialize,open, attempt to scan across known Frequencies.	Scan to an exact Frequency and signal parameters
4	TunerTermTest	Termination during a scan	STFRONTEND_TERM should return ST_NO_ERROR
5	TunerTracking	Test tracking of a locked Frequency with option to change default duration of tracking	Each time the driver is polled the Front End status should indicate it is still locked
6	TunerBlindScan	BlindScan test to scan channels in the band	The scan should lock onto all the frequencies specified in thex high to low
7	TunerScan	Scan Test to acquire channels in band in both scan direction	The scan should look onto all the frequencies specified in the scan listing.The scan should obtain identical results for both scan directions i.e low to high or high to low

Table 5.1: Test List

5.2 Test & Results

5.2.1 SDK-Tree Test

SDK TREE Testing Procédure on OS21 in Linux Environnent

1. Open PuTTy
2. Write an IP of corrosponding Server E.g ==>10.199.x.xx , Press Open
3. Login as :
   Password :
4. Set view
   E.g==>[patelb1@dlhl0567 ~]$ ct setview « view name »

   (NEWSTAPISDK_RELEASE_43_INPUTS_patelb1_dv)
5. Go to bin directory where configuration file is located.
   ==>cd /vob/newmain/STAPISDK/bin
6. Go to Bash by typing
   ==>bash
7. Select required configuration
   ==>. Setenv.sh «Boad Name(B2067_H237)»
   Message :B2067_H237 configuration selected !
8. Go to Stapp directory for Compilation
   ==>cd ../stapp
9. Compilation commands
   ==>make purge_apilib (It clears the apilib(After changing Driver flag it is used to clear apilib))
   ==>make purge (it cleans only applications from stapp directory(use it only when changes in application flag has been occured))
   ==>make purge_all (It cleans all apilib,stapp,stdebug etc.)
   ==>make apilib (builds apilib)
   ==>make all (builds apilib,stapp,stdebug)
10. Make run (Boots Board)

Result after running different commands on board

1. Open PuTTy
2. Login :
   Password :
3. telnet 10.199.131.xx(Board IP) 5331(port)
4. After booting of board
   Message :

```
STVOUT_RECEIVER_CONNECTED to    VERSION  : Sep  3 2013 at 14:32:47
STVOUT_RECEIVER_INACTIVE
-----------------------------------------------------

STAPI_SDK Revision : STAPI_SDK-REL_42.2

Please press a key...
Using uart I/Os
input from file "macros.ttm"
STH205 >
```

STH205 > TUNER_TUNE 0 +557 MOD_64QAM FEC_7_8

```
[0x6adb0e83] : TUNER(EVT_TUNER0) : TUNER_EV_LOCKED
Tuner 0 is currently locked
Frequency (Khz) : 557000
Channel BW      : 8
Modulation      : MOD_64QAM
FEC Rate        : FEC_7_8
Status          : LOCKED
Quality (%)     : 026
BER             : 0
PLP Id          : 0
```

STH205 > demux_mux

```
+--------+---------+------+---------------+-----------------+-------------------+--                                          ----------
------+--------------+-------------+-----------------+-------------+
| Input | Output | Lock | Fifo Overflow | Buffer overflow | Nb Error Packets | Nb Short Packets | OutOfOrderRP | PktOverflow |
DMAPointerError | DMAOverflow |
+--------+---------+------+---------------+-----------------+-------------------+--                                          ----------
------+--------------+-------------+-----------------+-------------+
| TSIN0 | PTI0-0 | 0  |     0     |     0     |     0     |     0     |     0     |     0     |     0     |
| TSIN1 | PTI0-3 | 0  |     0     |     0     |     0     |     0     |     0     |     0     |     0     |
| TSIN2 | PTI0-4 | 0  |     0     |     0     |     0     |     0     |     0     |     0     |     0     |
| TSIN3 | PTI0-1 | 1  |     0     |     0     |     0     |     0     |     0     |     0     |     0     |
| NONE  | PTI0-2 | ?  |     ?     |     ?     |     ?     |     ?     |     ?     |     ?     |     ?     |
| NONE  | PTI0-5 | ?  |     ?     |     ?     |     ?     |     ?     |     ?     |     ?     |     ?     |
```

STH205 > table_pat 1

```
Program association table
-------------------------
Transport Stream Id = 0x1001
Number of programs  = 7
+----------------+--------+
| program_number |  pid   |
+----------------+--------+
| 0x0000         | 0x0010 |
```

```
| 0x10ff      | 0x10ff |
| 0x113f      | 0x113f |
| 0x117f      | 0x117f |
| 0x123f      | 0x123f |
| 0x1081      | 0x1081 |
| 0x1041      | 0x1041 |
+-----------------+--------+
```

STH205 > play_prog 0 "1" 113f

[0x6adc55bf] : VID_Init_Codec():HD Video(0) Profile selected
[0x6adc55c2] : PLAYRECi_SDK_Event_Callback(PLAY0):PLAYREC_EVT_COMMAND_COMPLETED - PLAYREC_CMD_PLAY_INIT - ErrCode=0x00000000 - Param1=0x00000000
--> Playback 0 Started - File name is "1" - Nb pids = 4
--> 0x113f - "BBC" - "BBC NEWS 24" - "Digital television service"
STH205 > [0x6adc5834] : PLAYRECi_SDK_Event_Callback(PLAY0):PLAYREC_EVT_COMMAND_COMPLETED - PLAYREC_CMD_PLAY_START - ErrCode=0x00000000 - Param1=0x00000000

STH205 > play_stop

--> Playback 0 Stopped

STH205 > POWER_SETMODE 0 1 PM_CPU_SLEEP PM_CPU_END PM_WAKEUP_TIMER 5

(Low Power or standby test)

System is entering in a power state mode...
CPU has been wake-up, reason is 0x00000020 !
Cpu has been wake-up by timer !
[0x6adee11c] : HDMIi_HotplugCallback(0): Changing state from STVOUT_IDLE to STVOUT_NO_RECEIVER
STH205 > [0x6adee489] : TUNER(EVT_TUNER0) : TUNER_EV_LOCKED

STH205 > tuner_scan 0 +557

Tuner 0 Scanning from 557000 to 858000 KHz ...
[0x6adfc482] : TUNER(EVT_TUNER0) : TUNER_EV_UNLOCKED
[0x6adfc755] : TUNER(EVT_TUNER0) : TUNER_EV_LOCKED
+ F=557000KHz - MOD=MOD_64QAM - FEC=FEC_7_8 - Status='LOCKED' - Quality=026%

[0x6ae07cc7] : TUNER(EVT_TUNER0) : TUNER_EV_SEARCH_FAILED
Scan finished ErrCode=393217 !

STH205 >Relock Test

To perform this test turn [Modulator Off / Feed disconnect] ---> Tuner_unlock
[Modulator On / Feed connect] --->Tuner_lock

Check on going activity on board by connecting with STmicroconnect

---

1. Open cmd promt
2. Stmcconfig - -ip 10.199.131.xx - -serial-relay
3. telnet 10.199.131.xx 5331 (STmicroconnect IP)

5.2.2 Unitary Test

Unitary test on OS21

1. Open cmd prompt
2. Go to view drive
   E.g :==>G:\>
3. Go to Setup directory where configuration file is laying
   ==>cd newmain/STAPISDK/apilib/src/stfrontend/test_sdk/setup
4. Select required configuration
   ==>ct_setup.bat «Boad Name(B2067_H237)»
   Message :B2067_H237 configuration selected !
5. Go to Stapp directory for Compilation
   ==>cd ../stapp
6. Compilation commands
   ==>make purge_apilib (It clears the apilib(After changing Driver flag it is used to clear apilib))
   ==>Make purge_modules (clear modules)
   ==>make purge (it cleans only applications from stapp directory(use it only when changes in application flag has been occured))
   ==>make purge_all (It cleans all apilib,stapp,stdebug etc.)
   ==>make apilib (builds apilib)
   ==>make modules (builds modules)
   ==>make all (builds apilib,stapp,stdebug)
   ==>Make run

RESULTS

Message:

```
------------------------------------------------------
  SDK - Software development kit debug environment
------------------------------------------------------

STAPI_SDK Revision : STAPI_SDK-REL_A27

********** Frontend_RegisterCommands**********

Please press a key...

Using console I/Os
input from file "macros.ttm"
STH205 >
```

STH205 > fe_probe (It detects the tuner on I2C bus on specific address **)**

```
STFRONTEND(12748808):
*************************************************************************************************
STFRONTEND(12770697):                    DETECTED TUNERS
STFRONTEND(12785627): Probed Address d0   on Bus I2C0    STI2C_ERROR_ADDRESS_ACK     Chip ID 0
STFRONTEND(12839378): Probed Address d6   on Bus I2C0    STI2C_ERROR_ADDRESS_ACK     Chip ID 0
STFRONTEND(12858366): Probed Address 38   on Bus I2C0  ST_NO_ERROR Chip ID 60
STFRONTEND(12873791): Found STV0367  tuner on bus 'I2C0' DemodAddress 38 ChipID 60
STFRONTEND(12890697): Configuring Initparams for STI7167CAB.....
STFRONTEND(12906797): Probed Address 3a   on Bus I2C0    STI2C_ERROR_ADDRESS_ACK     Chip ID 0
STFRONTEND(12924433): Probed Address c0   on Bus I2C0
*************************************************************************************************
STFRONTEND(13951768): Subscribing for events....
STFRONTEND(13962377): done. 1 event handles obtained
```

STH205 > fe_api

```
STFRONTEND(17098626): ------------------------- StfrontendAPI Test for STV0367 -------------------------
STFRONTEND(17118699): 17118715   STV0367 1.0: Open without init
STFRONTEND(17133691): 17133707   STV0367 Purpose: Attempt to open the driver without initializing it first.
STFRONTEND(17153492): 17153508   STV0367 STFRONTEND_Open() = ST_ERROR_UNKNOWN_DEVICE
STFRONTEND(17170957):              Result: **** PASS ****
STFRONTEND(17185064):
STFRONTEND(17196508): 17196524   STV0367 1.1: Term without init
STFRONTEND(17211083): 17211099   STV0367 Purpose: Attempt to term the driver without initializing it first
STFRONTEND(17230090): 17230105   STV0367 STFRONTEND_Term() = ST_ERROR_UNKNOWN_DEVICE
STFRONTEND(17247631):              Result: **** PASS ****
STFRONTEND(17261822):
STFRONTEND(17272966): 17272982   STV0367 1.2: Multiple inits
STFRONTEND(17287493): 17287508   STV0367 Purpose: Attempt to init the driver twice with the same name.
STFRONTEND(17305738): 17305758   STV0367 STFRONTEND_Init() = ST_NO_ERROR
STFRONTEND(17321592): 17321608   STV0367 STFRONTEND_Init() = ST_ERROR_BAD_PARAMETER
STFRONTEND(17338664):              Result: **** PASS ****
STFRONTEND(17351774):
STFRONTEND(17363428): 17363444   STV0367 1.3: Open with invalid device
STFRONTEND(17377579): 17377595   STV0367 Purpose: Attempt to open the driver with an invalid device name.
STFRONTEND(17396467): 17396482   STV0367 STFRONTEND_Init() = ST_ERROR_UNKNOWN_DEVICE
STFRONTEND(17414250):              Result: **** PASS ****
STFRONTEND(17427638):
STFRONTEND(17439110): 17439126   STV0367 1.4: Multiple opens
STFRONTEND(17453446): 17453462   STV0367 Purpose: Attempt to open the device twice with the same name.
STFRONTEND(17538818): 17538843   STV0367 STFRONTEND_Open() = ST_NO_ERROR
STFRONTEND(17555173): 17555190   STV0367 STFRONTEND_Open() = ST_ERROR_OPEN_HANDLE
STFRONTEND(17572493):              Result: **** PASS ****
STFRONTEND(17585356):
STFRONTEND(17597013): 17597029   STV0367 1.5: Invalid handle tests
STFRONTEND(17610947): 17610963   STV0367 Purpose: Attempt to call APIs with a known invalid handle.
STFRONTEND(17629256): 17629272   STV0367 STFRONTEND_Close() = ST_ERROR_INVALID_HANDLE
STFRONTEND(17647164):              Result: **** PASS ****
STFRONTEND(17660579): 17660596   STV0367 STFRONTEND_Ioctl() = ST_ERROR_INVALID_HANDLE
STFRONTEND(17678630):              Result: **** PASS ****
STFRONTEND(17691719): 17691735   STV0367 STFRONTEND_SetFrequency() = ST_ERROR_INVALID_HANDLE
STFRONTEND(17709584):              Result: **** PASS ****
STFRONTEND(17722835): 17722851   STV0367 STFRONTEND_Scan() = ST_ERROR_INVALID_HANDLE
STFRONTEND(17739729):              Result: **** PASS ****
STFRONTEND(17753091): 17753107   STV0367 STFRONTEND_GetTunerInfo() = ST_ERROR_INVALID_HANDLE
STFRONTEND(17770732):              Result: **** PASS ****
```

```
STFRONTEND(17785643): 17785663   STV0367 STFRONTEND_Close() = ST_NO_ERROR
STFRONTEND(17801770): 17801787   STV0367 STFRONTEND_Term() = ST_NO_ERROR
STFRONTEND(17817792):
STFRONTEND(17829501): ----------------------------------------------------------------------------------------------------
STFRONTEND(17850515):          Running total: PASSED: 10
STFRONTEND(17863961):                         FAILED: 0
STFRONTEND(17878021): ----------------------------------------------------------------------------------------------------
```

STH205 > fe_scan

```
STFRONTEND(21839332): 21839355   STV0367 STFRONTEND_Init() = ST_NO_ERROR
STFRONTEND(21922186): 21922208   STV0367 STFRONTEND_Open() = ST_NO_ERROR
STFRONTEND(21938680): 21938696   STV0367 2.0: Start Scan
STFRONTEND(21953433): 21953449   STV0367 Purpose: Scan to search channel in the band.
STFRONTEND(21970677): 21970693   STV0367 Start Frequency = 440000 End Frequency = 444000 Step Size = 1000
STFRONTEND(21989436): 21989454   STV0367 STFRONTEND_Scan() = ST_NO_ERROR
STFRONTEND(22005467): 22005483   STV0367 Scanning  forward...
STFRONTEND(27233037): 27233059   STV0367 Event = STFRONTEND_EV_LOCKED
STFRONTEND(28101024): 28101042   STV0367 STFRONTEND_GetTunerInfo() = ST_NO_ERROR
STFRONTEND(28117927): 28117944   STV0367 CH-0  F = 443999 KHz   Status = LOCKED       MOD =  64QAM
STFRONTEND(28135268): 28135284   STV0367 CH-0  SR = 6874883   IQ = 0   Q = 50  BER = 0 10E-6 RF = -72 PER=0
STFRONTEND(28153854): 28153870   STV0367 STFRONTEND_ScanContinue() = ST_NO_ERROR
STFRONTEND(28171123): 28171139   STV0367 Scanning  forward...
STFRONTEND(28186380): 28186396   STV0367 Event = STFRONTEND_EV_SEARCH_FAILED
STFRONTEND(28203127): 28203142   STV0367 ChannelCount = 1
STFRONTEND(28218335):              Result: **** PASS ****
STFRONTEND(28233004): 28233024   STV0367 STFRONTEND_Close() = ST_NO_ERROR
STFRONTEND(28248674): 28248691   STV0367 STFRONTEND_Term() = ST_NO_ERROR
STFRONTEND(28264729): 28264745   2.0: STFRONTEND Init()->Open()->Close()->Term() Test
STFRONTEND(28282041): ----------------------------------------------------------------------------------------------------
STFRONTEND(28303844):          Running total: PASSED: 11
STFRONTEND(28318224):                         FAILED: 0
STFRONTEND(28331970): ----------------------------------------------------------------------------------------------------
```

STH205 > fe_tune

```
STFRONTEND(236277915): Purpose: To lock channels using Setfrequency API
STFRONTEND(236294854): 236294875   STV0367 Dynamic memory free before start:  18488952 bytes
STFRONTEND(236312348): 236312366   STV0367 STFRONTEND_Init() = ST_NO_ERROR
STFRONTEND(236394257): 236394277   STV0367 STFRONTEND_Open() = ST_NO_ERROR
STFRONTEND(236410818): ---------- StfrontendSetFrequency Test for STV0367 ----------
STFRONTEND(237540119): 237540140   STV0367 STFRONTEND_GetTunerInfo() = ST_NO_ERROR
STFRONTEND(237557424): 237557441   STV0367 CH-0  F = 443999 KHz   Status = LOCKED       MOD =  64QAM
STFRONTEND(237575513): 237575529   STV0367 CH-0  SR = 6874883   IQ = 0   Q = 50  BER = 0 10E-6 RF = -72 PER=0
STFRONTEND(237594540): 237594557   STV0367 Event = STFRONTEND_EV_LOCKED
STFRONTEND(237609702): 237609719   STV0367 Time taken to lock & print info = 236430296 ms
STFRONTEND(237628099):              Result: **** PASS ****
STFRONTEND(237643629): 237643646   STV0367 STFRONTEND_Close() = ST_NO_ERROR
STFRONTEND(237658547): 237658565   STV0367 STFRONTEND_Term() = ST_NO_ERROR
STFRONTEND(237675299): 237675318   UNKNOWN Dynamic memory free after 1 cycles: 18488952 bytes
STFRONTEND(237693947): 237693963   UNKNOWN No memory leaks detected
STFRONTEND(237710345):              Result: **** PASS ****
STFRONTEND(237723740): ----------------------------------------------------------------------------------------------------
STFRONTEND(237744659):          Running total: PASSED: 32
STFRONTEND(237759600):                         FAILED: 0
STFRONTEND(237772557): ----------------------------------------------------------------------------------------------------
STH205 >
```

STH205 > fe_standby

```
STFRONTEND(45760780): Purpose: To lock channels using Setfrequency API
STFRONTEND(45776893): 45776911   STV0367 STFRONTEND_Init() = ST_NO_ERROR
STFRONTEND(45859704): 45859723   STV0367 STFRONTEND_Open() = ST_NO_ERROR
STFRONTEND(45876188): 45876205   STV0367 3.0: Test the StandByMode just after open
STFRONTEND(45892261): 45892277   STV0367 Purpose: Test the StandByMode just after open
STFRONTEND(45918021): ***** StandByMode is switched ON *****
STFRONTEND(45933235): 45933251   STV0367 STFRONTEND_GetStatus() = ST_NO_ERROR
STFRONTEND(45951181): 45951197   STV0367 Demod Status() = STANDBY
STFRONTEND(45966311):              Result: **** PASS ****
STFRONTEND(45979197): 45979214   STV0367 3.1: Resume from Standby
STFRONTEND(45994672): 45994688   STV0367 Purpose: Resume from Standby just after Open
STFRONTEND(46021325): ***** StandByMode is switched OFF *****
STFRONTEND(46036433): 46036449   STV0367 STFRONTEND_GetStatus() = ST_NO_ERROR
STFRONTEND(46054205): 46054221   STV0367 Demod Status() = IDLE
STFRONTEND(46070268):              Result: **** PASS ****
STFRONTEND(46083733): 46083749   STV0367 3.2: Try to lock device without standby on
STFRONTEND(46100516): 46100532   STV0367 Purpose: Try to lock device without standby on
STFRONTEND(47393028): 47393050   STV0367 STFRONTEND_GetTunerInfo() = ST_NO_ERROR
STFRONTEND(47410836): 47410852   STV0367 CH-0  F = 443999 KHz   Status = LOCKED       MOD =  64QAM
STFRONTEND(47430058): 47430073   STV0367 CH-0  SR = 6874883   IQ = 0   Q = 50  BER = 0 10E-6 RF = -72 PER=0
STFRONTEND(47448953): 47448969   STV0367 Event = STFRONTEND_EV_LOCKED
STFRONTEND(47464182):              Result: **** PASS ****
STFRONTEND(47476860): 47476877   STV0367 3.3: Try to lock device with standby on
STFRONTEND(47493420): 47493436   STV0367 Purpose: Try to lock device with standby on
STFRONTEND(47519911): ***** StandByMode is switched ON *****
STFRONTEND(47535147): 47535164   STV0367 STFRONTEND_SetFrequency() = ST_ERROR_DEVICE_BUSY
STFRONTEND(47555945):              Result: **** PASS ****
STFRONTEND(47569039): 47569057   STV0367 Now waiting for TimeoutEvent
STFRONTEND(141335893): 141335910   STV0367 Event = STFRONTEND_EV_TIMEOUT
STFRONTEND(141352449):               Result: **** PASS ****
STFRONTEND(141366057): 141366074   STV0367 STFRONTEND_GetStatus() = ST_NO_ERROR
STFRONTEND(141381815): 141381832   STV0367 Demod Status() = STANDBY
STFRONTEND(141396686):               Result: **** PASS ****
STFRONTEND(141410294): 141410311   STV0367 3.4: Get device status after resuming from Standby
STFRONTEND(141427848): 141427864     STV0367 Purpose: Test whether previous state is resumed after switching to
Normal Power Mode
STFRONTEND(141457486): ***** StandByMode is switched OFF *****
STFRONTEND(141602536): 141602556   STV0367 Event = STFRONTEND_EV_LOCKED
STFRONTEND(141617676): 141617692   STV0367 STFRONTEND_GetStatus() = ST_NO_ERROR
STFRONTEND(141634070): 141634086   STV0367 Demod Status() = LOCKED
STFRONTEND(142470466): 142470485   STV0367 STFRONTEND_GetTunerInfo() = ST_NO_ERROR
STFRONTEND(142488075): 142488092   STV0367 CH-0  F = 443999 KHz   Status = LOCKED       MOD =  64QAM
STFRONTEND(142506945): 142506962   STV0367 CH-0  SR = 6874883   IQ = 0   Q = 50  BER = 0 10E-6 RF = -72 PER=0
STFRONTEND(142525891):               Result: **** PASS ****
STFRONTEND(142545476): 142545494   STV0367 STFRONTEND_Unlock() = ST_NO_ERROR
STFRONTEND(142561795): 142561811   STV0367 Event = STFRONTEND_EV_UNLOCKED
STFRONTEND(142578047): 142578063   STV0367 STFRONTEND_GetStatus() = ST_NO_ERROR
STFRONTEND(142593929): 142593945   STV0367 Demod Status() = IDLE
STFRONTEND(142610100): 142610117   STV0367 STFRONTEND_Close() = ST_NO_ERROR
STFRONTEND(142625511): 142625529   STV0367 STFRONTEND_Term() = ST_NO_ERROR
STFRONTEND(142641853): ------------------------------------------------------------------------------------------
STFRONTEND(142664181):          Running total: PASSED: 18
STFRONTEND(142679281):                         FAILED: 0
STFRONTEND(142692073): ------------------------------------------------------------------------------------------
```

STH205 > fe_relock

```
STFRONTEND(149027010): 149027032   STV0367 STFRONTEND_Init() = ST_NO_ERROR
STFRONTEND(149109788): 149109809   STV0367 STFRONTEND_Open() = ST_NO_ERROR
STFRONTEND(149126303): ---------- StfrontendRelock Test for STV0367 ----------
STFRONTEND(149146538): 149146554   STV0367 Locking on CH-0 443999 KHz
STFRONTEND(149162734): 149162752   STV0367 STFRONTEND_SetFrequency() = ST_NO_ERROR
STFRONTEND(149282636): 149282655   STV0367 Event = STFRONTEND_EV_LOCKED
STFRONTEND(150150554): 150150572   STV0367 CH-0  F = 443999 KHz   Status = LOCKED       MOD =  64QAM
STFRONTEND(150167719): 150167736   STV0367 CH-0  SR = 6874883    IQ = 0   Q = 50  BER = 0 10E-6 RF = -72 PER=0
STFRONTEND(150185208): 150185225   STV0367 ChannelCount = 1
STFRONTEND(150197213): 150197230   STV0367 ....Please DISCONNECT feed....
STFRONTEND(158249696): 158249714   STV0367 Event = STFRONTEND_EV_UNLOCKED
STFRONTEND(158266330): 158266347   STV0367 STFRONTEND_GetTunerInfo() = ST_NO_ERROR
STFRONTEND(158284996): 158285013   STV0367 CH-1  F = 0      KHz   Status = UNLOCKED      MOD =  NONE
STFRONTEND(158302409): 158302426   STV0367 CH-1  SR = 0          IQ = 0   Q = 0  BER = 0 10E-6 RF = -1000 PER=0
STFRONTEND(158320129): 158320145   STV0367 ....Please RECONNECT feed....
STFRONTEND(163497100): 163497118   STV0367 Event = STFRONTEND_EV_LOCKED
STFRONTEND(164365001): 164365019   STV0367 STFRONTEND_GetTunerInfo() = ST_NO_ERROR
STFRONTEND(164383898): 164383914   STV0367 CH-1  F = 443999 KHz   Status = LOCKED       MOD =  64QAM
STFRONTEND(164401902): 164401919   STV0367 CH-1  SR = 6874883    IQ = 0   Q = 50  BER = 0 10E-6 RF = -72 PER=0
STFRONTEND(164425874): 164425891   STV0367 STFRONTEND_Unlock() = ST_NO_ERROR
STFRONTEND(164442121): 164442137   STV0367 Event = STFRONTEND_EV_UNLOCKED
STFRONTEND(164460118):               Result: **** PASS ****
STFRONTEND(164479954): 164479974   STV0367 STFRONTEND_Close() = ST_NO_ERROR
STFRONTEND(164494134): 164494150   STV0367 STFRONTEND_Term() = ST_NO_ERROR
STFRONTEND(164518077): ----------------------------------------------------------------------------------------------------
STFRONTEND(164543873):          Running total: PASSED: 19
STFRONTEND(164558594):                         FAILED: 0
STFRONTEND(164570354): ----------------------------------------------------------------------------------------------------
```

STH205 > fe_locktime

```
STFRONTEND(179535556): 179535577   STV0367 STFRONTEND_Init() = ST_NO_ERROR
STFRONTEND(179618136): 179618156   STV0367 STFRONTEND_Open() = ST_NO_ERROR
STFRONTEND(179634536): ---------- StfrontendTimedScanExact Test in single instance environment STV0367 -
----------
STFRONTEND(179658342): 179658360   STV0367 Locking on CH-0 443999 KHz
STFRONTEND(179674939): 179674956   STV0367 STFRONTEND_SetFrequency()
STFRONTEND(180097636): 180097655   STV0367 Event = STFRONTEND_EV_LOCKED
STFRONTEND(180113065): 180113081   STV0367 Time taken = 260 ms to lock to CH-0
STFRONTEND(180135352): 180135369   STV0367 STFRONTEND_Unlock() = ST_NO_ERROR
STFRONTEND(180151446): 180151462   STV0367 Event = STFRONTEND_EV_UNLOCKED
STFRONTEND(180167517):               Result: **** PASS ****
STFRONTEND(180294467): 180294485   STV0367 Event = STFRONTEND_EV_LOCKED
STFRONTEND(180309885): 180309901   STV0367 Time taken = 72 ms to lock to CH-0
STFRONTEND(180331625): 180331643   STV0367 STFRONTEND_Unlock() = ST_NO_ERROR
STFRONTEND(180348717): 180348734   STV0367 Event = STFRONTEND_EV_UNLOCKED
STFRONTEND(180365135):               Result: **** PASS ****
STFRONTEND(180476243): 180476260   STV0367 Event = STFRONTEND_EV_LOCKED
STFRONTEND(180491465): 180491481   STV0367 Time taken = 61 ms to lock to CH-0
STFRONTEND(180512864): 180512882   STV0367 STFRONTEND_Unlock() = ST_NO_ERROR
STFRONTEND(180529297): 180529313   STV0367 Event = STFRONTEND_EV_UNLOCKED
STFRONTEND(180545189):               Result: **** PASS ****
STFRONTEND(182006533): 182006550   STV0367 Average locking time after 10 locks = 93
STFRONTEND(182024032): 182024051   STV0367 STFRONTEND_Close() = ST_NO_ERROR
STFRONTEND(182039354): 182039372   STV0367 STFRONTEND_Term() = ST_NO_ERROR
STFRONTEND(182055293):
----------------------------------------------------------------------------------------------------
----------
```

```
STFRONTEND(182076708):          Grand Total: PASSED: 29
STFRONTEND(182091340):                FAILED: 0
STFRONTEND(182105011):
================================================================================
==========
```

STH205 > fe_tracking

```
STFRONTEND(197881514): 197881535   STV0367 STFRONTEND_Init() = ST_NO_ERROR
STFRONTEND(197964372): 197964390   STV0367 STFRONTEND_Open() = ST_NO_ERROR
STFRONTEND(197981162): ---------- StfrontendTracking Test for STV0367 ----------
STFRONTEND(197999274): 197999290   STV0367 Locking on CH-0 443999 KHz
STFRONTEND(198015191): 198015208   STV0367 STFRONTEND_SetFrequency() = ST_NO_ERROR
STFRONTEND(198134995): 198135012   STV0367 Event = STFRONTEND_EV_LOCKED
STFRONTEND(199712556):
------------------------------------------------------------
STFRONTEND(199738813): 199738829   STV0367 CH-0  F = 443999 KHz  Status = LOCKED      MOD =  64QAM
STFRONTEND(199756849): 199756865   STV0367 CH-0  SR = 6874883   IQ = 0   Q = 50  BER = 0 10E-6 RF = -72 PER=0
STFRONTEND(199776507): 199776523   STV0367 STFRONTEND_GetStatus() = ST_NO_ERROR
STFRONTEND(199792015): 199792032   STV0367 Demod Status() = LOCKED
STFRONTEND(201370194):
------------------------------------------------------------
STFRONTEND(231401635): 231401653   STV0367 STFRONTEND_Close() = ST_NO_ERROR
STFRONTEND(231417581): 231417598   STV0367 STFRONTEND_Term() = ST_NO_ERROR
STFRONTEND(231434643):
================================================================================
==========
STFRONTEND(231456080):          Grand Total: PASSED: 30
STFRONTEND(231470713):                FAILED: 0
STFRONTEND(231484596):
================================================================================
==========
```

Chapter 6

Automation of Test Framework in STB

6.1 Automated Testing

Automated product testing is widely used in the production of both hardware and software. It allows a sequence of tests to be run at the push of a button, and a log to be generated showing which tests passed and which failed. This log can then be reviewed manually or processed automatically to determine whether all the test results are as expected for the current state of the product.

Generally, the same tests can be run manually, but running these is usually a laborious and time-consuming process not to mention prone to human errors. A manual testing model makes testing on any level of scale very costly and time consuming. This is particularly so if the production process requires tests to be run frequently and repetitively, for instance for regression testing when changes are made to the product under test.

Automating a set of tests can be expensive in terms of the purchase of test equipment,

and also for the design and implementation of the tests themselves which are typically coded in software. However, the return-on-investment in a testing solution can be seen quite rapidly particularly if tests are run many times over. Test cases only have to be created once, and the incremental cost of each test run can become very low compared to the cost of a manual test run. Typically the design and implementation of tests themselves, which are coded in software, can mean an investment of resources at the set-up phase however after this time the benefits of full automation can be enjoyed.

It is important to remember however that the value of an automated test system will be affected by the degree of automation that it supports. If there remains a very large subset of tests which cannot be automated, then the cost advantages of having the automated test system are eroded. This will depend on the nature of the product under test and the sophistication of the automated test equipment being used. In cases where an appropriate subset of the tests can be automated (generally 70% or greater) and especially in cases where it is necessary to run the full set of tests very frequently, the argument in favour of investing in an automated testing system is compelling[4].

6.2 Automated Testing for Set-Top Box Integration

In S3 Group, we have completed several projects which involved the full software integration of a Set-Top Box (STB) for various Digital TV networks worldwide. Such projects are generally very complex and involve the integration of multiple software components from third-party software suppliers. Our preferred way of working in

such projects is to integrate early and then gradually build up the functionality and stability of the software. For a long time S3 Group has worked with an automated nightly build process which ensures that the integrity of the build is maintained at all times. This is particularly useful in projects where we are responsible for some of the main components of the build, for instance the low-level drivers. Having a nightly build verifies that any new code or bug fixes added to the head of development in our software repository have not broken the build.

S3 Group had a desire to take this nightly build to the next level, by not only building the software each night but also running it. An automated test system for STBs would allow this.The build is automatically generated by our Configuration Management system and could then be loaded into a STB in a test system which would then run through a sequence of tests, checking that the output of the box matches the result expected for each test case.

In the case of a STB, the minimum requirement for such a test system would be a device that allows Infra-Red (remote control) key-presses to be sent to the box under test, and collection of debug output from the box via the serial log. The log could then be examined the next morning for any abnormal events.

Such an automated testing system would be a huge step forward from a system that only provides integrity checking of the build itself, and would provide at least the ability to verify that the basic operation of the STB is unaffected by any recent code updates. However, it would fall a long way short of a fully automated testing system outlined in Section 1 above. Therefore, we in S3 GROUP began to consider what that fully automated test system might look like for a set-top-box product. In order to have any chance of achieving a suitable level of automation of a set of tests for a set-top box, some way of automatically capturing the video output of the box and comparing it to a pre-defined video capture would be essential. We realized that

we would also need some way of programming the set of test cases, so that the degree of flexibility required to run a full range of test cases for the product would be supported. Finally, to be able to simulate real-world duration tests (stress tests) of the box, we wanted to be able to run tests on several boxes at the same time, possibly quite a large number of boxes, rather than just on one single box[4].

6.3 Features of an Automated Testing System for STBs

In the previous section we mentioned some features of an automated testing system that are essential to provide a high degree of automation. However, there are many more features that would be required in order to achieve a high degree of automation of the test regime of a digital set-top box. The following features are those which S3 Group regard as mandatory for an automated testing environment and which have been included in S3 Groups StormTest STB solution discussed below:

6.3.1 Client/Server Architecture

A client/server model which supports physically separate tester and STB locations client can be anywhere with a good network connection.

6.3.2 Remote Control of Set-Top Box

Each box in the test rack must be individually controlled by means of an IR device attached to the box. The IR commands are sent to the box under software control by the client machine, where the test engineer has selected the test case(s) to be run

for each box. We also need to be able to remotely power cycle each box individually, since some test cases can require this.

6.3.3 Programmable Test Cases

The test system must be easily programmable using a scripting language that can be easily learned by a typical test engineer. This will allow the test engineer to create new test cases when this is required.

As mentioned above, this is essential to automate testing of the basic operation of the box.

6.3.4 Logging of serial output

Typically a set-top box, particularly during the integration phase, will deliver debug output via its serial port. This will often provide useful information when a fault condition occurs that will allow the developers to understand the cause of the problem. The test system must capture and store the serial log from each box in the rack during testing, for possible later analysis.

6.3.5 Image Analysis and OCR

It must be possible to capture full resolution images from the live video stream for each STB and compare it against either images taken from other boxes or reference images previously stored. The image comparison must be fast and dependable. UI text recognition through an integrated OCR engine is also required, and this should support the majority of worldwide languages.

6.3.6 Transport Independent

S3 GROUP is involved in the development of digital STBs for OEMs and Operators worldwide, consequently an automated test system will have to support cable, satellite and IPTV transport networks.

6.3.7 Offline Review Mode

Use of an automated test system means there is an increased quantity of test results which require human post-analysis. Provision of support to ensure that this can be done efficiently would be a very valuable feature of the automated testing system.

6.3.8 User Interface

The user of the system (typically a test engineer) should have a user interface on the client machine that allows at least the following:

- Start-up of sequences of tests on each slot in the rack individually
- Monitoring of the status of tests in progress and completed
- Screenshot comparison (for automatic detection of deviations from expected behaviour on the video output)
- Automatic motion detection on the video output (lack of motion may indicate a box under test has stopped responding to IR commands)

6.3.9 Configuration Management Integration

The testing system should integrate tightly with the Configuration Management system in use on the development/integration project so that it is, for instance, possible to configure the system to do a nightly build of the latest code and then load it in some of the boxes in the rack and run a chosen set of tests. These tests would most likely be a standard set of stress tests, so that project management continually has a

measure of the stability of the current build throughout the integration work of the project.

6.3.10 Integration with Defect Tracking system

The testing system should integrate tightly with whatever defect tracking database is in use in the STB integration project. In particular, it should be possible for the system to automatically raise or update a defect when a test is found to fail.

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

As Digital TV is becoming an emerging consumer electronics appliance and Set Top Box is the migration from analog to digital broadcasting. In this report basic but essential fundamentals of Set top box frontend system are discussed. Brief overview of Linux device driver, Linux kernel and device driver development has been given along with an example of simple device driver code. As development of Linux based hardware devices is on the center point of the current industry trends. we choose Set top box system as an application in which all drivers of set top box systems are designed and implemented under Linux kernel which results in low cost solution to customer.For successful launch of any application, validation and testing has to follow designing development. We also discussed system setup, software environment required for testing and debugging the STB system, and finally the testing procedure followed by results.

Manual testing of STBs is time-consuming, expensive and error prone and most routine STB testing can be automated. An obvious use of automated STB testing is for final QA cycles and approvals at the Operators site prior to product launch, but

significant cost savings and project efficiencies can also be generated by the correct use of automation throughout the STB integration phase. Additionally the automated test cases developed during STB integration can be re-used during testing at the Operators site as part of the final test cycles pre-launch and again in testing during legacy maintenance. The benefits of identifying and resolving problems as early as possible in a product development lifecycle are well known and applicable to STB integrations. Use of automated testing throughout the STB integration can greatly reduce time and cost and ultimately increase the end product quality.

7.2 Future Scope

As a future enhancement in driver development we can start our work from a clean base code systematically developed instead of an unverified open source code and extend more features to the base code at a higher abstraction level. Thus, out approach will improve various quality attributes of device drivers such as productivity, reliability, and extensibility.

To develop the GUIs of various test cases in software framework. An effective approach to automate software framework is to develop test cases in a way that we can optimize the testing time and which requires less human interaction while testing. By doing automation in various test report generation and management tools we can easily manage and handle Test report generated by automation process. So it provides the results of all testing process in format in which it has been demanded.

Publication List

1. Preeti Dewani,Ankit Prajapati and Bhavin Patel , *"BER Improvement with combination of FEC and MDC in Heterogeneous Environment"* TechWeek-2014 at STMicroelectronics Pvt. Ltd,Greater Noida,FEB24-28,2014

Abstract: In a heterogeneous environment, packet loss rate and packet length is varying. Data sent via communication channel is very much sensitive to losses. So, there is a need to recover the data at the receiver end even in the worst environment conditions by error correction and detection techniques. Today Forward error correction (FEC) is used as an approach meeting the requirements but there are some limitations of this technique i.e. its performance is not considerable when packet loss rate is higher and packet length is larger. MDC (Multiple Description Coding) is the one which outperforms FEC in that situations where FEC fails to perform well.

The performance of MDC is even better when HEVC (High Efficiency Video Coding) is used as a source coding standard. On considering this point, we have proposed a scheme in which HEVC is used as a prior stage of MDC and we use FEC as the standard encoder to implement MDC. In this scheme, three routes are defined for sending the packets, according to the environment conditions and available bandwidth the one which best suits will be selected.

As HEVC is used to support the demand of increasing bit rate for high resolution video by providing better coding efficiency which is the key point of using it in our scheme. Due to this, better video quality will be obtained for 4k x 2k resolution and even the problem with the heterogeneous environment will be solved.

By the results we have observed that at low bit rate performance of HEVC is better than H.264 in terms of PSNR and when packet length is large and packet loss rate is high then combination of MDC and FEC(a double or multiple description coding scheme) outperforms FEC(a single description coding scheme).

References

[1] Dr. P.C.Jain, *"Set-Top-Box"*, Document, STmicroelectronics Pvt. Ltd.(Internal)

[2] Dr.P.C.Jain, Sushil Dutt, Joydip Chaudhary, Smita Joshi, S.Ahmed, Ashutosh Alok, V.Mitra, *"Digital Satellite, Cable and Terrestrial Set-Top- Box with conditional Access System"*, Himachal Futuristic communication Ltd,Gurgaon

[3] Asim Kadav and Michael M. Swift, *"Understanding Modern Device Drivers* ASPLOS12,March3-7,2012

[4] Charlie OBrien, *"The Importance of Automated Testing in Set-Top-Box Integration"*, Report on Quarter 4, 2009, S3 Group 2009-2010

[5] Mr.Nabrun Dasgupta, *"STFRONTEND_BASICS Test Specification"*, STmicroelectronics Pvt. Ltd.(internal), Document 0.9.0, 2008