YAŞAR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

(MASTERS THESIS)

AUDIO CAPTURE AND PLAYBACK IN REAL-TIME EMBEDDED LINUX SYSTEM USING ARM POWERED i.MX53 BOARD

Thesis Author: Sa'id Muhammad SA'ID

Thesis Advisor: Prof. Dr. Mustafa GÜNDÜZALP

Department of and Electronic Engineering

Presentation Date: 13 June, 2014

Bornova, Izmir

June, 2014

This study titled "AUDIO CAPTURE AND PLAYBACK IN REAL-TIME EMBEDDED LINUX SYSTEM USING ARM POWERED i.MX53 BOARD" and presented as Master Thesis by Sa'id Muhammad SA'ID has been evaluated in compliance with the relevant provisions of Y.U. Graduate Education and Training Regulation and Y.U Institute of Science Education and Training Direction. The jury members below have decided for the defense of this thesis and it has been declared by consensus / majority of votes that the candidate has succeeded in his thesis defense examination dated 13/06/2014.

Jury Members:	**Signature:**
Head: …….....	……
Rapporteur Member: ……...……..	……
Member: ……..……	……

ABSTRACT

AUDIO CAPTURE AND PLAYBACK IN REAL-TIME EMBEDDED LINUX SYSTEM USING ARM POWERED i.MX53 BOARD

Sa'id Muhammad SA'ID

MSc. in Electrical and Electronic Engineering

Supervisor: Prof. Dr. Mustafa GÜNDÜZALP

June, 2014, 72 pages

In this thesis, audio capture and playback is achieved on embedded ARM system. Audio handling technique in conventional desktop computers is employed to handle audio operations in real-time on embedded systems with limited memory.

A Linux host system is used to develop a driver based on ALSA (Advanced Linux Sound Architecture), a software framework that provides an API (Application Programming Interface) for sound cards. A graphical user interface is developed using GTK (GIMP Tool Kit) for the driver operations. C programming language is used throughout the driver development. It is then cross compiled for the ARM processor embedded in a tablet machine.

The various tools used in coding, debugging and cross compiling are explained with clear description of their roles in the project. The project is summarized and suggestions for further work are also detailed.

Keywords: Embedded system, ALSA, GTK, ARM Processor, real-time, audio capture and playback

ÖZET

ARM i.MX53 KULLANARAK GÖMÜLÜ LINUX SISTEMINDE GERÇEK - ZAMANLI SES YAKALAMA VE OYNATMA

SA'ID, Sa'id Muhammad

Yüksek Lisans Tezi, Elektrik Elektronik Mühendisliği

Tez Danışmanı : Prof. Dr. Mustafa Gündüzal p

Haziran 2014, 72 sayfa

Bu tezde gömülü ARM sistemlerinde, ses yakalama ve oynatma gerçekleştirilmiştir. Ses işleme tekniği, sıradan masaüstü bilgisayarlar kullanılarak, gerçek zamanlı olarak sınırlı hafızalı gömülü sistemlerde ses işlemek için kullanıldı.

Ses kartları için arayüz sağlayan, sürücü tabanlı ses mimarisi (ALSA) geliştirmek için Linüx host sistemi kullanıldı. Sürücü işlemleri için, GTK (GMIP Tool kit) kullanarak kullanıcı arayüzü geliştirildi. Sürücünün geliştirilmesi aşamasında C dili kullanıldı.Daha sonra tablet makine kullanarak, ARM işlemci için derlendi.

Kodlama, hata ayıklama, çapraz derlemede kullanılan türlü metodlar,projedeki roller ile birlikte açıklandı.Proje özetlendi ve ilerideki çalışmalar için öneriler anlatıldı.

Anahtar Kelimeler : Gömülü sistem , ALSA , GTK , ARM İşlemci , gerçek zamanlı ses yakalama, gerçek zamanlı ses oynatma

ACKNOWLEDGEMENT

I would like to express my profound gratitude to my supervisor and academic advisor, Prof. Dr. Mustafa Gündüzalp who despite his tight schedules has tirelessly provided me with his support and guidance throughout this research. He continually and convincingly conveyed a spirit of patience and hard work despite numerous obstacles we overcame.

I will also like to thank all staff and faculty members especially Asst. Prof. Dr. Erginer Ungan, Prof. Dr. Coşkun İsci, and Res. Assist. Gökhan Demirkıran, Asst. Prof. Dr. Nalan Ozkurt who was also part of the defense jury with my supervisor and Dr. Tuncay Ercan, among many others I may not be able to mention, for their advises and assistance in various ways.

This work wouldn't have been possible without the great assistance I enjoyed from KENTKART Research and Development Unit, who left all their resources at my disposal during the research. I will like to personally thank Veli Demirel and Ercüment Türk for their technical support.

Above all, I thank and praise God for his sustenance and blessings which I enjoy every day, best of which He bless me with the best parents and family I can ever wish for. My deepest appreciation goes to my parents whose prayers, guidance and support of all kinds are beyond mention.

Finally, my gratitude also goes to all friends around me here in İzmir and beyond for their loving company. To everyone, I say a big thank you. Without you all, this work wouldn't have been possible.

TEXT OF OATH

I declare and honestly confirm that my study titled “Audio Capture and Playback in Real-Time Embedded Linux System using ARM Powered i.MX53 Board**”** and presented as a Master’s Thesis, has been written without applying to any assistance inconsistent with scientific ethics and traditions, that all sources from which I have benefitted are listed in the bibliography, and that I have benefitted from these sources by means of making references.

13/06/2014

Sa’id Muhammad SA’ID

TABLE OF CONTENTS

TABLE OF CONTENTS (continue)

TABLE OF CONTENTS (continue)

INDEX OF FIGURES

INDEX OF ABBREVIATIONS

ALSA	Advanced Linux Sound Architecture
API	Application Programming Interface
ARM	Advanced RISC Machine
GCC	Gnu Compiler Collection
GIMP	Gnu Image Manipulation Program
GTK	GIMP Tool Kit
ISA	Industry Standard Architecture
MIDI	Musical Instrument Digital Interface
NFS	Network File System
OS	Operating System
OSS	Open Sound System
PCI	Peripheral Component Interconnect
PCM	Pulse Code Modulation
RISC	Reduced Instruction Set Computing
RTOS	Real Time Operating System

CHAPTER 1: INTRODUCTION

Importance of sound in computer systems cannot be over emphasized. It is therefore relevant to dwell into audio application in the growing field of real-time embedded systems.

Sound, consisting of waves of varying air pressure, is converted to its electrical form by a transducer, such as a microphone. An analog-to-digital converter (ADC) converts the analog voltages into discrete values called samples at regular intervals in time, known as the sampling rate. By sending the samples to a digital-to-analog converter and an output transducer, such as a loudspeaker, the original sound can be reproduced. This is performed on a computer system by the sound card or any chip with audio processing capability.

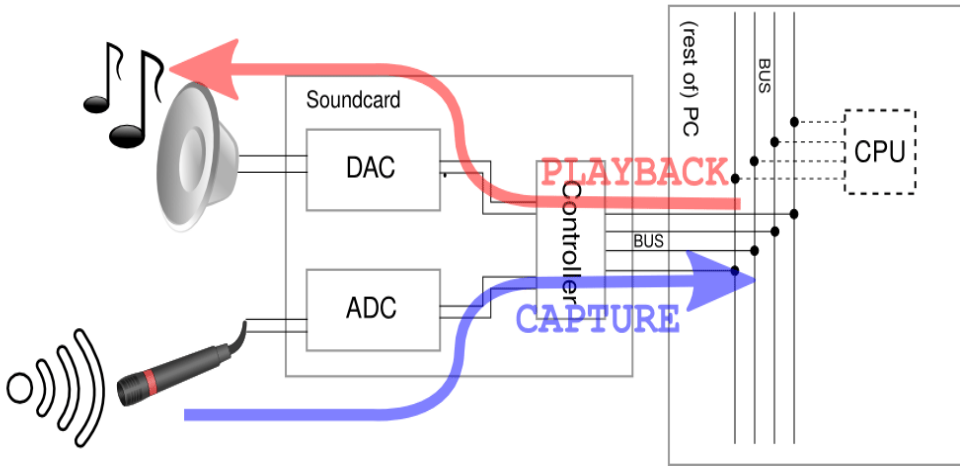

Figure 1.1 Sound card driver concepts (alsa-project.org August 2010)

In digital audio, the size of the samples, expressed in bits, is one factor that determines how accurately the sound is represented. The other major factor affecting sound quality is the sampling rate. [8]

Many embedded systems do not have audio capability other than emitting beep sound. Even the very graphic-rich avionics and factory automation systems don't have major sound output, except maybe in terms of decibel level. Sound-rich embedded systems are, however, becoming more and more popular with the proliferation of consumer- and service-oriented devices. Consumer-oriented devices feature complex audio and video codec support—including MP3, Ogg Vorbis, AAC, MPEG, MPEG4, and H264—and demand good support for audio, that is the capability to multiplex multiple audio streams out to the same device simultaneously in real-time.

Linux, however, was never designed to accommodate sound. Over the years, a number of schemes appeared to provide support for sound devices. These include the previously dominant Open Sound System (OSS) which allows applications like media players or web browsers to access the audio driver directly, and later the Advanced Linux Sound Architecture (ALSA) that has replaced it. [3]

ALSA have better hardware support. It allows improved user control and better support for multiple audio devices.

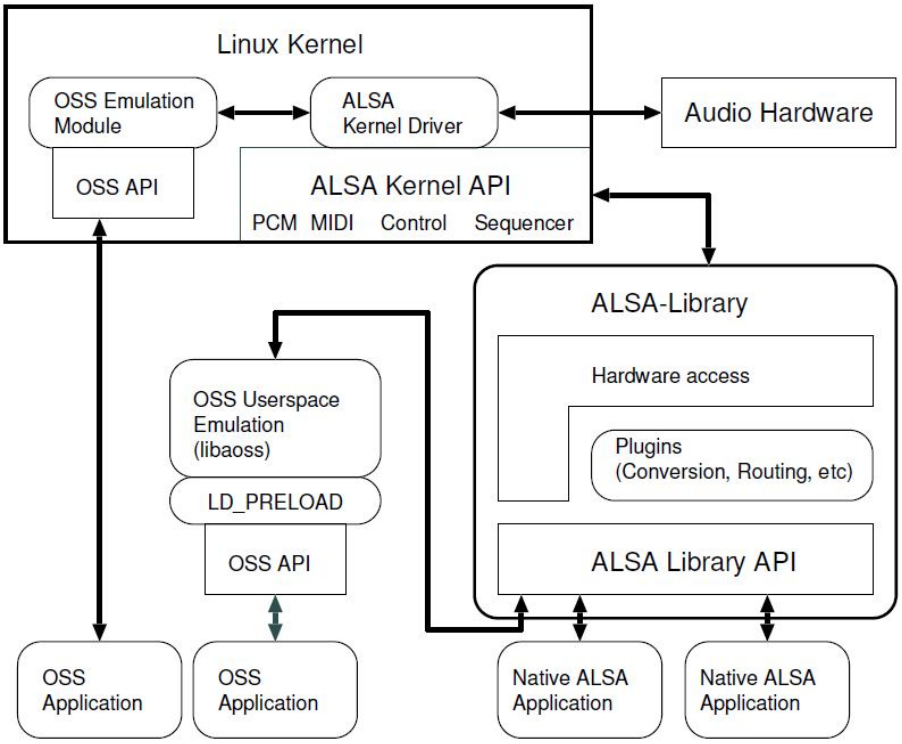

Figure 1.2 Basic Structure and Flow of ALSA System (Takashi, 2003)

Sound implementation on Linux is always a complex task with variant hardware and software demands. Documentation remains scanty with regards to audio implementation on Linux. With the increasing demand of embedding computers into devices, it is important that audio implementation is well understood. Accomplishing this in real-time on embedded Linux system will provide a clear understanding of the Linux sound system.

In this research, an ALSA driver will be developed using Eclipse C Development Tool for both coding and debugging. It shall then be cross-compiled for an ARM machine. A Gnu Image Manipulation Program (GIMP) tool kit, the GTK shall be used to design the user interface for which the driver control will be implemented graphically.

Due to lack of published resources in the on-going research area, a lot of on-line publications, forums and blogs shall be utilized for information and guides during the cause of the project. However, only trusted sources will be used and shall be well referenced at the end of the report.

1.1 SCOPE

The aim of this research is to exploit the mechanism involved in Linux desktop sound architecture and develop a similar more composite sound capture and playback capability in real-time embedded systems using ALSA. Both the device driver and the audio application will be designed and implemented.

1.2 OUTLINE

The outline of this project shall be as follows. Definition and evolution of real time and embedded systems shall be given in chapter two. The basic utility tools used like ALSA, GTK and Network File System (NFS) will also be explained.

Chapter three will explain the procedure of setting up the Linux system for development of the project. It shall also entail details of setting up the NFS server and client, mounting drives, installing GTK repositories, including ALSA libraries and running some basic applications on the various platforms.

In chapter four, details of developing an ALSA driver for the ARM system will be described. The process of coding in Eclipse CDT, the debugging and cross compilation will all be detailed out. Transferring the compiled program to the

client machine over NFS shall be detailed also. Finally, chapter five will summarize the entire project. Recommendations for further works shall also be given at the end.

CHAPTER 2: LITERATURE REVIEW

2.1 EMBEDDED SYSTEMS

An embedded system is a special-purpose computer which is completely encapsulated by the device it controls to increase intelligence and provide better functionality and efficiency. An embedded system development involves both the software and hardware co-development.

Unlike general-purpose computers, embedded systems have specific requirements and perform pre-defined tasks. They can satisfy the strict requirement of functionality, reliability, cost, volume, and power consumption of particular applications. They are integrated into technical environments intended to execute internal functions as responses to certain input values and process technical information.

2.2.1 Embedded system characteristics

Embedded systems come in a variety of shapes and sizes, from the largest multiple-rack data storage or networking powerhouses to tiny modules such as personal MP3 player or cellular handset. Following are some of the usual characteristics of an embedded system: [6]

• Contains a processing engine, such as a general-purpose microprocessor.

• Typically designed for a specific application or purpose.

• Includes a simple (or no) user interface, such as an automotive engine ignition

controller.

• Often is resource-limited. For example, it might have a small memory footprint and no hard drive.

• Might have power limitations, such as a requirement to operate from batteries.

• Not typically used as a general-purpose computing platform.

• Generally has application software built in, not user-selected.

• Ships with all intended application hardware and software pre-integrated.

• Often is intended for applications without human intervention.

Figure 2.1: Some devices with embedded processors

2.1.2 Bootloader

A unique feature of an embedded system is its startup. For a desktop computer, when power is first applied, a piece of software known as the BIOS is loaded, which has basic knowledge of the hardware, and also initializes the activities of the processor. However, in embedded systems, this routine is carried out by the bootloader.

The bootloader initializes major hardware components like the memory and I/O controllers, allocate system resources to peripheral controllers and provide mechanism for locating and loading operating system image.

2.2 REAL TIME OPERATING SYSTEM

Operating system is any software that manages computer hardware. It basically makes the hardware works. Common examples of operating systems are Windows, Linux and Mac OS. Operating systems intended to serve real-time application request are real-time operating systems (RTOS)

RTOS processes data quickly enough so that further action can be taken based on the results; therefore, scheduling, resource allocation, interrupt handling, real-time response, multitasking and predictability are key features for its operation.

2.3 LINUX ON EMBEDDED SYSTEMS

Linux is a free and open source operating system that runs on the Linux kernel. Operating systems based on the Linux kernel are mostly used in embedded systems due to their versatility. There are multiple suppliers for software development and support. Also free licensing, stable kernel and ability to read, modify and redistribute source code also make Linux more desirable.

2.4 AUDIO AND LINUX AUDIO DRIVERS

There are two basic components which build the sound system: the sound device driver and the sound server. The former is the hardware abstraction in the

lower level, while the latter gives more high-end capabilities like multiplex access and mixing. In other OS like Windows, the boundary between these two components is not clear. The driver does some heavy jobs like mixing in the kernel, too. On the Linux system, however, these are regarded still separately.

In earlier computers sound cards were usually ISA (Industry Standard Architecture) cards that handle 16 bit stereo PCM (pulse code modulation) playback and capture. However, lack of multiple channels and digital I/O interfaces led to the migration to PCI (Peripheral Component Interconnect). PCI have versatility especially with regards to higher level audio processing, therefore the need to go beyond the OSS arise.

Figure 2.2: ISA sound card

The core part of the sound system is the sound device drivers. On the Linux kernel, the OSS (Open Sound System) has been employed as the standard sound

driver. With the variation of compressions used for different sound formats, the OSS has lots of limitations regarding audio hardware support. This paved way for the development of the Advanced Linux Sound Driver (ALSA) which operates most PCI cards.

Figure 2.3: PCI Sound Card

2.5 ALSA

The Advanced Linux Sound Architecture (ALSA) is the audio framework used in newer (current) Linux kernels for audio support. It consists of a set of kernel drivers, an application programming interface (API) library and utility programs for supporting sound in Linux. It is backwards compatible with the older OSS. [2]

The ALSA Project was started because the sound drivers in the Linux kernel (OSS) were not being maintained actively and were lagging behind the capabilities of new sound technology. Jaroslav Kysela, who previously had written a sound card driver, started the project. Over time, more developers joined, support for

many sound cards was added and the structure of the API was refined.

During development of the 2.5 series of Linux kernel, ALSA was merged into the official kernel source. With the release of the 2.6 kernel, ALSA became part of the stable Linux kernel and is widely used. [8]

2.5.1 ALSA basics

ALSA consists of a series of kernel device drivers for many different sound cards (or sound chips in case of smaller and or embedded systems), and it also provides an API library, **libasound**, which provides a higher-level and more developer-friendly programming interface.

In contrast, OSS drivers are programmed at the kernel system call level and require the developer to specify device filenames and perform many functions using **ioctl** calls. For backward compatibility, ALSA provides kernel modules that emulate the OSS sound drivers, so most existing sound applications continue to run unchanged. An emulation wrapper library, **libaoss**, is available to emulate the OSS API without kernel modules.

ALSA has a capability called plugins that allows extension to new devices, including virtual devices implemented entirely in software. ALSA provides a number of command-line utilities, including a mixer, sound file player and tools for controlling special features of specific sound cards/chips. [8]

2.5.2 ALSA architecture

The ALSA API can be broken down into the major interfaces it supports:

- Control interface: a general-purpose facility for managing registers of sound cards/chips and querying the available devices.
- PCM interface: the interface for managing digital audio capture and playback.
- Raw MIDI interface: supports MIDI (Musical Instrument Digital Interface), a standard for electronic musical instruments. This API provides access to a MIDI bus on a sound card.
- Timer interface: provides access to timing hardware on sound cards used for synchronizing sound events.
- Sequencer interface: a higher-level interface for MIDI programming and sound synthesis than the raw MIDI interface. It handles much of the MIDI protocol and timing.
- Mixer interface: controls the devices on sound cards that route signals and control volume levels. It is built on top of the control interface.

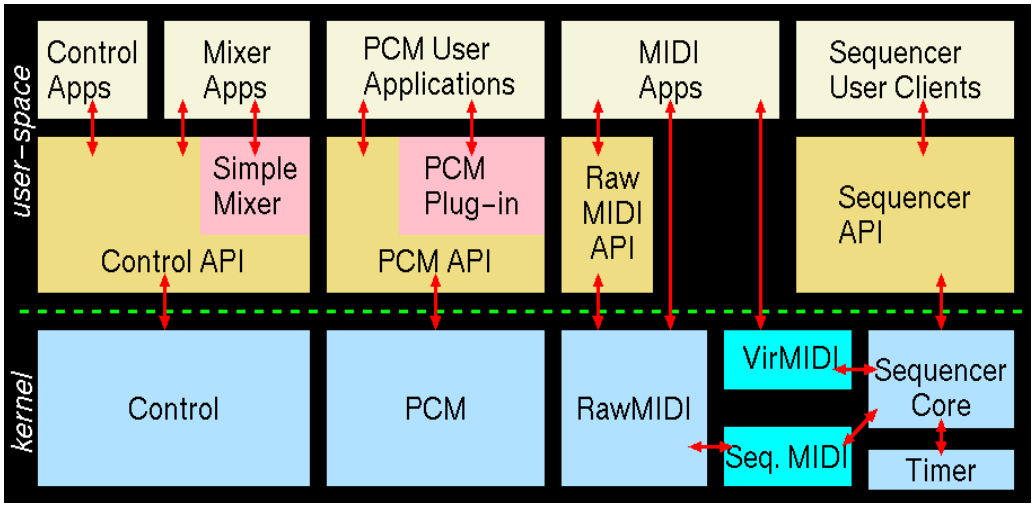

Figure 2.2: Structure of ALSA [9]

2.6 GTK

GTK (GIMP Toolkit) is a free licence library for creating graphical user interfaces. GTK is authored by Peter Mattis, Spencer Kimball and Josh MacDonald. It's called the GIMP toolkit because it was originally written for developing the GNU Image Manipulation Program (GIMP), but GTK has now been used in a large number of software projects, including the GNU Network Object Model Environment (GNOME) project. GTK is built on top of GDK (GIMP Drawing Kit) which is basically a wrapper around the low-level functions for accessing the underlying windowing functions (Xlib in the case of the X windows system).

GTK is essentially an object oriented Application Programmer Interface (API). Although written completely in C, it is implemented using the idea of classes and callback functions (pointers to functions). It also has a third component called GLib which contains a few replacements for some standard calls, as well as some additional functions for handling linked lists, etc. [15]

2.7 GNOME TERMINAL

GNOME Terminal is a terminal emulator for the GNOME desktop environment that allow users to execute commands using a real UNIX shell while remaining on graphical desktop. The GNOME Terminal comes pre-installed with GNOME desktop installation like Ubuntu.

The terminal emulator is handy in Linux OS. Most activities in this project were done through the terminal emulator. It provides easy access to file systems,

directories, and other activities with root user permissions.

2.8 SETTING UP THE HOST AND GUEST MACHINES

The host computer is an HP655 64-bit AMD laptop. It has 64-bit windows 7 OS. Since all platforms for the project has to be Linux, a 64-bit Ubuntu 12.04 LTS was installed to run alongside Windows. The choice of Ubuntu 12.04 is due to its stability and Long Time Support (LTS) as well as regular system and security updates available like in no other Linux OS before.

After Ubuntu installation, the following steps are followed to restore GRUB (Grand Unified Bootloader) for dual booting option menu. GRUB is a universal boot-loader for Linux distributions which allows multiple operating systems accessibility.

The terminal was accessed as root user after booting the machine with a USB flash drive containing Ubuntu LTS. The following commands to mount the drive containing the Linux OS, set it as default bootloader partition and install grub for boot menu option are ran sequentially.

```
sudo su

fdisk –l

mount /dev/sda1 /mnt

mount --bind /dev /mnt/dev

mount --bind /proc /mnt/proc
```

```
cp /etc/resolv.conf /mnt/etc/resolv.conf

chroot /mnt

grub-install --recheck /dev/sda

reboot
```

After reboot, the dual boot option appears and both Ubuntu and windows can be accessed.

2.9 SOFTWARE AND ACCESSORIES

For the development of this project, some software and accessories will be utilized. Some are just for connectivity like terminal console, other for code development, and some for cross compiling for the various platforms to be used.

2.9.1 PuTTY

PuTTY is free software for terminal emulation, serial console and network file transfer. PuTTY is open source software developed by Simon Tatham and supported by a group of volunteers.

PuTTY is used in this project as serial console to access the ARM machine from the host computer.

2.9.2 Eclipse CDT (C/C++ Development Tooling)

The Eclipse CDT project, a product of the Eclipse Foundation is a platform that provides a fully functional C and C ++ Integrated Development Environment. Both ALSA and GTK programs development will be carried out on Eclipse.

Debugging and cross compilation are also features of Eclipse to be utilized.

2.9.3 GCC

The GNU Compiler Collection, originally the Gnu C Compiler, is a compiler system produced by the GNU Project supporting C, C++, Fortran, Java, Ada and various other programming languages.

GCC has been ported to a wide variety of processor architecture. Therefore the GCC library will be used in the project to cross-compile the C program developed in the Eclipse CDT for running on the targeted ARM machine.

2.9.4 Network File System (NFS)

Network File System developed by Sun Microsystems in the 1980s is a means for UNIX to share files and applications across a network. It allows file sharing between computers running different UNIX variants, as well as UNIX systems connected to non-UNIX systems such as windows machines. [7]

NFS allows the attachment of a remote drive or directory to a virtual file system and work with it as if it were a local drive. The computer 'sharing' the directory is called the server and the computers or devices connecting to that server are called clients. The client mounts the shared directory making it part of its own directory structure.

In this research work, NFS is used to share the directory in which the C program is crossed compiled in the host computer, which is the workspace directory of Eclipse CDT so that it can be accessed directly from the client which

is an ARM powered Tablet machine.

2.10 ARM PROCESSOR ARCHITECTURE

ARM is a processor architecture based on reduced instruction set computing (RISC). It was developed by a British company, ARM Holdings.

A RISC-based computer design approach means ARM processors require significantly fewer transistors than typical CISC x86processors in most personal computers. This approach reduces costs, heat and power usage. These are desirable traits for light, portable, battery-powered devices—including smart phones, laptops, tablet and notepad computers, as well as other embedded systems. A simpler design facilitates more efficient multi-core CPUs and higher core counts at lower cost, providing improved energy efficiency. All these are perfect features for an embedded system which explains the wide usage of ARM processors in most embedded devices.

2.11 i.MX53

I.MX53 is a SoC (System on Chip) processor for multimedia application based on ARM Cortex-A8 core. It is optimized for both performance and power to meet the demands of high-end, advanced applications. It includes an integrated display controller, full HD capability, enhanced graphics and connectivity features.

Figure 2.5: i.MX53 Powered Tablet

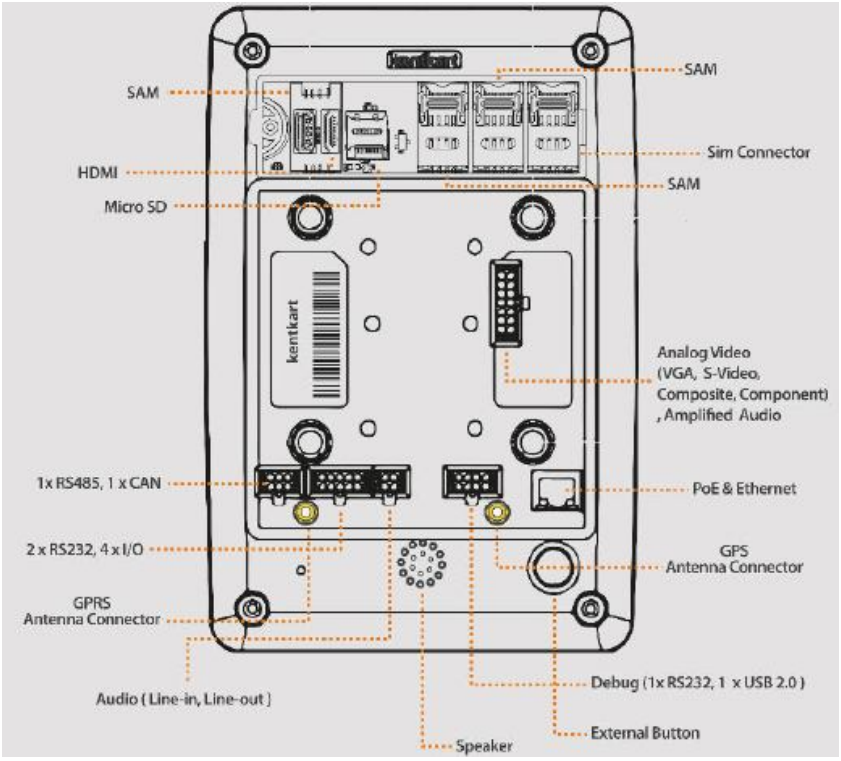

Figure 2.6: Posterior outline of i.MX53 Tablet machine

CHAPTER 3: SOFTWARE UTILITIES

3.1 SETTING UP AN NFSv4 SERVER

The host machine is a HP 655 64-bit laptop computer run by Linux, Ubuntu 12.04 LTS. The client is a tablet machine with an ARM Cortex A8 processor and a Linux OS.

3.1.1 NFSV4 server

The required packages available through debian file management tool are installed through terminal with root permission.

```
apt-get install nfs-kernel-server
```

This will install NFS server on the host machine, a laptop running Ubuntu 12.04LTS.

NFSv4 exports exist in a single pseudo filesystem, where the real directories are mounted with the --bind option.

To export the home directory of the host system, the export file system is first created.

```
mkdir –p /export/users
```

The real users directory is mounted with

```
mount –bind /home/users /export/users
```

To avoid retyping this after every reboot, the following line is added to /etc/fstab

```
/home/users /export/users none bind 0 0
```

Both /export and /export/users directories must have 777 permissions so that the NFS could be accessed from the client without LDAP/NIS authentication. [7]

3.1.2 File permissions

Files and directories in Linux system are assigned access rights to the file owner or the members of a group of related users. Rights can be assigned to read a file, write to a file, and to execute a file (i.e., run the file as a program).

The following command shows the permission settings for a file sample-gtk

```
said@said-VirtualBox:~$ ls –l sample-gtk
-rwxrwxr-x 1 said said 7491 May 8 16:20 sample-gtk
said@said-VirtualBox:~$
```

This shows that the file sample-gtk is owned by user “said” and group “said”. Both the user and the group have right to read (r), write (w) and execute (x) this file. Everyone else can only read or execute the file without write permission.

The permission settings are series of bits with each r, w or x in the series as a binary digit.

For the sample-gtk file, it is translated as

```
rwx  rwx  r-x

111  111  101

7    7    5
```

Both /export and /export/users directories have 777 permissions so that the NFS could be accessed from the client without LDAP/NIS authentication.

3.1.3 Changing file/directory permission

The chmod (change mode) command is used to change the permissions of a file or directory.

```
chmod 777 <file/directory>
```

3.1.4 Superuser access

The substitute user command su gives superuser (root) access which grants permissions to access files, directories and applications without restriction. However, password is required to get into the su mode. To exit the superuser mode, exit command suffices

In some distributions like Ubuntu, the sudo command is used instead, for one or more users when the need arises. A command is preceded by the su command and a user's password request follows. However, it is better to change the file permission permanently with chmod.

3.1.5 Mounting the NFS directory on the client machine

The /export/user directory is mounted on the client, the tablet machine.

First, the IP address of the host machine is looked up with ifconfig command.

This returns the result

```
said@said-VirtualBox:~$ ifconfig

eth0      Link encap:Ethernet HWaddr 08:00:27:12:49:c9

          inet      addr:10.0.2.15      Bcast:10.0.2.255
          Mask:255.255.255.0

          inet6     addr:     fe80::a00:27ff:fe12:49c9/64
          Scope:Link

          UP  BROADCAST  RUNNING  MULTICAST  MTU:  1500
          Metric:1

          RX  packets:100  errors:0  dropped:0  overruns:0
          frame:0

          TX  packets:165  errors:0  dropped:0  overruns:0
          carrier:0

          collisions:0 txqueuelen:1000

          RX bytes:33555 (33.5 KB) TX bytes:19789 (19.7
          KB

Lo        Link encap:Local Loopback
```

```
inet addr:127.0.0.1  Mask:255.0.0.0

inet6 addr: ::1/128 Scope:Host

UP LOOPBACK RUNNING MTU:16436 Metric:1

RX packets:58 errors:0 dropped:0 overruns:0
frame:0

TX packets:58 errors:0 dropped:0 overruns:0
carrier:0

collisions:0 txqueuelen:0

RX bytes:5228 (5.2 KB)  TX bytes:5228 (5.2 KB)
```

Using the system IP returned above, that is 10.0.2.15; the drive is mounted with the command

Mount 10.0.2.15:/export/users /mnt/data -o rsize=1024, wsize=1024, intr, nolock

Mount This will mount the directory /export/users from the host machine with the network inet address 10.0.2.15 on /mnt/data of the client

-o Options are specified with a -o flag followed by a comma-separated string of options. The above command will mount the drive with the following options

The mount command options rsize and wsize specify the size of the chunks of data that the client and server pass back and forth to each other. If no rsize and

wsize options are specified, the default varies by which version of NFS we are using. [13]

rsize rsize=n The number of bytes NFS uses when reading files from an NFS server. The default value is dependent on the kernel, currently 1024 bytes. rsize=1024

wsize wsize=n The number of bytes NFS uses when writing files to an NFS server. The default value is dependent on the kernel, currently 1024 bytes. wsize=1024.

Intr If an NFS file operation has a major time-out and it is hard mounted, then allow signals to interrupt the file operation and cause it to return EINTR to the calling program. The default is to not allow file operations to be interrupted. [1]

3.2 GTK+ 3.12 INSTALLATIONS AND UPDATE

GTK+ requires Glib, GObject-Introspection, Pango, Gdk-Pixbuf, GTK and GTK+ packages to build. However, binary packages of GTK+ prebuilt for Ubuntu are available. This gets programming with GTK+ much faster than building it from scratch. GTK+ 3.12 (the latest version available) is installed via terminal as

```
sudo apt-get install libgtk-3-dev
```

This installs GTK library files (header etc) in **/usr/include/gtk-3.0/gtk** directory. Other linkedfiles are searched in Ubuntu Packages website whenever necessary. [17]

On UNIX-like systems like Ubuntu, GTK+ uses the standard GNU build system, using **autoconf** for package configuration and resolving portability issues, **automake** for building makefiles that comply with the GNU Coding standards, and **libtool** for building shared libraries on multiple platforms.

3.2.1 GTK programming: C- coding

A simple program to generate an empty 200x200 pixel window is used to test the library and cross compilation of GTK.

The code is written in a text editor and saved as **sample-gtk.c**

```
#include <gtk/gtk.h>

int

main (int   argc,

char *argv[])

{

  GtkWidget *window;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

  gtk_widget_show (window);

  gtk_main ();

  return 0;
```

```
}
```

The GTK header **gtk/gtk.h**, which declares functions, types and macros is included for the compiler and proceeds to the **main()** function of the application, a window variable is declared as a pointer of type **GtkWidget**.

The following line will call **gtk_init()**, which is the initialization function for GTK+; this function will set up GTK+, the type system, the connection to the windowing environment, etc. The **gtk_init()** takes as arguments, the pointers to the command line arguments counter and string array; this allows GTK+ to parse specific command line arguments that control the behavior of GTK+ itself. The parsed arguments will be removed from the array, leaving the unrecognized ones for the application to parse.

The call to **gtk_window_new()** will create a new **GtkWindow** and store it inside the window variable. The type of the window is **GTK_WINDOW_TOPLEVEL**, which means that the **GtkWindow** will be managed by the windowing system: it will have a frame, a title bar and window controls, depending on the platform, in this case, the Ubuntu 12.04 LTS.

GtkWidgets is hidden by default. By calling **gtk_widget_show()**, **GtkWidget** is asking GTK+ to set the visibility attribute to display. This continues within the main loop of the program and goes on until the close icon on the frame title bar is clicked, which terminates the program. [18]

3.2.2 Cross compiling GTK

To compile a GTK+ application, the compiler is told where to find the GTK+ header files and libraries. This is done with the pkg-config utility using the GCC compiler.

```
gcc ‘pkg-config –cflags gtk+-3.0’ -o sample-gtk sample-gtk.c ‘pkg-config –libs gtk+-3.0’
```

This outputs an executable (application/x-executable) file “**sample-gtk**” in the home directory.

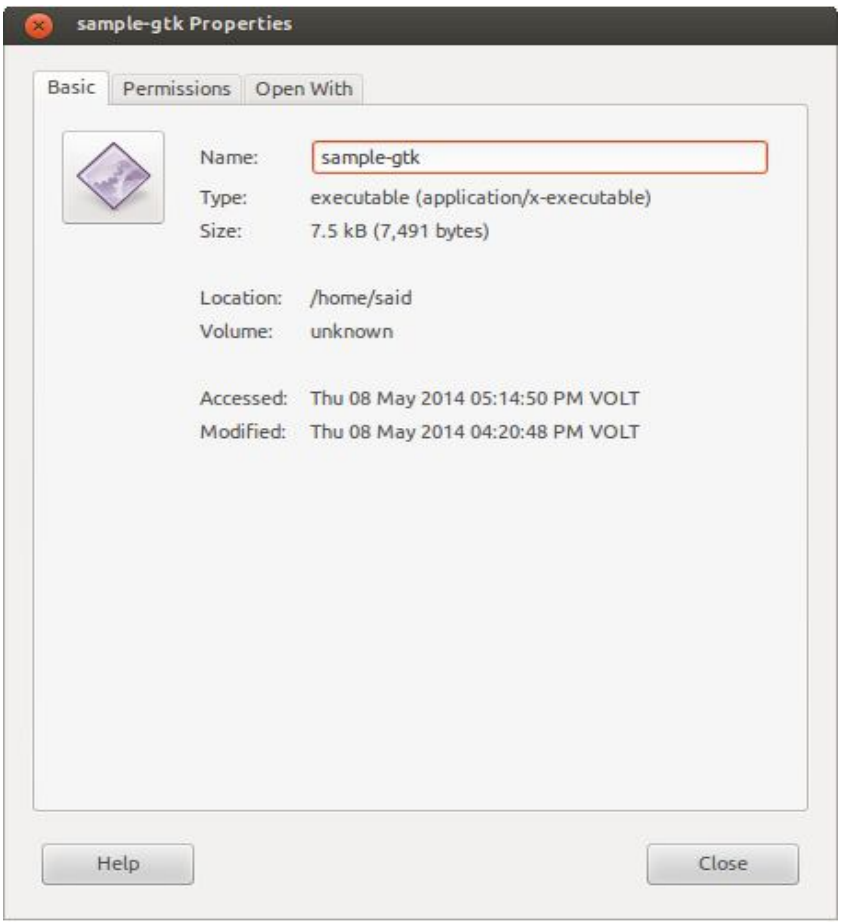

Figure 3.1: Sample GTK file properties

Running the file will open an empty 200x200 pixel window as shown below

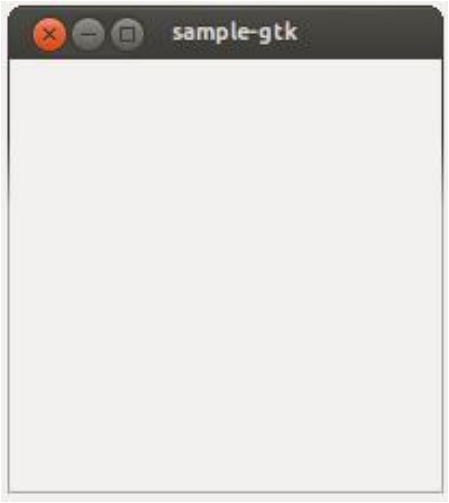

Figure 3.2: Sample-gtk application

3.3 ALSA

When writing an ALSA audio application, the following flowchart shows the typical structure followed:

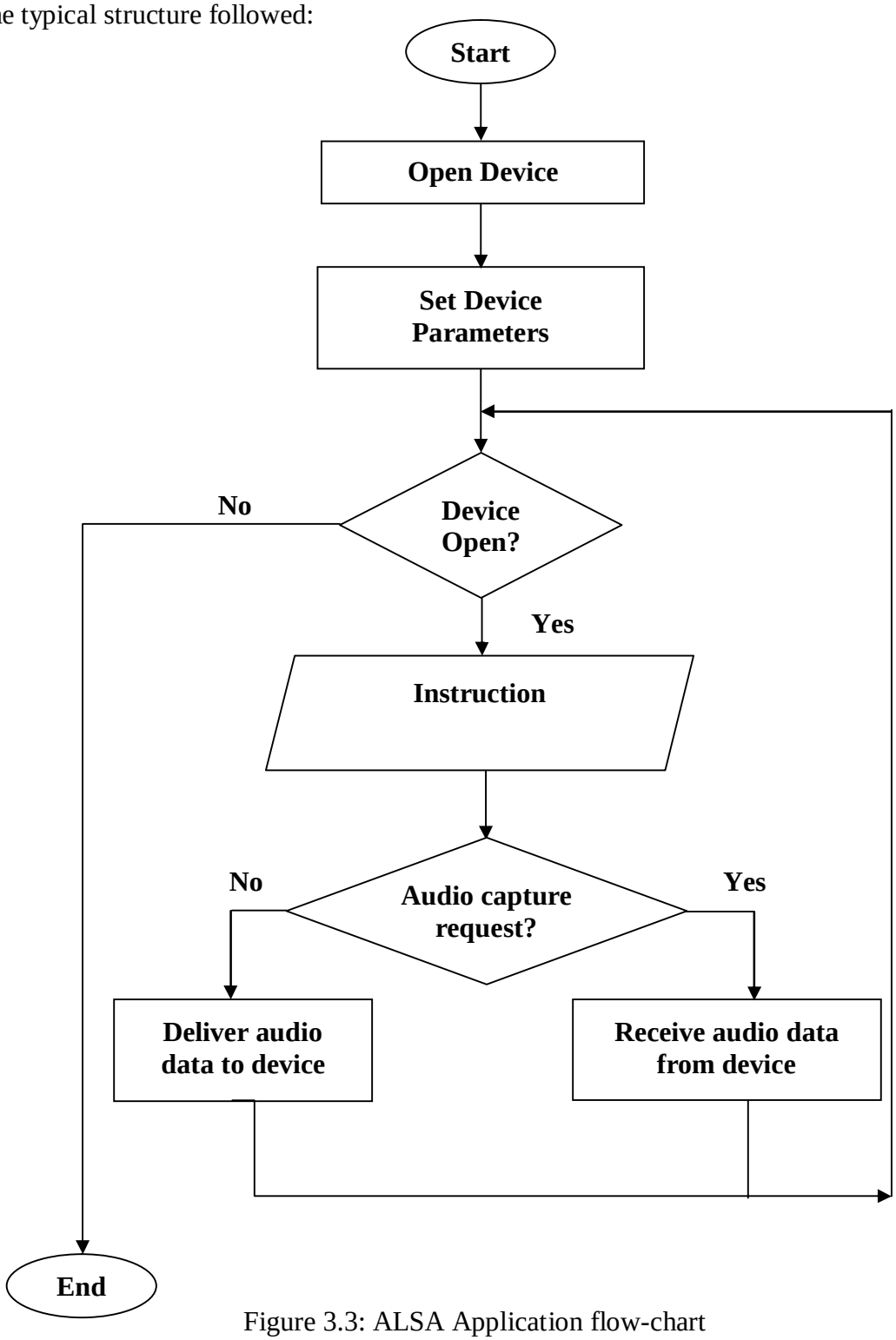

Figure 3.3: ALSA Application flow-chart

3.3.1 Listing ALSA sound cards

The first thing to do in developing an ALSA application is to list all the available sound cards/devices on the machine. ALSA has some functions to list all available sound cards/devices. One of such functions is snd_card_next().

Therefore, a loop is created with the pointer at an integer value of -1. The function will then change the value of int to the number of the first card/device in the system. This continue to increment until there is no more sound cards to show, then ALSA sets int back to -1.

```
“#include <stdio.h>
#include <string.h>
#include <alsa/asoundlib.h>
Int main (int argc, char **argv)
{
   register int    err;
   int             cardNum, totalCards;
   totalCards =    0;
   cardNum = -1;
   for (;;)
   {
    if ((err = snd_card_next(&cardNum)) < 0)
    {
    printf(“Next   card   not   available:   %s\n”,
    snd_strerror(err));
```

```
        break;
        }
        If (cardNum < 0) break; ++totalCards;
        }
        printf("ALSA found %i cards\n", totalcards);
        snd_config_update_free_global();
}
```
[8]

After including all necessary libraries, that's the **stdio**, **string** and **asoundlib** headers, variables were then defined. **cardNum** is the card number at the instance initially set at -1 and **totalCards** is the total number of cards found set to 0.

An "if" block is used for the **snd_card_next** function. When card number is -1, ALSA will fetch the first card. This continues untill there are no more cards, then it sums up all cards found and print out **totalCards** found.

ALSA allocates memory space to load its configuration file when a handle is called. To free the memory and unload the information, the funtion **snd_config_update_free_global** is called with null value.

3.3.2 Setting hardware parameters

The following parameters can be set for audio application based on card type or audio chip capabilities.

Sample rate (8Khz, 22Khz or 44.1Khz)

Playback bit resolution (8 bit, 16 bit, 32 bit)

Channels (mono, stereo or multi-channel)

ALSA has a function that enables the direct settings of hardware parameters, that is the **snd_pcm_set_params**()

With the device parameters set, a Linux audio driver can be developed using ALSA for various applications which will be detailed in the next chapter. A comprehensive GUI designed with GTK will also accompany the driver for easy access.

CHAPTER 4: DRIVER AND USER INTERFACE DEVELOPMENT

4.1 DRIVER DEVELOPMENT LIBRARIES

The driver development required the following packages that provide libraries and utilities for the program:

libasound2: The shared library package for ALSA that contains the ALSA library and its standard plug-ins. It is provided by liboss-alsa-asound2 (OSS to ALSA compatibility library)

libasound2-dev: The shared library package for ALSA that contains files required for developing software that makes use of libasound2, the ALSA library. It is provided by liboss4-salsa-dev (OSS to ALSA compatibility library development files)

With the **asoundlib.h** header file included in the program, all required ALSA files will be included and cross linked accordingly.

4.2 INITIALIZATION

To initialize the ALSA program, three steps are involved

i. Opening the sound device

Creating structure to hold the parameters to be used

ii. Instructing sound device to use the settings

4.2.1 Listing sound devices

As discussed in previous chapter, the sound devices on the machine can be listed with

```
“#include <stdio.h>
#include <string.h>
#include <alsa/asoundlib.h>
Int main (int argc, char **argv)
{
     register int  err;
     int           cardNum, totalCards;
     totalCards =  0;
     cardNum = -1;
     for (;;)
     {
     if ((err = snd_card_next(&cardNum)) < 0)
    {
    printf(“Next   card   not   available:   %s\n”,
    snd_strerror(err));
    break;
     }
     If (cardNum < 0) break;
     ++totalCards;
     }
```

```
        printf("ALSA found %i cards\n", totalcards);
        snd_config_update_free_global();
    }"
```
[8]

A more direct way to get information on the capture and playback devices on the host computer on which the driver will be written and tested is through the terminal as below described below;

```
said@said-HP-655-Notebook-PC:~$ aplay –l
**** List of PLAYBACK Hardware Devices ****
card 0: Generic [HD-Audio Generic 3: HDMI 0 [HDMI 0]
   Subdevices: 1/1
   Subdevice #0: Subdevice #0
Card 1: Generic_1 [HD-Audio Generic], device 0: ALC269VC
Analog [ALC269VC Analog]
   Subdevices: 1/1
   Subdevice #0: Subdevice #0
```

The device is **plughw:0,0**. **0,0** are for the **Device 0** and **Subdevice #0**

```
said@said-HP-655-Notebook-PC:~$ arecord –l
**** List of CAPTURE Hardware Devices ****
Card 1: Generic_1 [HD-Audio Generic], device 0: ALC269VC
Analog [ALC269VC Analog]
   Subdevices: 1/1
```

```
   Subdevice #0: Subdevice #0
```

This device is also **plughw:0,0**. **0,0**, the default.

4.2.2 Opening device and setting parameters

The following opens the default PCM device for playback and set some parameters using newest ALSA API

```
“#define ALSA_PCM_NEW_HW_PARAMS_API
#include <alsa/asoundlib.h>
int main() {
   int rc;
   snd_pcm_t *handle;
   snd_pcm_hw_params_t *params;
   unsigned int val;
   int dir;
   snd_pcm_uframes_t frames;
// to open PCM device for playback
   rc =
snd_pcm_open(&handle,"default",SND_PCM_STREAM_PLAYBACK,
0);

   if (rc < 0) {
   fprintf(stderr,"unable to open pcm device:%s\n",
snd_strerror(rc));
```

```
   exit(1);
   }
// Allocating default hardware parameters object
   snd_pcm_hw_params_alloca(&params);
   snd_pcm_hw_params_any(handle, params);
//changing to desired hardware parameters
   snd_pcm_hw_params_set_access(handle, params,
SND_PCM_ACCESS_RW_INTERLEAVED);
   snd_pcm_hw_params_set_format(handle, params,
SND_PCM_FORMAT_S16_LE);
   snd_pcm_hw_params_set_channels(handle, params, 2);
   val = 44100;
   snd_pcm_hw_params_set_rate_near(handle,params, &val,
&dir);
   frames = 32;
   snd_pcm_hw_params_set_period_size_near(handle,
params, &frames, &dir);
// Writing the parameters to the driver
   rc = snd_pcm_hw_params(handle, params);
   if (rc < 0) {
        fprintf(stderr,"unable to set hw parameters:
%s\n",snd_strerror(rc));
   exit(1);
   }
```

```
return 0;
}”
```
[8]

This will open a default playback device, initially with default hardware parameters, the change them to [interleaved – stereo (2-channel) – signed 16 bit little endian – 44100 bit rate – 32 frame size]. This completes the device initialization. [22]

4.3 AUDIO PLAYBACK

Audio playback is achieved by reading stream audio data from memory and writing it to the PCM device. For this section, a 5sec data is read from input and written to the default PCM device. This follows after opening the device and setting its parameters as shown above.

```
“#define ALSA_PCM_NEW_HW_PARAMS_API
#include <alsa/asoundlib.h>
int main() {
   long loops;
   int rc;
   int size;
   snd_pcm_t *handle;
   snd_pcm_hw_params_t *params;
   unsigned int val;
   int dir;
   snd_pcm_uframes_t frames;
```

```
   char *buffer;
// *****************
   snd_pcm_hw_params_get_period_size(params, &frames,
&dir);
   size = frames * 4; /* 2 bytes/sample, 2 channels */
   buffer = (char *) malloc(size);
   snd_pcm_hw_params_get_period_time(params,&val,
&dir);
// 5 seconds in microseconds/period time
   loops = 5000000 / val;
   while (loops > 0) {
   loops--;
   rc = read(0, buffer, size);
   if (rc == 0) {
   fprintf(stderr, "end of file on input\n");break;
        }
   else if (rc != size) {
   fprintf(stderr, "short read: read %d bytes\n", rc);
        }
   rc = snd_pcm_writei(handle, buffer, frames);
        if (rc == -EPIPE) {
   fprintf(stderr, "underrun occurred\n");
   snd_pcm_prepare(handle);
   }
```

```
    else if (rc < 0) {
    fprintf(stderr,"error from
writei:%s\n",snd_strerror(rc));
         }
    else if (rc != (int)frames) {
    fprintf(stderr,"short write, write %d rames\n", rc);
         }
    }
    snd_pcm_drain(handle);
    snd_pcm_close(handle);
    free(buffer);
    return 0;
    }”
```
[8]

The error code EPIPE means **xrun** (**underrun** for playback or **overrun** for capture). **Underrun** happens when an application does not feed new samples in time to **alsa-lib** while **overrun** happens when an application does not take new captured samples in time from **alsa-lib**. [21]

After initialization, sound samples are written to the sound card to produce playback. Bytes are read from standard input for one period, then written to sound card continuously for 5sec.

Default period size is used and made size of the buffer for storage. The period is found so that the number of periods the program has to process in order to run for 5sec is determined.

While constantly looking for errors, a loop is created to manage the data to read from standard input and fill the buffer with a sample period.

snd_pcm_writei call is used to send data to the PCM device. The return code is checked for a number of error conditions like EPIPE, etc.

To recover from the EPIPE error, **snd_pcm_prepare** function call is used to put the stream in the PREPARED state so it can start again the next time data is written to the stream. Any other error received, the program will display the error code and continue.

The program is looped until 5sec length of data frames is transferred or end of file is reached. **snd_pcm_drain** is then called to transfer pending sound samples before closing the stream. The dynamically allocated buffer is then freed. [22]

4.4 AUDIO CAPTURE

The same initialization procedure applies for audio capture or sound recording, but instead of opening the device for playback, it is opened for capture with the syntax

```
rc = snd_pcm_open(&handle,"default",
SND_PCM_STREAM_CAPTURE, 0)
```

The audio capture program is also similar to the playback, the program is as below

```
#define ALSA_PCM_NEW_HW_PARAMS_API
```

```
#include <alsa/asoundlib.h>
int main() {
   long loops;
   int rc;
   int size;
   snd_pcm_t *handle;
   snd_pcm_hw_params_t *params;
   unsigned int val;
   int dir;
   snd_pcm_uframes_t frames;
   char *buffer;
// *****************
   snd_pcm_hw_params_get_period_size(params, &frames,
&dir);
   size = frames * 4; /* 2 bytes/sample, 2 channels */
   buffer = (char *) malloc(size);
   snd_pcm_hw_params_get_period_time(params,&val,
&dir);
   loops = 5000000 / val;
   while (loops > 0) {
   loops--;
   rc = snd_pcm_readi(handle, buffer, frames);
   if (rc == -EPIPE) {
   fprintf(stderr, "overrun occurred\n");
```

```
    snd_pcm_prepare(handle);
     } else if (rc < 0) {
    fprintf(stderr,"error from read:
%s\n",snd_strerror(rc));
     } else if (rc != (int)frames) {
    fprintf(stderr, "short read, read %d frames\n",rc);
     }
    rc = write(1, buffer, size);
    if (rc != size)
    fprintf(stderr,"short write: wrote %d bytes\n", rc);
  }
  snd_pcm_drain(handle);
  snd_pcm_close(handle);
  free(buffer);
  return 0;
}”
```
[8]

When the PCM stream is opened, the capture mode is specified as SND_PCM_STREAM_CAPTURE. Within the main processing loop, samples are read from the sound hardware using snd_pcm_readi and written to standard output using write. Errors and overrun are checked like in the play program.

For both audio capture and playback, the data stream has to be directed to a file. The mixer tool is used to set the level of the recording source. For ALSA,

there is an in package ALSAmixer, a textmode based mixer program for ALSA soundcard drivers. [22]

4.5 GTK INTERFACE

For the interfacing of the ALSA driver application on a graphical user environment, a GTK program is developed. It is made as easy as possible to avoid complication just to record and playback sound. Other driver properties relating to the sound card are accessed with the ALSAmixer program.

```
#include <gtk/gtk.h>
int
main (int   argc, char *argv[])
{
GtkWidget *window;
GtkWidget *grid;
GtkWidget *button;
gtk_init (&argc, &argv);
window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (window), "Grid");
g_signal_connect (window, "destroy", G_CALLBACK
(gtk_main_quit), NULL);
gtk_container_set_border_width (GTK_CONTAINER (window),
10);
grid = gtk_grid_new ();
```

```
gtk_container_add (GTK_CONTAINER (window), grid);
button = gtk_button_new_with_label ("CAPTURE");
g_signal_connect (button, "clicked", G_CALLBACK
(print_hello), NULL);
gtk_grid_attach (GTK_GRID (grid), button, 0, 0, 1, 1);
button = gtk_button_new_with_label ("PLAYBACK");
g_signal_connect (button, "clicked", G_CALLBACK
(print_hello), NULL);
gtk_grid_attach (GTK_GRID (grid), button, 1, 0, 1, 1);
button = gtk_button_new_with_label ("Quit");
g_signal_connect (button, "clicked", G_CALLBACK
(gtk_main_quit), NULL);
gtk_grid_attach (GTK_GRID (grid), button, 0, 1, 2, 1);
gtk_widget_show_all (window);
gtk_main ();
return 0;
}
```

After including the GTK library and defining variables, a window is created. A grid container is constructed to pack the buttons to be used. The buttons are placed in such a way that first button (CAPTURE) is in grid cell (0,0) and fill just 1 cell horizontally and vertically. The second button (PLAYBACK) is placed in (1,0). The last button (Quit) is placed in (0,1) and spanned 2 columns so it can appear at the center of the window. All the widgets are shown at once by calling gtk_widget_show_all().

Running this program will create a window similar to following

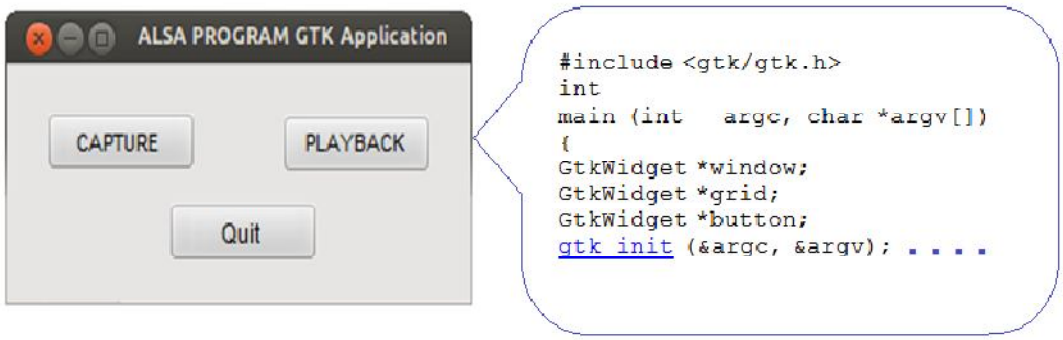

Figure 4.1 Audio capture and playback GTK interface

This application is linked with the ALSA handle to capture and playback audio according to default parameters. A pop-up will give directory option for both storing and retrieving the sound files.

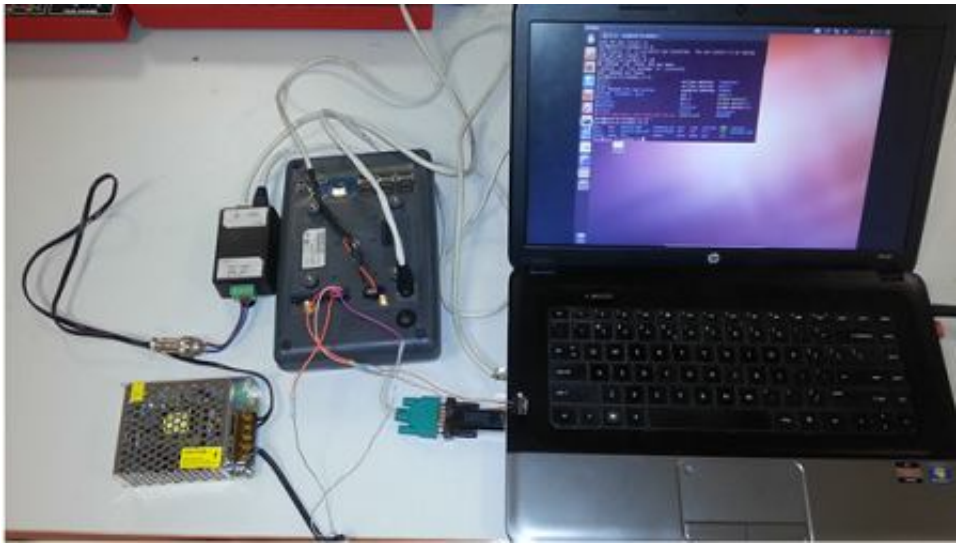

Figure 4.2: Experimental Setup

CHAPTER 5: GENERAL CONCLUSION

The mechanism involved in Linux desktop sound architecture had been analyzed in this research work. The evolution of audio in Linux systems from the OSS (Open Sound System) to the current ALSA (Advance Linux Sound Architecture) had all been looked into. The properties and limitations of the OSS were analyzed and thus, the need for the transition to ALSA.

Embedded systems that are on the rise today usually don't have conventional audio capability. Mostly sound in embedded system is restricted to beeps. However with the development of more capable systems, audio incorporation is necessary in such systems that have become vital part of today.

Majority of embedded systems are developed on Linux kernel. Therefore it is paramount that the research centered mainly on the development of basic audio application driver for capture and playback on such systems in similar fashion by which audio is developed for conventional desktop computers.

The project utilizes ALSA, the Advanced Linux Sound Architecture to develop an application on a Linux host machine running Ubuntu operating system. The program is developed using the Eclipse C Development Tool and crossed compiled with GCC so that it can run on ARM processor of the target machine.

The target machine is an i.MX53 powered board in a tablet PC. This ARM machine was chosen because it is the processor used in as many embedded systems. It has the RISC (Reduced Instruction Set Computer) architecture which has fewer instruction commands.

The driver developed has the capability to find all the audio devices (cards or chips as in most modern computing devices) for playback and capture. Once these devices are found, the program will go on to open these devices for either capture (sound recording) or playback (sound playing) and also set the parameters with which either of these will be done. The parameters set are the bit rate, little endian, and frame size.

Once these parameters are set according to the mode of operation, the driver will use these settings to record any audio from the capturing device, e.g. a microphone and store it in a memory directly accessible by the program. In the playback mode, the program will go to the memory location pointed and stream the data found through the playback device, mostly a speaker.

At the end of these audio devices are transducers that convert the digital data sent by the program into the analog audible audio by a DAC or an ADC in the reverse process.

For an easy user accessible interface, the GTK (GIMP Tool Kit) is used to design a graphical user interface that provides a simple window on which buttons are pressed to initialize the program and perform these basic sound operations. The GTK application is also developed on the Eclipse CDT and cross compiled for the ARM target processor.

RECOMMENDATIONS

This driver is developed as an application to run on an ARM based embedded machine that already has a Linux operating system running on it. Since most of the Linux distributions are free and open source, this application can be introduced into the operating system kernel so that it will run come pre installed in the OS without the necessary need to develop it separately as carried out in this research work.

The project also targeted only ARM based processors running Linux OS. In future works, other processors and operating systems platforms can be considered especially with Android which is rapidly developing into the embedded system technology.

APPENDIX A: SECREEN SHOTS

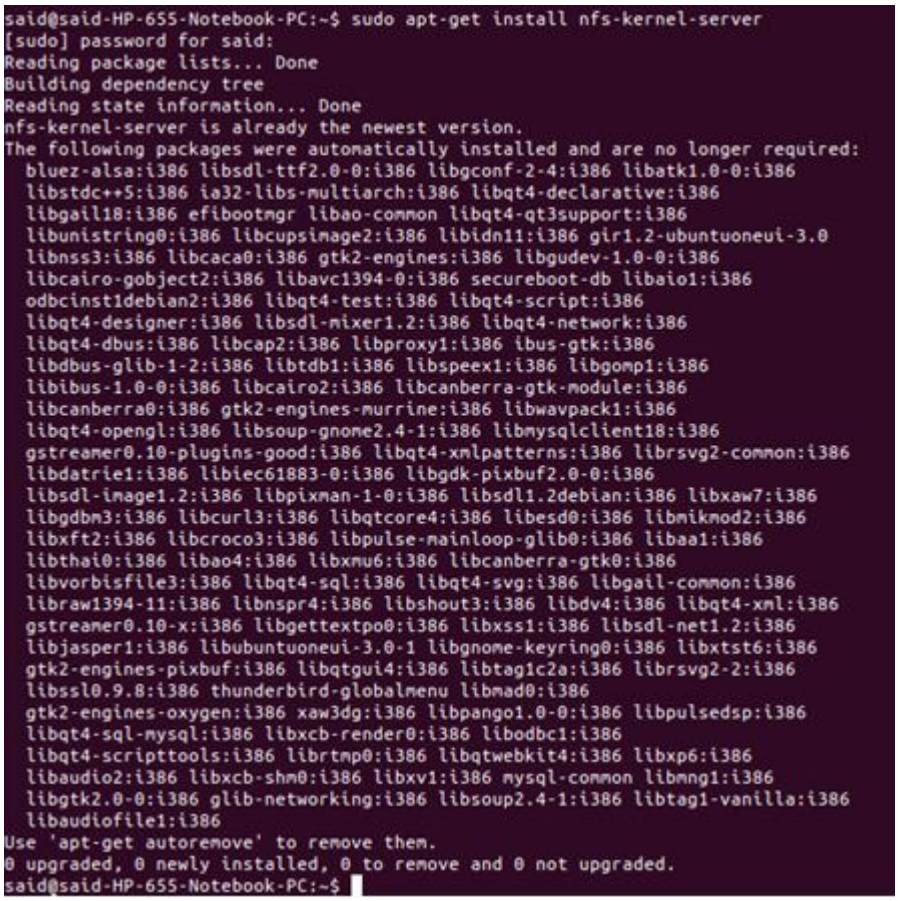

Figure A.1: NFS repository installation

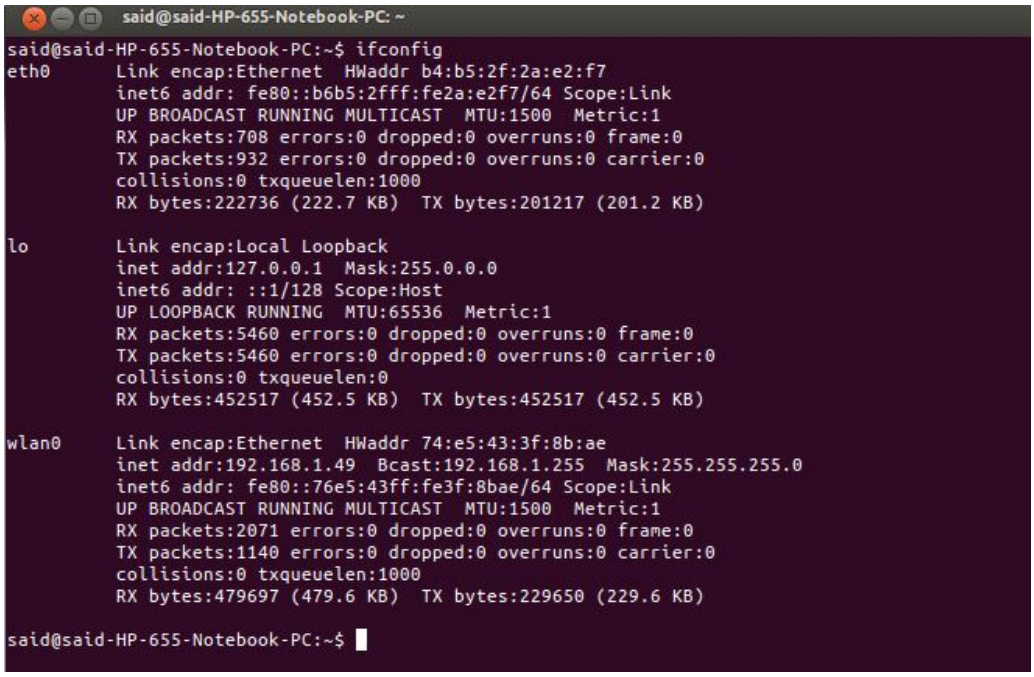

Figure A.2: ifconfig secreenshot

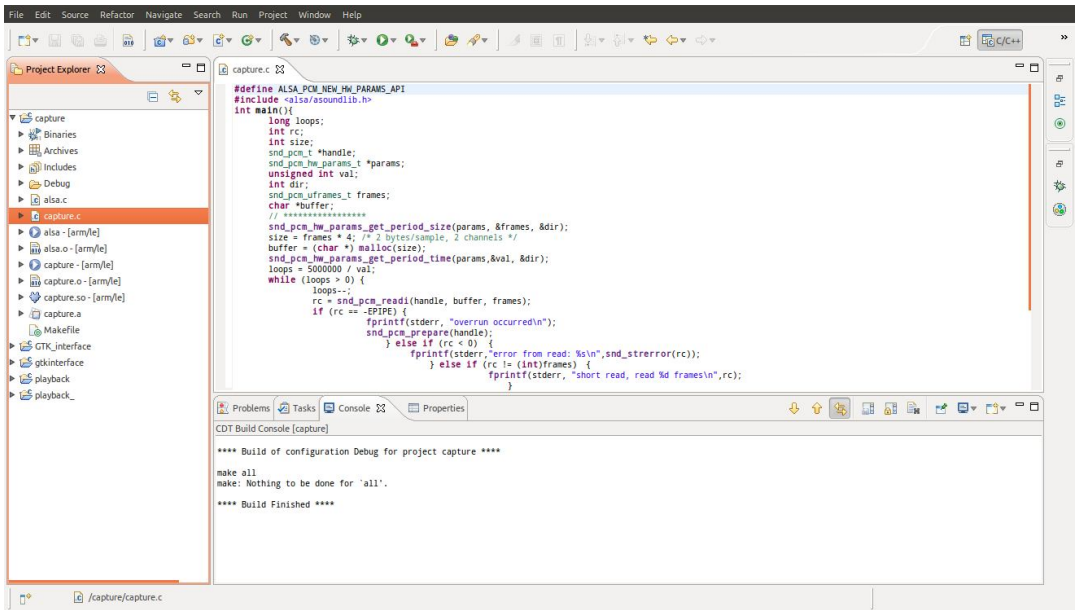

Figure A.3: Eclipse CDT Environment

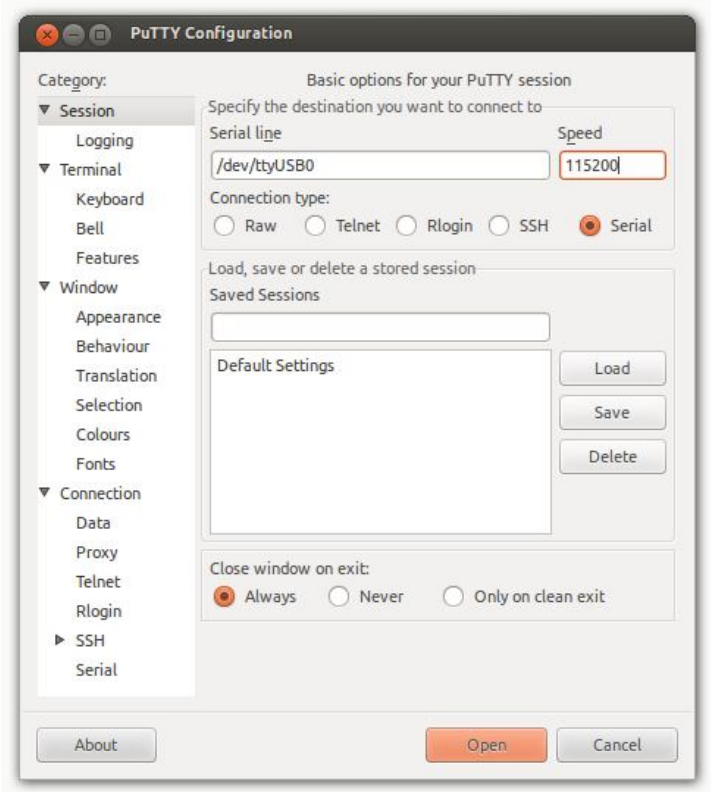

Figure A.4: PuTTY emulator

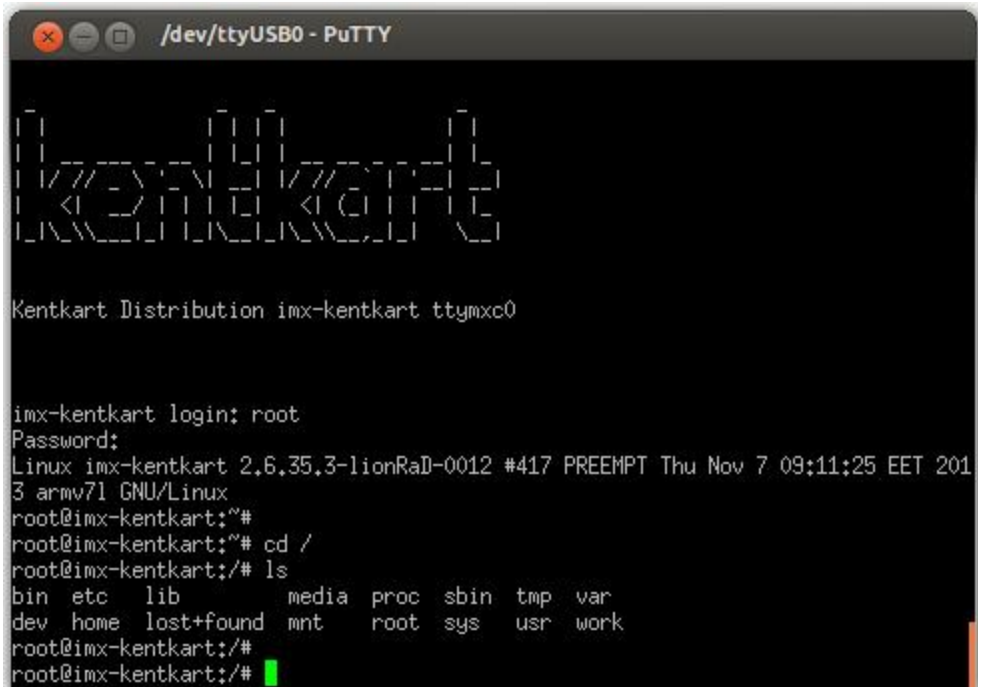

Figure A.5: PuTTY emulator in tablet console

APPENDIX B: HEADER FILES

asoundlib.h

/*file include/asoundlib.h. brief Application interface library for the ALSA driver.

Author Jaroslav Kysela perex@perex.cz, Abramo Bagnara abramo@alsa-project.org, Takashi Iwai tiwai@suse.de, 1998-2001 */

```
#ifndef __ASOUNDLIB_H

#define __ASOUNDLIB_H

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <assert.h>

#include <endian.h>

#include <sys/poll.h>

#include <errno.h>

#include <stdarg.h>

#include <alsa/asoundef.h>
```

```
#include <alsa/version.h>

#include <alsa/global.h>

#include <alsa/input.h>

#include <alsa/output.h>

#include <alsa/error.h>

#include <alsa/conf.h>

#include <alsa/pcm.h>

#include <alsa/rawmidi.h>

#include <alsa/timer.h>

#include <alsa/hwdep.h>

#include <alsa/control.h>

#include <alsa/mixer.h>

#include <alsa/seq_event.h>

#include <alsa/seq.h>

#include <alsa/seqmid.h>

#include <alsa/seq_midi_event.h>

#endif /* __ASOUNDLIB_H */
```

gdk.h

```
/* GDK - The GIMP Drawing KitCopyright (C) 1995-1997 Peter Mattis, Spencer Kimball and Josh MacDonald  */

#ifndef __GDK_H__

#define __GDK_H__

#define __GDK_H_INSIDE__

#include <gdk/gdkconfig.h>

#include <gdk/gdkversionmacros.h>

#include <gdk/gdkapplaunchcontext.h>

#include <gdk/gdkcairo.h>

#include <gdk/gdkcolor.h>

#include <gdk/gdkcursor.h>

#include <gdk/gdkdevice.h>

#include <gdk/gdkdevicemanager.h>

#include <gdk/gdkdisplay.h>

#include <gdk/gdkdisplaymanager.h>

#include <gdk/gdkdnd.h>

#include <gdk/gdkenumtypes.h>
```

```
#include <gdk/gdkevents.h>

#include <gdk/gdkkeys.h>

#include <gdk/gdkkeysyms.h>

#include <gdk/gdkmain.h>

#include <gdk/gdkpango.h>

#include <gdk/gdkpixbuf.h>

#include <gdk/gdkproperty.h>

#include <gdk/gdkrectangle.h>

#include <gdk/gdkrgba.h>

#include <gdk/gdkscreen.h>

#include <gdk/gdkselection.h>

#include <gdk/gdktestutils.h>

#include <gdk/gdkthreads.h>

#include <gdk/gdktypes.h>

#include <gdk/gdkvisual.h>

#include <gdk/gdkwindow.h>

#undef __GDK_H_INSIDE__
```

#endif /* __GDK_H__ */

gtk.h

/* GTK - The GIMP Toolkit. Copyright (C) 1995-1997 Peter Mattis, Spencer Kimball and Josh MacDonald */

#ifndef __GTK_H__

#define __GTK_H__

#define __GTK_H_INSIDE__

#include <gdk/gdk.h>

#include <gtk/gtkaboutdialog.h>

#include <gtk/gtkaccelgroup.h>

#include <gtk/gtkaccellabel.h>

#include <gtk/gtkaccelmap.h>

#include <gtk/gtkaccessible.h>

#include <gtk/gtkaction.h>

#include <gtk/gtkactionable.h>

#include <gtk/gtkactiongroup.h>

#include <gtk/gtkactivatable.h>

#include <gtk/gtkadjustment.h>

```
#include <gtk/gtkalignment.h>

#include <gtk/gtkappchooser.h>

#include <gtk/gtkappchooserdialog.h>

#include <gtk/gtkappchooserwidget.h>

#include <gtk/gtkappchooserbutton.h>

#include <gtk/gtkapplication.h>

#include <gtk/gtkapplicationwindow.h>

#include <gtk/gtkarrow.h>

#include <gtk/gtkaspectframe.h>

#include <gtk/gtkassistant.h>

#include <gtk/gtkbbox.h>

#include <gtk/gtkbin.h>

#include <gtk/gtkbindings.h>

#include <gtk/gtkborder.h>

#include <gtk/gtkbox.h>

#include <gtk/gtkbuildable.h>

#include <gtk/gtkbuilder.h>
```

```
#include <gtk/gtkbutton.h>

#include <gtk/gtkcalendar.h>

#include <gtk/gtkcellarea.h>

#include <gtk/gtkcellareabox.h>

#include <gtk/gtkcellareacontext.h>

#include <gtk/gtkcelleditable.h>

#include <gtk/gtkcelllayout.h>

#include <gtk/gtkcellrenderer.h>

#include <gtk/gtkcellrendereraccel.h>

#include <gtk/gtkcellrenderercombo.h>

#include <gtk/gtkcellrendererpixbuf.h>

#include <gtk/gtkcellrendererprogress.h>

#include <gtk/gtkcellrendererspin.h>

#include <gtk/gtkcellrendererspinner.h>

#include <gtk/gtkcellrenderertext.h>

#include <gtk/gtkcellrenderertoggle.h>

#include <gtk/gtkcellview.h>
```

```
#include <gtk/gtkcheckbutton.h>

#include <gtk/gtkcheckmenuitem.h>

#include <gtk/ubuntumenuproxy.h>

#include <gtk/ubuntumenuproxymodule.h>

#include <gtk/gtkclipboard.h>

#include <gtk/gtkcolorbutton.h>

#include <gtk/gtkcolorchooser.h>

#include <gtk/gtkcolorchooserdialog.h>

#include <gtk/gtkcolorchooserwidget.h>

#include <gtk/gtkcolorutils.h>

#include <gtk/gtkcombobox.h>

#include <gtk/gtkcomboboxtext.h>

#include <gtk/gtkcontainer.h>

#include <gtk/gtkcssprovider.h>

#include <gtk/gtkcsssection.h>

#include <gtk/gtkdebug.h>

#include <gtk/gtkdialog.h>
```

```
#include <gtk/gtkdnd.h>

#include <gtk/gtkdrawingarea.h>

#include <gtk/gtkeditable.h>

#include <gtk/gtkentry.h>

#include <gtk/gtkentrybuffer.h>

#include <gtk/gtkentrycompletion.h>

#include <gtk/gtkenums.h>

#include <gtk/gtkeventbox.h>

#include <gtk/gtkexpander.h>

#include <gtk/gtkfixed.h>

#include <gtk/gtkfilechooser.h>

#include <gtk/gtkfilechooserbutton.h>

#include <gtk/gtkfilechooserdialog.h>

#include <gtk/gtkfilechooserwidget.h>

#include <gtk/gtkfilefilter.h>

#include <gtk/gtkfontbutton.h>

#include <gtk/gtkfontchooser.h>
```

```
#include <gtk/gtkfontchooserdialog.h>

#include <gtk/gtkfontchooserwidget.h>

#include <gtk/gtkframe.h>

#include <gtk/gtkgradient.h>

#include <gtk/gtkgrid.h>

#include <gtk/gtkiconfactory.h>

#include <gtk/gtkicontheme.h>

#include <gtk/gtkiconview.h>

#include <gtk/gtkimage.h>

#include <gtk/gtkimagemenuitem.h>

#include <gtk/gtkimcontext.h>

#include <gtk/gtkimcontextinfo.h>

#include <gtk/gtkimcontextsimple.h>

#include <gtk/gtkimmulticontext.h>

#include <gtk/gtkinfobar.h>

#include <gtk/gtkinvisible.h>

#include <gtk/gtklabel.h>
```

```
#include <gtk/gtklayout.h>

#include <gtk/gtklinkbutton.h>

#include <gtk/gtkliststore.h>

#include <gtk/gtklockbutton.h>

#include <gtk/gtkmain.h>

#include <gtk/gtkmenu.h>

#include <gtk/gtkmenubar.h>

#include <gtk/gtkmenuitem.h>

#include <gtk/gtkmenushell.h>

#include <gtk/gtkmenutoolbutton.h>

#include <gtk/gtkmessagedialog.h>

#include <gtk/gtkmisc.h>

#include <gtk/gtkmodules.h>

#include <gtk/gtkmountoperation.h>

#include <gtk/gtknotebook.h>

#include <gtk/gtknumerableicon.h>

#include <gtk/gtkoffscreenwindow.h>
```

```
#include <gtk/gtkorientable.h>

#include <gtk/gtkoverlay.h>

#include <gtk/gtkpagesetup.h>

#include <gtk/gtkpapersize.h>

#include <gtk/gtkpaned.h>

#include <gtk/gtkprintcontext.h>

#include <gtk/gtkprintoperation.h>

#include <gtk/gtkprintoperationpreview.h>

#include <gtk/gtkprintsettings.h>

#include <gtk/gtkprogressbar.h>

#include <gtk/gtkradioaction.h>

#include <gtk/gtkradiobutton.h>

#include <gtk/gtkradiomenuitem.h>

#include <gtk/gtkradiotoolbutton.h>

#include <gtk/gtkrange.h>

#include <gtk/gtkrecentaction.h>

#include <gtk/gtkrecentchooser.h>
```

```
#include <gtk/gtkrecentchooserdialog.h>

#include <gtk/gtkrecentchoosermenu.h>

#include <gtk/gtkrecentchooserwidget.h>

#include <gtk/gtkrecentfilter.h>

#include <gtk/gtkrecentmanager.h>

#include <gtk/gtkscale.h>

#include <gtk/gtkscalebutton.h>

#include <gtk/gtkscrollable.h>

#include <gtk/gtkscrollbar.h>

#include <gtk/gtkscrolledwindow.h>

#include <gtk/gtkselection.h>

#include <gtk/gtkseparator.h>

#include <gtk/gtkseparatormenuitem.h>

#include <gtk/gtkseparatortoolitem.h>

#include <gtk/gtksettings.h>

#include <gtk/gtkshow.h>

#include <gtk/gtksizegroup.h>
```

```
#include <gtk/gtksizerequest.h>

#include <gtk/gtkspinbutton.h>

#include <gtk/gtkspinner.h>

#include <gtk/gtkstatusbar.h>

#include <gtk/gtkstatusicon.h>

#include <gtk/gtkstock.h>

#include <gtk/gtkstylecontext.h>

#include <gtk/gtkstyleproperties.h>

#include <gtk/gtkstyleprovider.h>

#include <gtk/gtkswitch.h>

#include <gtk/gtksymboliccolor.h>

#include <gtk/gtktextattributes.h>

#include <gtk/gtktextbuffer.h>

#include <gtk/gtktextbufferrichtext.h>

#include <gtk/gtktextchild.h>

#include <gtk/gtktextiter.h>

#include <gtk/gtktextmark.h>
```

```
#include <gtk/gtktexttag.h>

#include <gtk/gtktexttagtable.h>

#include <gtk/gtktextview.h>

#include <gtk/gtkthemingengine.h>

#include <gtk/gtktoggleaction.h>

#include <gtk/gtktogglebutton.h>

#include <gtk/gtktoggletoolbutton.h>

#include <gtk/gtktoolbar.h>

#include <gtk/gtktoolbutton.h>

#include <gtk/gtktoolitem.h>

#include <gtk/gtktoolitemgroup.h>

#include <gtk/gtktoolpalette.h>

#include <gtk/gtktoolshell.h>

#include <gtk/gtktooltip.h>

#include <gtk/gtktestutils.h>

#include <gtk/gtktreednd.h>

#include <gtk/gtktreemodel.h>
```

```
#include <gtk/gtktreemodelfilter.h>

#include <gtk/gtktreemodelsort.h>

#include <gtk/gtktreeselection.h>

#include <gtk/gtktreesortable.h>

#include <gtk/gtktreestore.h>

#include <gtk/gtktreeview.h>

#include <gtk/gtktreeviewcolumn.h>

#include <gtk/gtktypebuiltins.h>

#include <gtk/gtktypes.h>

#include <gtk/gtkuimanager.h>

#include <gtk/gtkversion.h>

#include <gtk/gtkviewport.h>

#include <gtk/gtkvolumebutton.h>

#include <gtk/gtkwidget.h>

#include <gtk/gtkwidgetpath.h>

#include <gtk/gtkwindow.h>
```

```
#include <gtk/deprecated/gtkcolorsel.h>

#include <gtk/deprecated/gtkcolorseldialog.h>

#include <gtk/deprecated/gtkfontsel.h>

#include <gtk/deprecated/gtkhandlebox.h>

#include <gtk/deprecated/gtkhbbox.h>

#include <gtk/deprecated/gtkhbox.h>

#include <gtk/deprecated/gtkhpaned.h>

#include <gtk/deprecated/gtkhsv.h>

#include <gtk/deprecated/gtkhscale.h>

#include <gtk/deprecated/gtkhscrollbar.h>

#include <gtk/deprecated/gtkhseparator.h>

#include <gtk/deprecated/gtkrc.h>

#include <gtk/deprecated/gtkstyle.h>

#include <gtk/deprecated/gtktable.h>

#include <gtk/deprecated/gtktearoffmenuitem.h>

#include <gtk/deprecated/gtkvbbox.h>

#include <gtk/deprecated/gtkvbox.h>
```

```
#include <gtk/deprecated/gtkvpaned.h>

#include <gtk/deprecated/gtkvscale.h>

#include <gtk/deprecated/gtkvscrollbar.h>

#include <gtk/deprecated/gtkvseparator.h>

#undef __GTK_H_INSIDE__

#endif /* __GTK_H__ */
```

REFERENCES

[1] John Purcell, 1997, *Linux Complete Command Reference* Red Hat Software,Inc. USA

[2] Takashi Iwai, 2003, *Sound Systems on Linux: From the Past to the Future* Linux Conference, Edinburgh, Scotland

[3] Karim Yaghmour, Jon Masters, Gilad Ben-Yossed, Philippe Gerum 2008 *Building Embedded Linux Systems* O'Reilly Media, Inc., Sebastopol, California

[4] Remi Lorriaux, 2011, *Real-time Audio on Embedded Linux* Adeneo Embedded. Embedded Linux Conference

[5] Brian Fraser (2011) *Linux Audio Guide.* Last update: November 15, 2011

[6] Christopher Hallinan, 2010. *Embedded Linux Primer, Second Edition. A* Practical, Real-World Approach Prentice Hall, Boston, USA Pg10-11

[7] Kenneth H. Rosen, Douglas A. Host, James M. Farber, Richard R. Rosinski, 1999, *UNIX, The Complete Reference* Osbourne/McGraw-Hill Califonia USA

[8] Jeff Tranter, Sep 30, 2004 *Introduction to Sound Programming with ALSA*

http://www.linuxjournal.com/node/6735/print

[9] Takashi Iwai Sep, 2000. *ALSA Sequencer System,* SuSE GmbH, Nuremberg, Germany

[10] Takashi Iwai 2002-2005 *Writing an ALSA Driver*. http://ftp.rz.tu-bs.de/pub/mirror/ftp.kernel.org/people/tiwai/docs/writing-an-alsa-driver.pdf

[11] Matthias Nagorni 24 Feb 2010 *ALSA Programming HOWTO* http://users.suse.com/~mana/alsa090_howto.html

[12] Paul Davis 2002 *A Tutorial on Using the ALSA Audio API* http://equalarea.com/paul/alsa-audio.html

[13] Tavis B. Nicolai L. Seth V. Tom M., 2002 Linux NFS−HOWTO. The Linux Documentation Project http://www.tldp.org/HOWTO/pdf/NFS-HOWTO.pdf

[14] http://www.putty.org/

[15] http://www.gtk.org/tutorial1.2/gtk_tut-1.html

[16] http://ftp.acc.umu.se/pub/gnome/sources/

[17] http://packages.ubuntu.com/

[18] https://developer.gnome.org/gtk3/3.2/gtk-getting-started.html

[19] http://www.alsa-project.org/

[20] http://www.wikipedia.org

[21] http://www.alsa-project.org/alsa-doc/alsa-lib/pcm.html

[22] http://www.linuxjournal.com/article/7705