Design of Embedded Linux Based Voice Calling Device

DISSERTATION-II

Submitted in a partial fulfillment of the requirement for the award of the degree of engineering

MASTER OF TECHNOLOGY
In
Electronics and Communication Engineering

By

Mahendra Swain (11307913)

Under the guidance of

Mr. Abhishek Kumar Srivastava
Assistant Professor, ECE
LPU, JALANDHAR

Transforming Education Transforming India

Department of Electronics & Comm. Engineering

Lovely Professional University

Jalandhar–144402, Punjab (India)

ABSTRACT

Now a days, Linux is the choice for every OS compatible embedded platform. The design of "Embedded Linux based Voice Calling Device" is a prototype designed using Raspberry Pi (Model B), piTFT (2.8"), GSM Modem (SIM 900A) and python based programming for voice calling purpose. At the core of this prototype, Embedded Linux kernel image have been implemented, which is developed and customized using Yocto Project, used for reducing memory footprint (both at kernel and package level) and adding selective features as needed. Bitbake (command line) and hob (GUI based) are two available ways to customize LINUX packages in Yocto project to build optimum sized Image with preferred features.

The Linux has been chosen for its advantages like open source, platform independent, more secure, and development oriented. The Raspberry Pi board used is cost effective in comparison to other available Linux operated boards (with optimum specs) and a have better community support. Popularity of Raspberry Pi can be estimated that till date 5 million Raspberry Pi boards have been sold as per survey done in Jan 2015.

The software platform used to design this prototype is Ubuntu 14.04 and programming language used is Python 2.7. The use of Python programming language makes it more Real time and easy to understand.

This Embedded Linux based device gives the flexibility to have multitasking feature with minimum memory footprint and optimum speed on same platform which makes voice calling possible along with available games or/and music or/and web browsing features etc. Till date, we have successfully made voice transmission using custom LINUX over Raspberry Pi via GSM modem using terminal (without using Pi TFT display).

ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to **Mr. Abhishek Kumar Srivastava** Assistant Professor, Electronics and Communication Engineering Department, Lovely professional University, Jalandhar for his gracious flawless efforts and forth right suggestions blended with an innate intelligent application have crowned my task with success. I am truly very fortunate to have the opportunity to work with him. I found his guidance to be extremely valuable.

I am also thankful to entire faculty and staff members of Electronics and Communication Engineering Department and my friends who contributed directly or indirectly through there constructive criticism in evolution and preparation of this report work.

I extend my deepest gratitude to my parents and brother for their love, affection, encouragement and support.

My acknowledgements would not be complete without expressing my gratitude towards Almighty God. I feel very fortunate to come to know Him during all these months of my life and have continually been blessed by His endless love ever since. He is the true shepherd of my life.

DECLARATION

I hereby declare that the Dissertation-II report entitled "**DESIGN OF EMBEDDED LINUX BASED VOICE CALLING DEVICE**", is an authentic record of my own work carried out as per the requirements for the award of degree of Master of Technology in ECE, Embedded Systems at Lovely Professional University, Jalandhar under the guidance of **Mr. Abhishek Kumar Srivastava**, Assistant Professor, Department of Electronics and Communication Engineering.

Dated: **Mahendra Swain**

Reg. No. 11307913

It is certified that the above statement is correct to the best of my knowledge and belief.

Dated: **Mr. Abhishek Kumar Srivastava**

Lovely Professional University

Phagwara, Punjab

CERTIFICATE

This is to certify that **Mahendra Swain** (11306135) has completed objective formulation of his Dissertation-II titled, "**DESIGN OF EMBEDDED LINUX BASED VOICE CALLING DEVICE**" under my guidance and supervision. To the best of my knowledge, the present work is the result of their original study and research. No part of the report has ever been submitted for any other degree at any University. The thesis is fine for the submission and fulfillment of the conditions for the award of degree Masters of Technology.

Mr. Abhishek Kumar Srivastava
Assistant Professor
ECE Department

Lovely Professional University

Phagwara, Punjab

Date:

LIST OF ABBREVIATIONS

TFT	Thin Film Transistor
GSM	Global System for Mobile
GPIO	General Purpose Input Output
GUI	Graphical User Interface
Tx	Transmitter
Rx	Receiver
BSP	Board Support Packages
GPU	Graphics Processor Unit
SoC	System on Chip
OS	Operating System

LIST OF FIGURES

Fig 1 Basic Block Diagram…………………………………………………………..1

Fig 2 Raspberry Pi Model B………………………………………………………….8

Fig 3 GPIO pins configuration………………………………………………………..9

Fig 4 Pi TFT 2.8 inch (320x240)…………………………..…..10

Fig 5 GSM SIM 900 A Modem…………………………………………………...10

Fig 6 SD-Card for Raspberry Pi……………………………………………………...11

Fig 7 Raspberry Pi first boot………...……………………………………………….15

Fig 8 GUI for Raspberry Pi………………………………………………………….16

Fig 9 Image building using Bitbake………………………………………………….21

Fig 10 BitBake core image building…………………………………………………22

Fig 11 BitBake successful image built……………………………………………….23

Fig 12 Successful build after removing error……………………………………..…24

Fig 13 Hob main menu selection…………………………………………………….27

Fig 14 Selection of hob target and basic image……………………………...………28

Fig 15 Recipe work flow…………………………………………………………….30

Fig 16 Hob default package for image……………………………………………….31

Fig 17 Hob while building the packages……………………………………………..32

Fig 18 Editing recipes………………………………………………………………..32

Fig 19 Hardware connections………………………………………………………..33

Fig 20 Physical Hardware interconnections…………………………………………34

Fig 21 Serial to parallel conversion circuit. …………………………………………35

Fig 22 SPI bus with single master single slave………………………………………36

Fig 23 ITEAD B02 3.2” LCD…………………………………………………………..36

Fig 24 Pi TFT 2.8” interfaced with Raspberry Pi……………………………………………37

Fig 25 Python based number dialer on Raspberry Pi………………………………………..38

Fig 26 Number dialer pad……………………………………………………………………39

Fig 27 Prototype during voice call…………………………………………………………..40

TABLES OF CONTENTS

APPROVAL……………………………………………………………………………………i

ABSTRACT…………………………………………………………………………………ii

ACKNOWLEDGEMENT……………………………………………………………………..iii

DECLARATION………………………………………………………………………………iv

CERTIFICATE……………………………………………………………………..………..v

LIST OF ABBREVIATIONS…………………………………………………………….........vi

LIST OF FIGURES……………………………………………………………………...vii

CHAPTER 1 INTRODUCTION………..…………………………………………………...1

CHAPTER 2 LITERATURE REVIEW…..………………………………………….........3

CHAPTER 3 EMBEDDED HARDWARE AND SOFTWARE SETUP……………………...6

CHAPTER 4 RESEARCH METHODOLOGY………………………………………………...17

CHAPTER 5 RESULT AND DISCUSSION…………………………………………………36

CHAPTER 6 CONCLUSION AND FUTURE SCOPE……………………………………...41

CHAPTER 7 REFERENCES…………………………………………………………………42

CHAPTER 1

INTRODUCTION

The aim of the thesis is to design of an "Embedded Linux Based Voice Calling Device" using Raspberry Pi. The Raspberry Pi is the product by the foundation of Raspberry Pi. It is based on an ARM microcontroller produced by Broadcom that was originally designed for set-top box and having many more applications in different field. It is a cost effective and multi purposed device. This research focus on developing Embedded Linux based prototype which can primarily make voice calls for which we need to configure Linux Kernel and customize Linux packages as per application requirements and calibrate Pi TFT with Raspberry Pi for proper touch sensitivity. The Raspberry pi has 26 GPIO pins through which Pi TFT 2.8" can be interfaced. Pi TFT requires display driver which must be available in Linux kernel for its smooth functioning. In case of Raspbian OS, we can add easily add display driver to it. For making voice calls, GSM modem SIM 900A is used has to be interfaced with Raspberry pi through TX and RX pins present in GPIO. Raspberry Pi is relatively low power hungry device which can perform multitasking when compared to its opponents (of similar specs). The Model A Pi draws power of 300mA, which shows the whole board can be powered from USB port. The Model A Pi consumes power of 1.4 watts, whereas the Model B consumes at most 3.5watts. If we add lot of power-consumable devices (like Pi TFT and GSM modem) to our board, the amount of power consumption would increase considerably and we have to look for alternative measures to reduce power, ex: sleep mode etc.

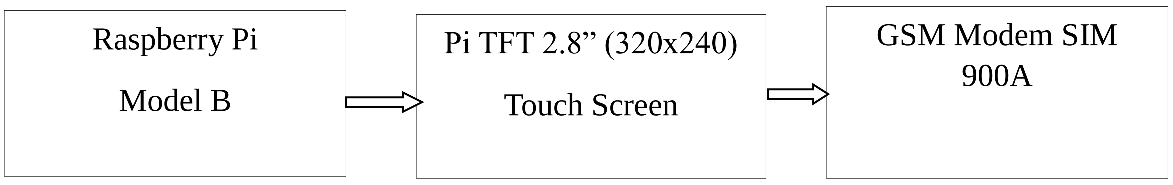


Fig 1: Basic Block Diagram

CHAPTER 2
LITERATURE REVIEW

1. Building Linux Kernel for Raspberry Pi Hannu Flinck Nokia Solutions and Networks +358504839522 hannu.flinck@saunalahti.fi:- This paper discuss about how Linux kernel is built to ARM Raspberry Pi platform to enable kernel porting. The Linux kernel had been built using available tool-chain via different methods for QEMU (target). This paper describes the kernel basics, configuration files and the important ARM-specific files. Initially, this file shows how to configure and compile an embedded Linux kernel using gcc-4.6-arm-linux-gnueabi cross compiler. Then they had configured and compiled Linux kernel for QEMU (taget). QEMU offers test and development environment that emulates the target environment without a need for real hardware. Finally, they tell how to prepare a Raspberry Pi system through Yocto embedded Linux tool chain that results into full SD card image as a boot media. [1]

2. Using Phase Behavior in Scientific Application to Guide Linux Operating System Customization, 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS’05) 1530-2075/05:- This paper, represents how to design a system which can generate application-specified Linux images automatically for scientific applications that execute using batched cluster resources. Key to this approach is the use of recurring patterns in program performance, i.e., phase-behavior that can be exploited potentially to guide automatic Linux customization and to enable significantly higher levels in program performance. [2]

3. Porting the Linux Kernel to Arm System-On-Chip and Implementation of RFID Based Security System Using ARM, International Journal of Advanced Research in Computer Science and Software Engineering:- This paper emphasizes the porting of the Linux kernel to an ARM board which is implementing the RFID (Radio Frequency Identification) based security system using the ARM board. ARM system-on-chip is the best platform for reducing the risk and the cost of SOC designs and thus accelerating the speed, accuracy and flexibility. ARM boards are the combination of FPGA and ASIC technology and hence provides the optimal solution in terms of speed, cost and time. [3]The embedded modules based on ARM delivers varied tasks such as process management, memory management and peripheral interfaces. Linux being an open source domain provides an additional edge to the embedded systems and as Linux can be freely downloaded so it can be compiled for any system architecture including ARM. For new CPU architectures OS has to be configured, compiled burnt to the core and ported to the target platform. [3]

Source: http://www.ijarcsse.com/docs/papers/Volume_3/5_May2013/V3I5-0184.pdf

4. Studying Main Differences between Android & Linux Operating Systems, International Journal of Electrical & Computer Sciences IJECS-IJENS Vol:12 No:05. October 2012:-

This paper gives a comparative analysis between Android and Linux operating system. Some of the key differences that this paper states are:

i) Android OS is an open source developed by Android, which is now owned by Google, Inc. whereas Linux is developed as an open source operating system under the GNU project by Linus Torvalds and others. [4]

ii) Android is developed for Mobile Internet Devices and mobile phones whereas Linux is developed for desktops/laptops/servers.

iii) The Android operating system has its own C library called Bionic whereas Linux systems use GNU C library.

iv) The Android systems use flash memory instead of hard drives while the standard Linux systems use magnetic drives.

v) The Android systems have their own power manager whereas the Linux systems use APM and ACPI to manage the power. [4]

5. Porting the Linux Kernel to a New ARM Platform, Wookey and Tak-Shing, Aleph One • www.aleph1.co.uk: - This paper explains the importance of .config files in Linux kernel porting in to ARM based platforms. It also tell the importance of linux/arch/arm/ directory and the codes present in this directories are kernel, mm, arch, hardware, boot, defconfig etc. The certain options like menuconfig, kconfig are used to update the kernel whereas the defconfig generate new Linux kernel according to the target. [5] Here the list the most imperative files, and portray their motivation and the kind of things you ought to put in them. It looks overwhelming to begin with yet a large portion of what is needed is simply an issue of filling in the numbers suitable to our equipment. Since such a variety of distinctive machines are bolstered it is uncommon that we need to compose much new code - about everything can be taken from a suitable giver machine. This is less demanding to do in the event that we know which machines have a comparative construction modeling to our own.

6. Raspberry pi user guide Eben Upton, co-creator of the raspberry pi: This article gives the detail architecture of Raspberry pi, what are its possible uses and how to make it useful. This guide also gives a knowhow on using GPIO ports, SPI in Raspberry pi, etc. It defines the significances of DHCP server and VNC, in case of accessing the raspberry pi terminal on Laptop Screen through Ethernet cable.

7. Analysis of TOI (Things on Internet) Industrial Monitoring System on Raspberry pi Platform, International Journal of Computer Science and Mobile Applications, Vol.2 Issue. 11, November- 2014, pg. 33-40 ISSN: 2321-8363: This paper explains controlling real time system from remote location using internet and also explains some sophisticated features to monitor using smart phones from remote area locations. This research paper was developed to produce web based temperature, humidity & pressure monitoring system that allows to continue monitor these parameters. This system allowed the data to be anytime & anywhere from the internet after we login into webpage. This research paper also concludes that user can set limit range of the above parameters & if these parameters goes beyond that value, it will turn off devices. With Ethernet-based, internet-enabled instrumentation, remote access can be anywhere a smartphone has a signal

Source: http://ijcsma.com/publications/november2014/V2I1113.pdf

8**. Linux kernel Development Manual, Yocto Project:** This manual provides some background information of Linux Kernel metadata. For building Linux kernel for Raspberry pi requires different metadata depending on the applications, we are going to develop. (e.g. Meta-raspberry)**.** The repositories helps us to create new recipes. This manual also signifies concept of patching, configuring files and the Board Support Packages available for Raspberry pi. The Yocto Project also provides a powerful set of kernel tools for managing Linux kernel sources and configuration data. We can use these tools to make a single configuration change, apply multiple patches, or work with your own kernel sources. The corresponding changes can be done as per requirement in the kernel level.

9. Smart Projectors using Remote Controlled Raspberry Pi International Journal of Computer Applications (0975 – 8887): This paper proposes the use of Raspberry Pi and its web interface, to store files that have been sent from remote sources and view these power point files or Portable Document Files (PDF) on the projector. The proposed system aims to substitute laptops with Raspberry Pi which will not only drastically reduce the cost involved, but also will help achieving quality of service as the system will consume a smaller amount of power, yet will provide the same functionality as any other similar system does. The proposed system will be controlled by using a smart phone based remote control, thus adding to the convenience. [7]

Source: http://research.ijcaonline.org/volume82/number16/pxc3892250.pdf

CHAPTER 3

EMBEDDED HARDWARE AND SOFTWARE SETUP

The equipment needed for this projects are mentioned in below:

3.1 Embedded Hardware Setup

1. Raspberry pi Model B
2. piTFT LCD 2.8" (Adafruit)
3. GSM modem SIM 900A
4. Micro SD card (8GB)
5. SD card reader
6. USB keyboard and mouse
7. Micro USB adapter for Power
8. Metal mini speaker
9. Mini Metal Microphone
10. 13x2 Serial Bus
11. Patch cable

Raspberry Pi Model B: The Raspberry pi is a cost effective minicomputer. It operates on 700 MHz frequency, with ARM 11 series controller. It supports USB ports, Model B has two USB ports, Ethernet port, Audio jack 3.5 mm and SD card slot for installation with any supported OS. Raspberry pi supports USB, Ethernet and HDMI ports, The 26 GPIO (general purpose input output) the GPIO pins have SPI, I²C, and serial UART, 3V3 and 5V power. The pins use a 3V3 logic level and are not able to tolerate logic levels of 5V.

In comparison to Raspberry Pi, other boards which are available in market are Beagle Bone, Beagle Board, Panda Board, Odroid Board and many more. The Raspberry Pi has been chosen for this research because it encourage the purpose of learning, innovation and experimental studies with vibrant support community and relatively cheaper development cost. In this project, Raspberry pi serves our purpose well as of providing Linux based platform where we can add features like GUI for easy handling, Python based games, web browser, music player, and many more as per need. The different parts of the Raspberry pi has been shown in the figure 1

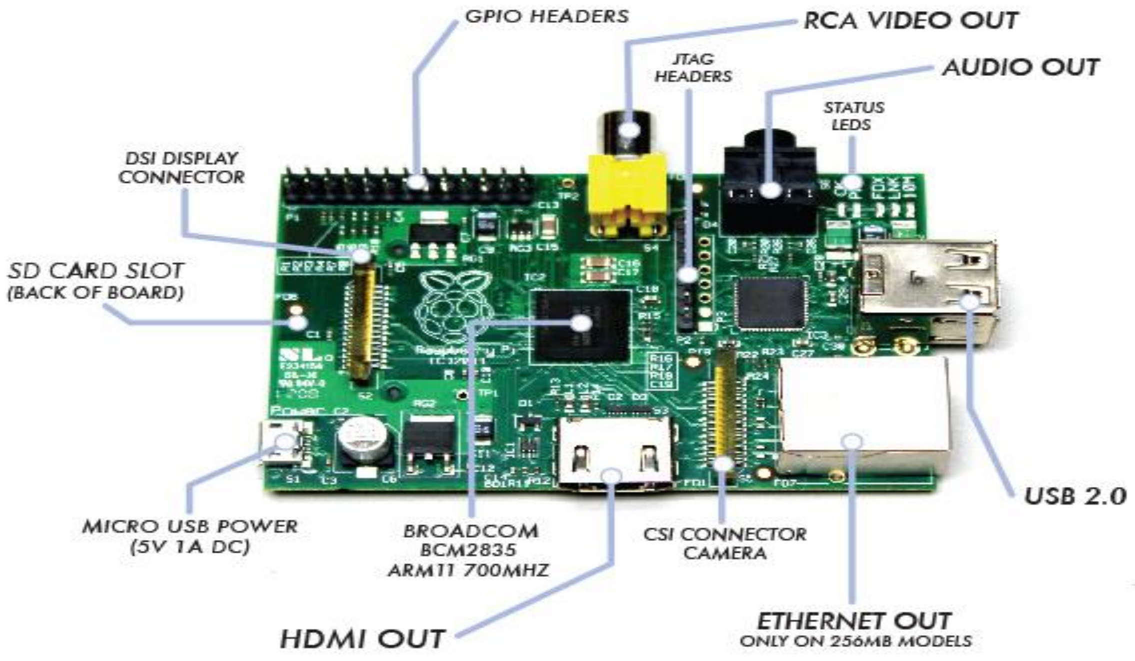

GPIO HEADERS
RCA VIDEO OUT
JTAG HEADERS
AUDIO OUT
STATUS LEDS
DSI DISPLAY CONNECTOR
SD CARD SLOT (BACK OF BOARD)
USB 2.0
MICRO USB POWER (5V 1A DC)
BROADCOM BCM2835 ARM11 700MHZ
CSI CONNECTOR CAMERA
HDMI OUT
ETHERNET OUT ONLY ON 256MB MODELS

Fig 2: Raspberry Pi Model B

GPIO Pins: The raspberry Pi Model b has 26 GPIO pins for interfacing with other peripherals. The GPIO pins configuration is shown in figure no 3. The GPIO voltage level is maintained at 3.3 V, this is not tolerable to 5 volt power supply.

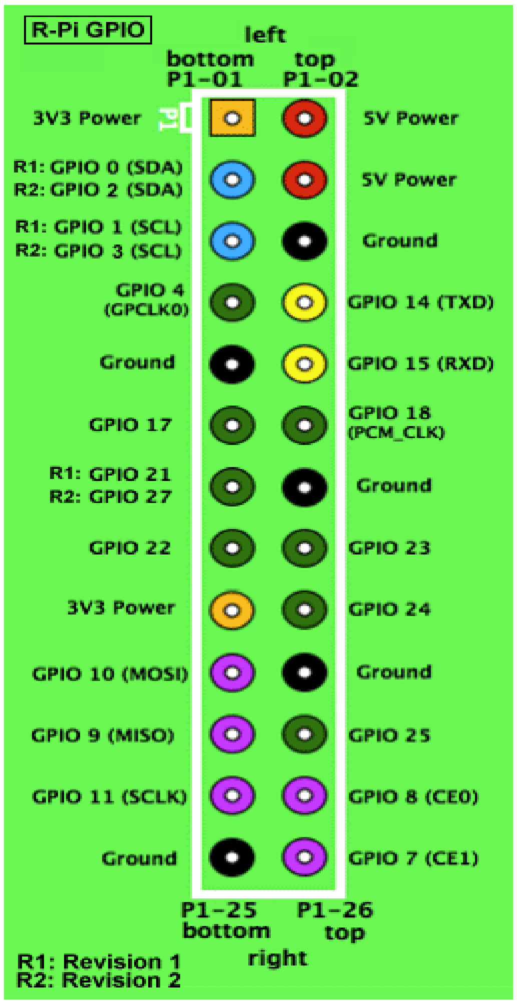


Fig 3. GPIO pins configuration

Pi TFT 2.8”: The piTFT 2.8" display comes with 320x240 16-bit color pixels and a resistive touch sensitive. It uses the high speed SPI interface to the Raspberry Pi and can use the mini display as a console. We have tested with ITEAD 3.2” ITB 02 LCD with Raspberry Pi but we get some compatibility confliction. So, we have chosen piTFT 2.8” which is compatible with Raspberry pi. Before using for our application the piTFT driver has to be installed to the Linux Kernel we are going to use. The calibration has to be done for better touch response. As per our requirement the display format from landscape to portrait can be customized by changing the angle of rotation by 90 degree.

Fig 4: piTFT2.8” (320x240)

GSM Modem SIM900A

This GSM Modem SIM 900A can accept any GSM network operator through SIM card and provides flexibility to make call and receive, message with its own unique phone number.

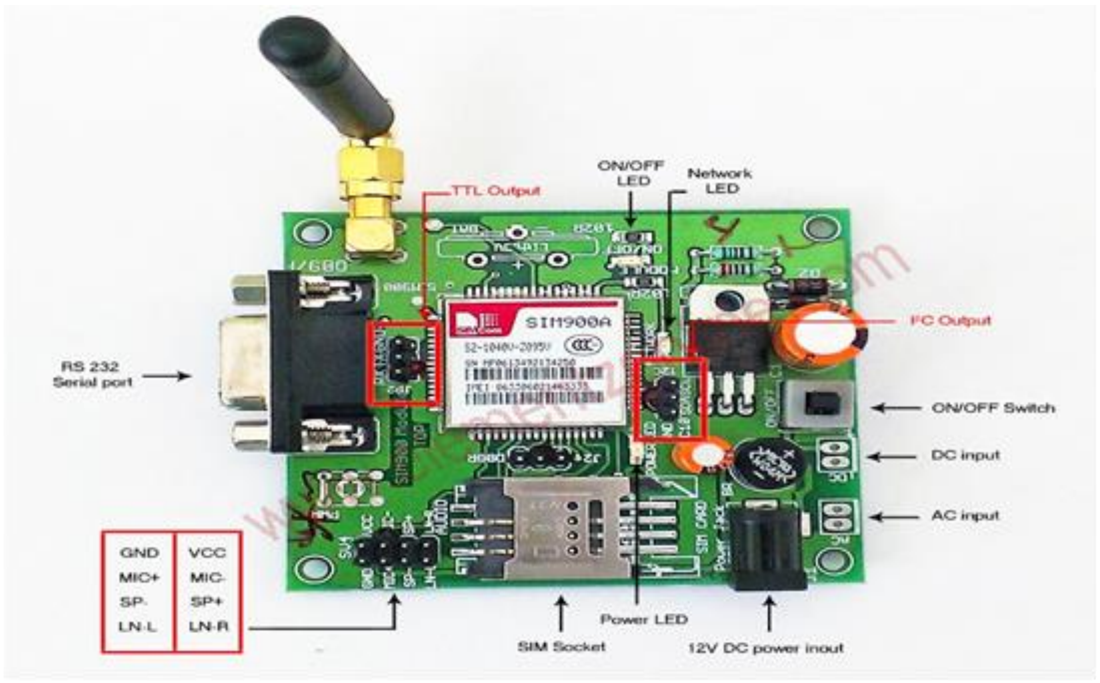


Fig 5: GSM SIM 900A Modem

SD-Card: It plays a vital role in Raspberry Pi for storing Operating system and boot loader. The SD-Card used in Raspberry Pi Model B is the large one. In case of micro SD-Card we can use SD- Card reader to fix the card in to the socket available in Raspberry Pi. Most preferable SD-card for raspberry pi is of class 6 and size of 8GB. Depending on the speed the SD-Card are of different types of class like class 4 which attains maximum speed of 4 MB/S whereas class 10 gives the speed of 10MB/s. It is always recommended to take genuine SD-Card and shutdown properly the raspberry pi board after remove the SD-Card from the socket.

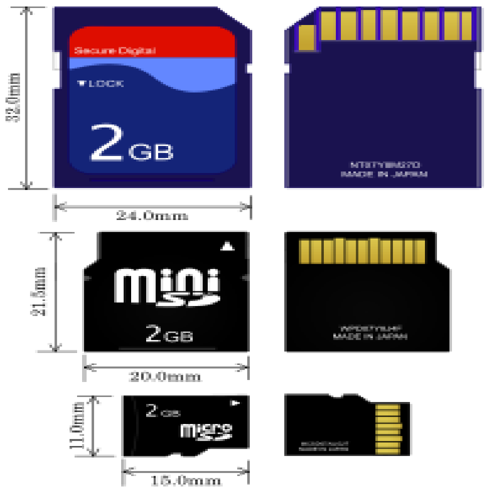


Fig 6: SD-Card for Raspberry pi

Power USB adapter: In order to power up the Raspberry Pi board 5 volt is required that can be provided through the USB adapter. It is always referred to take good quality USB power adapter for raspberry pi.

Audio interface with GSM: The mini metal micro-phone interfaced with GSM for audio input and mini speaker has been used for audio output for the device. The Mini Speaker of low resistance is preferred for audio interface with GSM Modem.

Patch or LAN cable: The network access of the Raspberry Pi on Laptop interfaced through the patch cable. Through a particular IP address the Raspberry Pi able to communicate with the Laptop. The LAN cable mostly used for wired network connection with others computers. Since Raspberry pi supports Ethernet so that can be connected to the other computer or Laptop in order to achieve terminal mode of Raspberry pi on laptop.

13x2 Serial Bus: The 13x2 serial bus is connected to the GPIO pins of the Raspberry Pi to interface peripherals to it. The serial bus allows to connect peripherals from GPIO pins.

3.2 Embedded Software Setup:

1. Linux OS (Ubuntu 14.04)
2. Raspbian OS (from www.raspberrypi.org)
3. Yocto for image building
4. Python

Linux OS: The Linux platform has been chosen for this research work. The Linux distribution Ubuntu 14.04 is used in host system. Since the research work is Linux based so Linux is the appropriate platform for compilation. The Linux gives other advantages like command line mode which is quite a powerful tool in many aspects.

Raspbian OS: Raspbian operating system is a Debian based used in Raspberry pi. It contains all the packages required for the Raspberry pi basic operations. Raspbian is basically designed for the Desktop environment. The packages can be customized as per our requirement. The package customization has been explained in the methodology.

Yocto Project for target image build: Yocto project has been used in this research for building of custom Linux operating system for raspberry pi. The Yocto Project is freely available Linux image building tool which provide us to create or build new Linux Image for our target. The Yocto Linux kernel is independent of hardware platforms ARM, PPC, MIPS, x86, and x86-64. The recommended Linux distributions for Yocto Projects are Fedora, openSUSE, Debian, Ubuntu, or CentOS .We need minimum of 50 GB of free space on the disk in order to building images. The Open Embedded build system requires the some packages exist on our development system (e.g. Python 2.6 or 2.7).The source repository for Yocto project can be downloaded from Yocto project.

By using command **$ git clone git://git.yoctoproject.org/poky**

The Yocto project used two basic methods to build image for Raspberry pi one is using Bitbake and another one is using hob.

BitBake: BitBake was originally a part of the Open Embedded project. BitBake has certain significances .These are BitBake, a generic task executor BitBake is a generic task execution engine that allows shell and Python tasks to be run efficiently and in parallel while working within complex inter-task dependency constraints. BitBake executes tasks according to provided metadata that builds up the tasks. Metadata is stored in recipe (.bb) and related recipe "append" (.bbappend) files, configuration (.conf) and underlying include (.inc) files, and in class (.bbclass) files. The metadata provides BitBake with instructions on what tasks to run and the dependencies between those tasks. BitBake includes a fetcher library for obtaining source code from various places such as local files, source control systems, or websites. Some important original goals for BitBake are: Handle cross-compilation. handle inter-package dependencies (build time on target architecture, build time on native architecture, and runtime).Support running any number of tasks within a given package, including, but not limited to, fetching upstream sources, unpacking them, patching them, configuring them, and so forth. Linux kernel configuration for target systems like Raspberry pi and beagle bone board.

Hob: The hob is graphical user interface for the Bitbake. Which is powerful and easy to handle all the packages required for the Raspberry Pi. The image building process using by the hob has been explained in research methodology.

Python: python is a powerful programming language used in Raspberry pi. The advantages to use python in this work is it makes bridge between the software computing and real time.

The python 2.7 version has been used. The syntax is simple, standard, emphasis on readability. Python version 2 and 3 both are recommended in Raspberry pi. It allows us to create Graphical User Interface in real world where as other programming languages are not. The execution, compilation are easier than others.

3.3 Experimental Works

- As per experimental work is concerned, I have flashed the SD card with Raspbian OS with suitable display driver both using windows platform as well as Linux.
- We have interfaced Raspberry Pi with my laptop screen to access Raspbian OS in text mode and access Raspbian OS file system
- We have also accessed Raspbian OS in graphical mode on laptop screen using VNC server
- Finally, we have witness the Raspbian OS on Pi TFT 2.8" from Adafruit and made small python application run on Raspbian OS.

3.3.1 Flashing of Raspbian in to SD card

Using win Disk-32 manager.exe on Windows

- We have to download Raspbian from http://www.raspberrypi.org/downloads
- Extract the zip file on to a directory
- SD card was inserted Laptop
- Image file was selected (example 2012-10-28-wheezy-raspbian.img) to be written to the SD card
- Target device has been selected for "Write to device" after sometime the SD card was ready

3.3.2 Using the Linux command line

In case of Linux, we can use "dd" tool can overwrite any partition of SD card. Download the zip file containing the image from a mirror or torrent http://www.raspberrypi.org/download extracting the image, with unzip ~/2014-10-11-wheezy-raspbian.zip

Run $**df -h** to see what devices are currently mounted

$umount /dev/sdd1
$sudo dd if=*path_of_your_image.img* of=/dev/disk*n* bs=1M

Note that if we are not logged in as root you will need to prefix this with "sudo" Instead of "dd" we can use "dcfld"; it will give a progress report about how much has been written.Now we can inserted the Flashed SD card to SD card socket given on Raspberry pi board.

3.4 Set-up of Raspberry Pi to access laptop screen as command-line display on Ubuntu 12.04

We have performed this work on Ubuntu, the procedure I am mentioning below first I have connected Raspberry pi to Laptop through Ethernet cable. I edited the wired connection setting, IPv4Settings- Method: Shared to other computers, click Save. Now, my laptop is able to find Raspberry Pi. .I have used command "$ifconfig", it prints out "inet addr: 10.42.0.1 Bcast: 10.42.0.255 Mask: 255.255.255.0".Now by using nano editor the following set up was written and saved.

```
auto lo
iface lo inet loopback
```

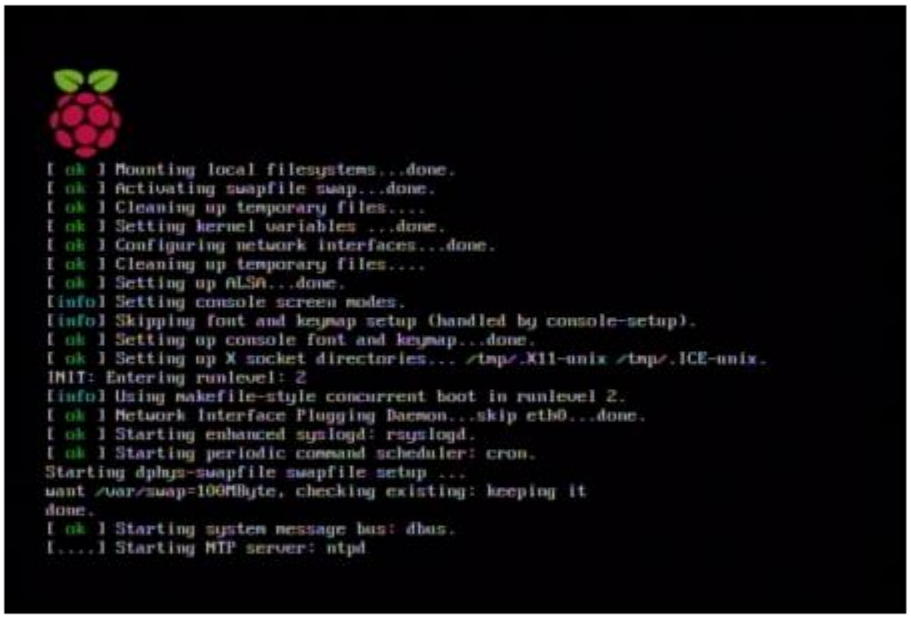
iface eth0 inet static
address 10.42.0.2
netmask 255.255.255.0

Initially, laptop /etc/network/interfaces setup is given
auto lo wlan0
iface lo inet loopback
$sudo apt-get install nmap
$sudo nmap –sP 10.42.0.2-254

Using ssh command as "$ssh pi@10.42.0.2" on the laptop, username is pi, then enter my password: raspberry, which is default password, and finally my Raspberry Pi with my laptop connection established on Ubuntu 12.04.

3.5 First Boot of Raspberry pi

After inserting the SD-card in to the raspberry pi slot for SD card, it will start booting is given below in the figure.



```
[ ok ] Mounting local filesystems...done.
[ ok ] Activating swapfile swap...done.
[ ok ] Cleaning up temporary files....
[ ok ] Setting kernel variables ...done.
[ ok ] Configuring network interfaces...done.
[ ok ] Cleaning up temporary files....
[ ok ] Setting up ALSA...done.
[info] Setting console screen modes.
[info] Skipping font and keymap setup (handled by console-setup).
[ ok ] Setting up console font and keymap...done.
[ ok ] Setting up X socket directories... /tmp/.X11-unix /tmp/.ICE-unix.
INIT: Entering runlevel: 2
[info] Using makefile-style concurrent boot in runlevel 2.
[ ok ] Network Interface Plugging Daemon...skip eth0...done.
[ ok ] Starting enhanced syslogd: rsyslogd.
[ ok ] Starting periodic command scheduler: cron.
Starting dphys-swapfile swapfile setup ...
want /var/swap=100MByte, checking existing: keeping it
done.
[ ok ] Starting system message bus: dbus.
[....] Starting NTP server: ntpd
```

Fig 7: Pi First Boot

Now, it will as for the Login id and password, the

Login id: ~$ pi

Password:~$ raspberry

These are the default id and password. For Graphical User interface we have to write the command “startx” the x window as shown in figure 8.After the experiment setup complete, now we will able to see the X window of raspberry pi given below in figure

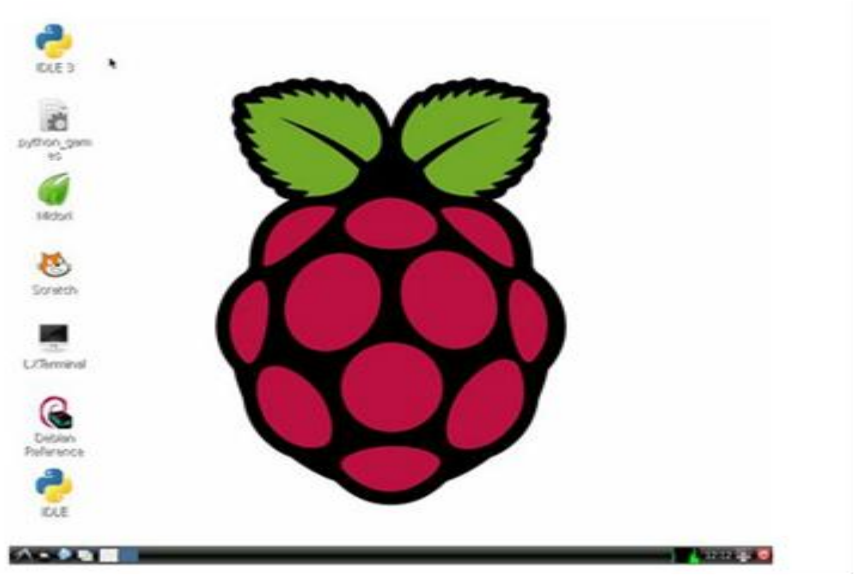


Fig 8: GUI of Raspberry pi

CHAPTER 4
RESERCH METHODOLOGY

4.1 Top-Down approach

4.1.1 Using Raspbian OS

Many of the applications designed for Embedded Systems does not require X_{11} and we want to keep the files which are really needed in to the SD card to reduce memory foot print as possible. So we decided to prepare minimal image for Raspberry Pi. Those packages are not needed for my application we can remove that packages. The different steps involved are

Step 1: Fresh Raspbian image with piTFT driver has been flash on SD-Card

Step 2: Customization of Raspbian as per required

Step 3: Add/modify Python based GUI for voice calling device

Step 4: GSM modem interface

Step 5: Result as per required

4.1.2 Removal of unwanted Packages

To remove the packages we have to access raspberry pi terminal on host through SSH.

Now, we can check the file system size used avail use% mounted on

```
$pi@raspberry~ df –h
```

X11 windows, omxplayer, LXDE, Scratch, Wolfram is not needed we can remove this package by invoking the command

```
$ sudo apt-get remove x11-common midori lxde python3 python3-minimal

$ sudo apt-get remove lxde-common lxde-icon-theme omxplayer raspi-config

$ sudo apt-get autoremove
```

After removing the extra packages from the raspbian the size becomes 931MB .The size can be reduced up to 84 MB but in our case we need python, pygame packages which acquire memory space up to 380MB.

4.2 Bottom-Up Approach

Using Yocto Project

.4.2.1 Yocto on Raspberrypi

The Yocto project can be used to build customized Linux kernel for Raspberry Pi. Since Yocto supports ARM platform. The different steps involves in it are

Step 1: Download Yocto for Raspberry Pi

Step 2: Package selection for custom Linux OS

Step 3: Build the packages and generate Linux image for it .If build succeed? Then add piTFT driver in order to interface with piTFT 2.8”

Step 4: Flash the image on to SD-Card and plug in to Raspberry Pi.

The following BSP layer naming schemes are used in Yocto: we are going to use meta-raspberrypi, which is master branch of meta-raspberrypi.

```
$ cd poky
 $ “git clone git://git.yoctoproject.org/meta-raspberrypi.git”
```

4.2.3 Configuring the host

Here, we have used Ubuntu 12.04 to generate the image for that we need to install the required packages on Ubuntu to build the Yocto image. The command to install the essential Ubuntu packages is:

```
$”sudo apt-get install gawk wget git-core diffstat unzip texinfo \
 build-essential chrpath libsdl1.2-dev xterm”
```

4.2.4 Preparing the environment

First, We have downloaded Yocto with this command:

```
$git clone git://git.yoctoproject.org/poky
```

Then we have move into poky directory and download the required layers: meta-raspberrypi and with this commands:

```
$cd poky
$git clone git://git.yoctoproject.org/meta-raspberrypi
```

To create our build directory and configure the system to build a Yocto image for a Raspberry Pi board with the command:

```
 $.oe-init-build-env raspberrypi
```

4.2.5 Configuring Linux Kernel Using Bitbake:

The development system will be created into the raspberrypi directory. In the conf directory there are the two configuration files: local.conf and bblayers.conf. The main parameter that we can customize in the local.conf are:

```
MACHINE? = "raspberrypi"
DL_DIR? = "/home/mahi/downloads"
IMAGE_FEATURES += "package-management"
```

MACHINE is used to specify the target type. DL_DIR can be used to specify the directory where source tarballs are downloaded during the build process. IMAGE_FEATURES is used to install rpm tool, to add to the image new packages. In the bblayers.conf file you have to add the path for the new layers to BBLAYER variable:

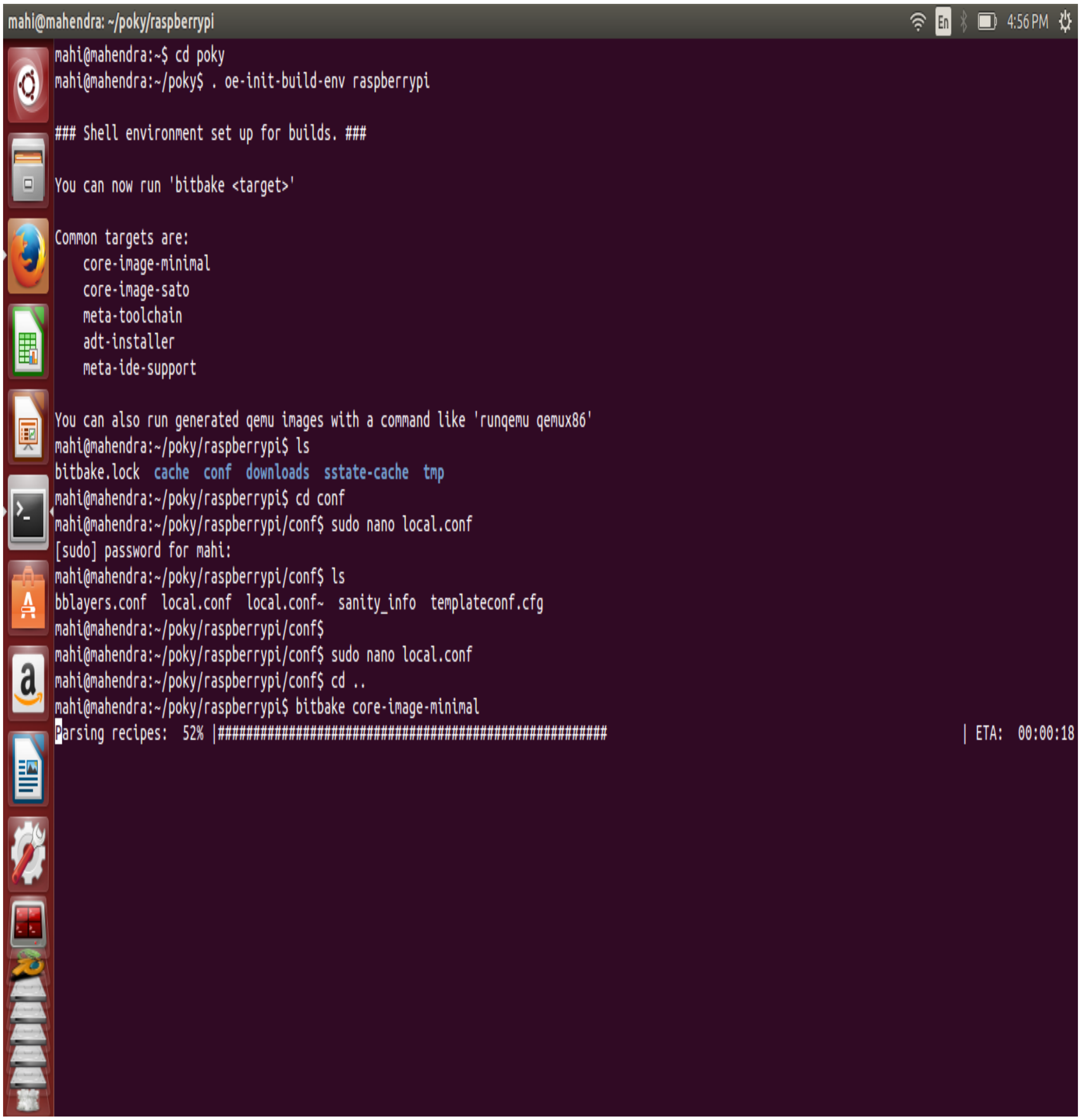
```
BBLAYERS ?= " \
  /home/mahi/yocto/poky/meta \
  /home/mahi/yocto/poky/meta-yocto \
  /home/mahi/yocto/poky/meta-yocto-bsp \
  /home/mahi/yocto/poky/meta-raspberrypi \
"
```

In this case, We have added the last one line, with meta-raspberrypi and meta-openembedded layers.

4.2.6 Build and install the system

To build the system we have to use the command bitbake from
the /home/mahi/yocto/poky/raspberrypi/ directory. The command was used “bitbake rpi-hwup-image “to build a minimal system, without the graphical environment .The images was in poky/raspberrypi/tmp/deploy/images/raspberrypi directory. The image named as “rpi-hwup-image-raspberrypi.rpi-sdimg”.Now, the image can be flush to SD-card and plugged to Raspberry pi perform respective tasks.

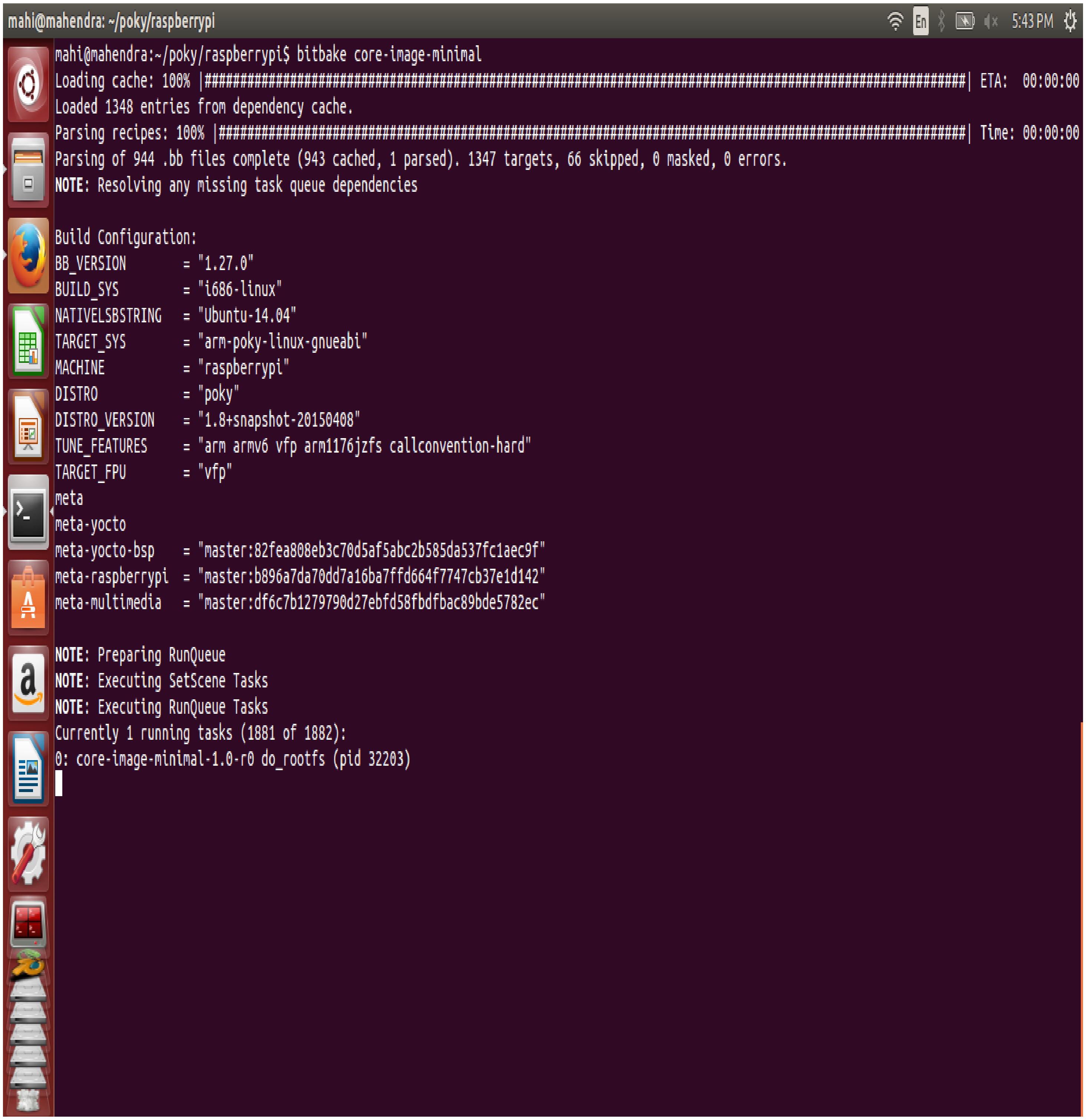
Snapshots during building image using Bitbake: During the build of image these snapshots are taken on native system. These snapshots are taken during the building the custom image for raspberry Pi.



```
mahi@mahendra:~$ cd poky
mahi@mahendra:~/poky$ . oe-init-build-env raspberrypi

### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    core-image-sato
    meta-toolchain
    adt-installer
    meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'
mahi@mahendra:~/poky/raspberrypi$ ls
bitbake.lock  cache  conf  downloads  sstate-cache  tmp
mahi@mahendra:~/poky/raspberrypi$ cd conf
mahi@mahendra:~/poky/raspberrypi/conf$ sudo nano local.conf
[sudo] password for mahi:
mahi@mahendra:~/poky/raspberrypi/conf$ ls
bblayers.conf  local.conf  local.conf~  sanity_info  templateconf.cfg
mahi@mahendra:~/poky/raspberrypi/conf$
mahi@mahendra:~/poky/raspberrypi/conf$ sudo nano local.conf
mahi@mahendra:~/poky/raspberrypi/conf$ cd ..
mahi@mahendra:~/poky/raspberrypi$ bitbake core-image-minimal
Parsing recipes:  52% |#######################################################                                                    | ETA:  00:00:18
```

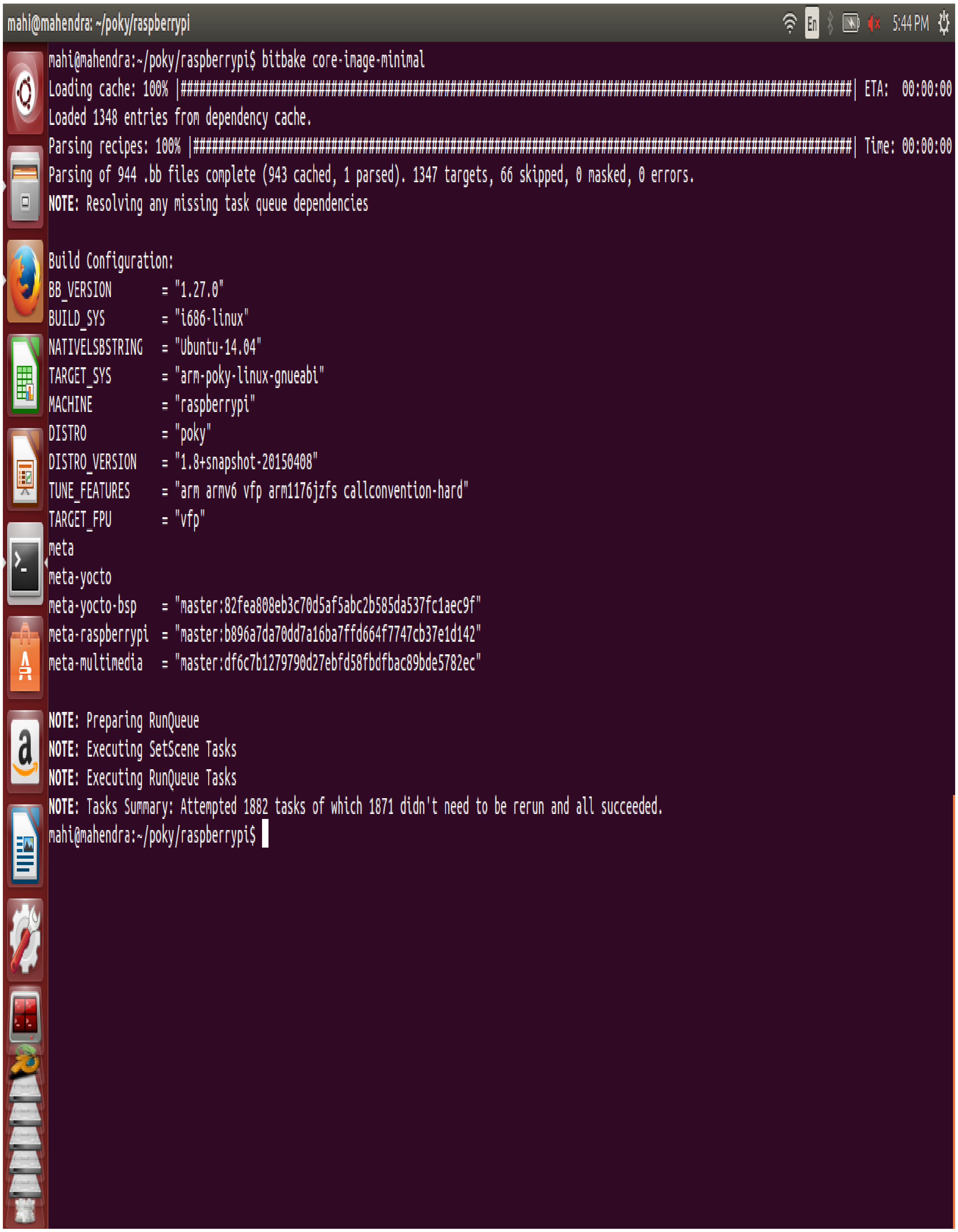
Fig 9: Image building using BitBake

`



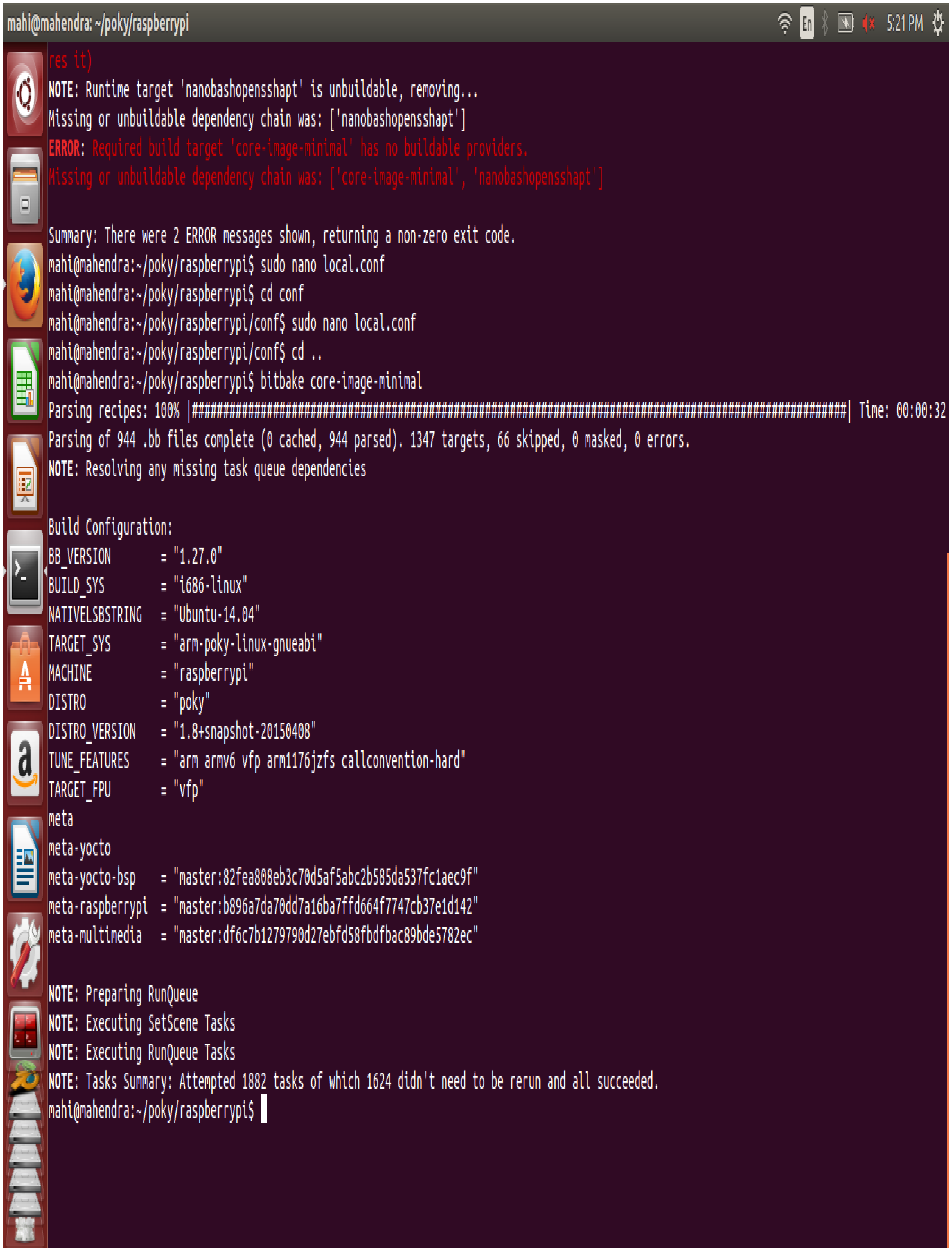
```
mahi@mahendra:~/poky/raspberrypi$ bitbake core-image-minimal
Loading cache: 100% |############################################################################################| ETA:  00:00:00
Loaded 1348 entries from dependency cache.
Parsing recipes: 100% |##########################################################################################| Time: 00:00:00
Parsing of 944 .bb files complete (943 cached, 1 parsed). 1347 targets, 66 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION        = "1.27.0"
BUILD_SYS         = "i686-linux"
NATIVELSBSTRING   = "Ubuntu-14.04"
TARGET_SYS        = "arm-poky-linux-gnueabi"
MACHINE           = "raspberrypi"
DISTRO            = "poky"
DISTRO_VERSION    = "1.8+snapshot-20150408"
TUNE_FEATURES     = "arm armv6 vfp arm1176jzfs callconvention-hard"
TARGET_FPU        = "vfp"
meta
meta-yocto
meta-yocto-bsp    = "master:82fea808eb3c70d5af5abc2b585da537fc1aec9f"
meta-raspberrypi  = "master:b896a7da70dd7a16ba7ffd664f7747cb37e1d142"
meta-multimedia   = "master:df6c7b1279790d27ebfd58fbdfbac89bde5782ec"

NOTE: Preparing RunQueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
Currently 1 running tasks (1881 of 1882):
0: core-image-minimal-1.0-r0 do_rootfs (pid 32203)
```

Fig 10: BitBake Core image building



```
mahi@mahendra:~/poky/raspberrypi$ bitbake core-image-minimal
Loading cache: 100% |############################################################################################| ETA:  00:00:00
Loaded 1348 entries from dependency cache.
Parsing recipes: 100% |##########################################################################################| Time: 00:00:00
Parsing of 944 .bb files complete (943 cached, 1 parsed). 1347 targets, 66 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION        = "1.27.0"
BUILD_SYS         = "i686-linux"
NATIVELSBSTRING   = "Ubuntu-14.04"
TARGET_SYS        = "arm-poky-linux-gnueabi"
MACHINE           = "raspberrypi"
DISTRO            = "poky"
DISTRO_VERSION    = "1.8+snapshot-20150408"
TUNE_FEATURES     = "arm armv6 vfp arm1176jzfs callconvention-hard"
TARGET_FPU        = "vfp"
meta
meta-yocto
meta-yocto-bsp    = "master:82fea808eb3c70d5af5abc2b585da537fc1aec9f"
meta-raspberrypi  = "master:b896a7da70dd7a16ba7ffd664f7747cb37e1d142"
meta-multimedia   = "master:df6c7b1279790d27ebfd58fbdfbac89bde5782ec"

NOTE: Preparing RunQueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 1882 tasks of which 1871 didn't need to be rerun and all succeeded.
mahi@mahendra:~/poky/raspberrypi$
```

Fig 11: BitBake successfully image built



```
mahi@mahendra: ~/poky/raspberrypi
res it)
NOTE: Runtime target 'nanobashopensshapt' is unbuildable, removing...
Missing or unbuildable dependency chain was: ['nanobashopensshapt']
ERROR: Required build target 'core-image-minimal' has no buildable providers.
Missing or unbuildable dependency chain was: ['core-image-minimal', 'nanobashopensshapt']

Summary: There were 2 ERROR messages shown, returning a non-zero exit code.
mahi@mahendra:~/poky/raspberrypi$ sudo nano local.conf
mahi@mahendra:~/poky/raspberrypi$ cd conf
mahi@mahendra:~/poky/raspberrypi/conf$ sudo nano local.conf
mahi@mahendra:~/poky/raspberrypi/conf$ cd ..
mahi@mahendra:~/poky/raspberrypi$ bitbake core-image-minimal
Parsing recipes: 100% |########################################################################################| Time: 00:00:32
Parsing of 944 .bb files complete (0 cached, 944 parsed). 1347 targets, 66 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION        = "1.27.0"
BUILD_SYS         = "i686-linux"
NATIVELSBSTRING   = "Ubuntu-14.04"
TARGET_SYS        = "arm-poky-linux-gnueabi"
MACHINE           = "raspberrypi"
DISTRO            = "poky"
DISTRO_VERSION    = "1.8+snapshot-20150408"
TUNE_FEATURES     = "arm armv6 vfp arm1176jzfs callconvention-hard"
TARGET_FPU        = "vfp"
meta
meta-yocto
meta-yocto-bsp    = "master:82fea808eb3c70d5af5abc2b585da537fc1aec9f"
meta-raspberrypi  = "master:b896a7da70dd7a16ba7ffd664f7747cb37e1d142"
meta-multimedia   = "master:df6c7b1279790d27ebfd58fbdfbac89bde5782ec"

NOTE: Preparing RunQueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 1882 tasks of which 1624 didn't need to be rerun and all succeeded.
mahi@mahendra:~/poky/raspberrypi$
```

Fig 12: Success build after removing error

4.2.7 Board Support Packages (BSP)

It plays an important role in customization of Linux kernel build for specific target. In our case the target is Raspberry pi. A Board Support Package (BSP) is a set of programs which defines how to support a particular hardware device, example **piTFT (2.8)**, or hardware platform. Addition and Removal of packages as per requirement the yocto project provides flexibilities to build Linux kernel as per our requirement for target.In our case, the packages needed for us are

(i) Python

(ii) Pygame

(iii) Gcc

(iv) openssh /dropbear

(v) vi editor

(vi) wget

(vii) libsdl

(viii) package manager

(ix) Bash

(x) Tar

(xi) xorg

The required packages can be add to build the Linux kernel for Raspberry pi using hob. As the packages get added to the image the size of the Linux OS increases.This also can be added using terminal mode also. For that, we need to add the required packages name in the local.conf file present in conf directory of raspberry. The line which is to be add to the local.conf file inorder to get “nano” and “wget” are

IMAGE_INSTALL_append =”nano”, similarly for “wget”

IMAGE_INSTALL_append=”wget”

4.2.8 Linux Kernel Customization using hob

Hob is a graphical user interface for BitBake. Its primary goal is to enable a user to perform common tasks more easily. The steps followed in hob are

Step 1: Build images

Step 2: Edit existing image recipes

Step3: Create our own image recipes

Step 4: Deploy images to target device

Before running Hob, we need to make sure our environment set up. To do that we have to run the environment script.

$. oe-init-build-env

After the environment script finishes, we typed

$ hob

The Hob initial screen will appear as shown in figure 1. When Hob has launched, we should switch to "Settings" dialog and to make sure everything is set up for how we want to build as per our target. By default, the Hob "Settings" dialog provides reasonable values for the different variables.Hob is independent on BitBake. It does not affects the bblayers.conf file. Hob takes the update value from local.conf for configuring the parameters given below.

```
MACHINE =”raspberrypi”
PACKAGE_CLASSES= “package management”
DISTRO=”poky”
DL_DIR=”home/mahi/downloads’
PARALLEL_MAKE=”-j6” (for i3 processor)
GPU_MEM = “16”
```

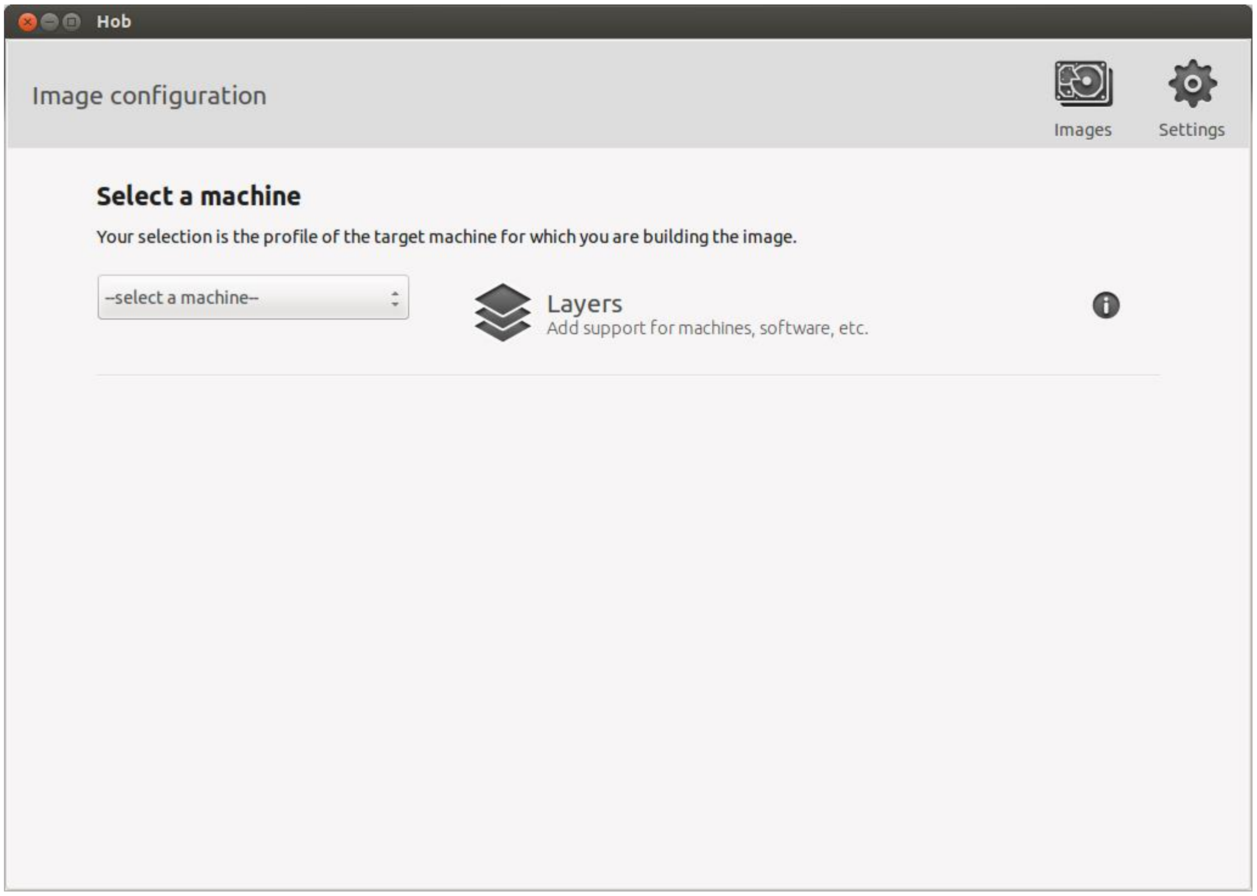

Hob
Image configuration
Images
Settings
Select a machine
Your selection is the profile of the target machine for which you are building the image.
--select a machine--
Layers
Add support for machines, software, etc.

Fig13: Hob main menu selection

4.2.9 Building images:

Once we are done with settings and saved, we can decide to take one of the profile for the target machine for which platform we are going to build the image. The Hob machine selection menu once we pick the target machine, recipes are parsed. After the recipes are parsed, we can set image-related options using the "Advanced configuration" dialog.

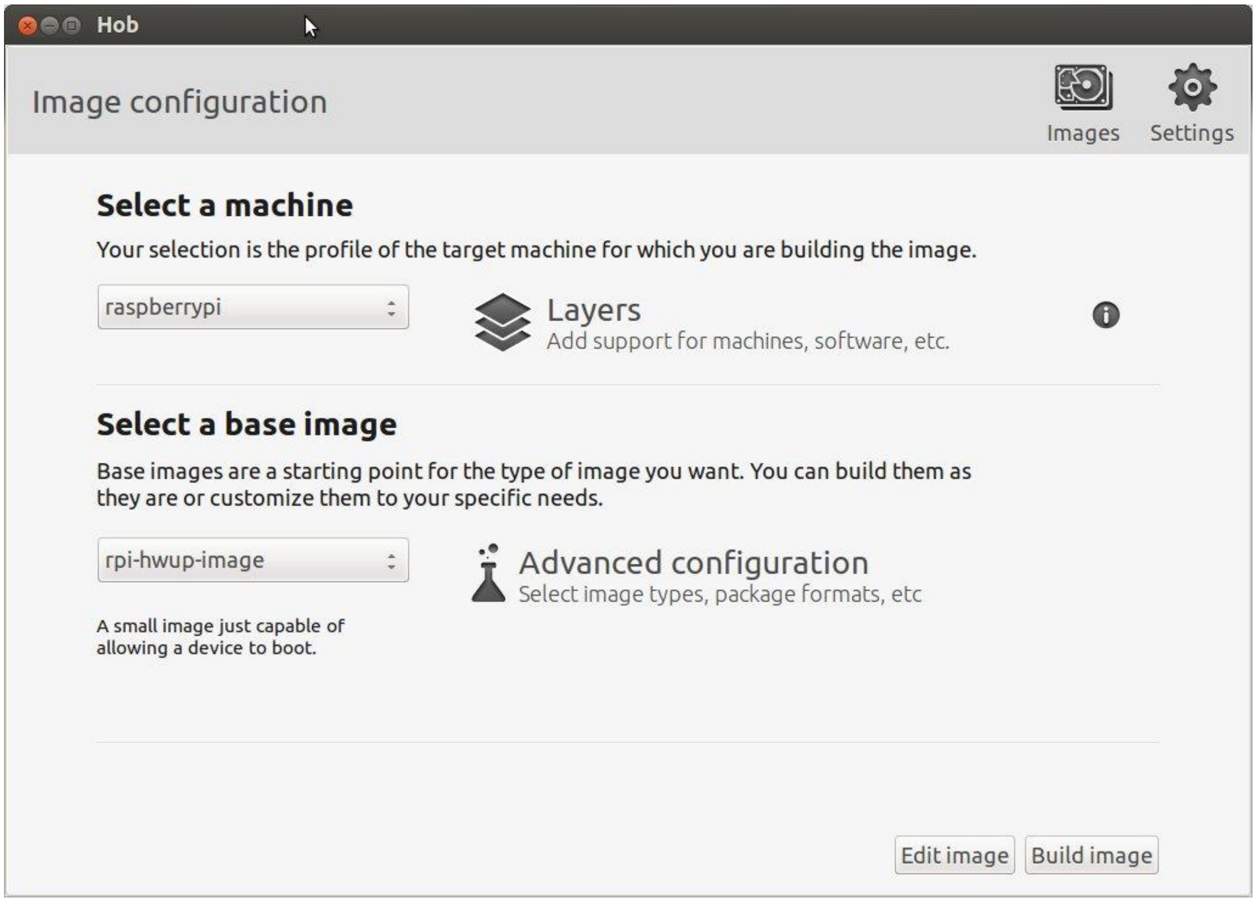

Hob

Image configuration

Select a machine

Your selection is the profile of the target machine for which you are building the image.

raspberrypi

Layers
Add support for machines, software, etc.

Select a base image

Base images are a starting point for the type of image you want. You can build them as they are or customize them to your specific needs.

rpi-hwup-image

A small image just capable of allowing a device to boot.

Advanced configuration
Select image types, package formats, etc

Fig 14: Selection of Hob target and basic image

Image types: The different distribution are used (e.g. poky, poky-lsb, etc.), and the types of root file systems we want to build or produce the Output: The package types, the size of our image and the option to populate an SDK for a certain host platform.

4.2.10 Creating own recipes

In case, we need to decide the type of image recipe we decided to build for our target. We can choose from many image recipes. Hob will also show a brief explanation of our selected image recipe as per our requirement. If we want to customize the recipes and packages included by default in our selected image recipe, we can "Edit image recipe" according to our requirement image. When we want to modify the kernel recipes, it is better to go for that which can be created and prepared as per our own layer do our work. The layer has BitBake append files such as (.bbappend).

4.2.11 Creating Layers

Layers allow us to differentiate different types of customizations from each other. The Source Directory has its own general layers for the hardware and Board Support Packages layers.It is not mandatory that a layer should be begin with the prefix as meta-, which is commonly recommended standard in the Yocto Project. Some of the widely used Layers are meta, “meta-skeleton”, “meta-selftest”, “meta-yocto”, and “meta-yocto-bsp”.

4.2.12 Creating Our Own Layer

It is better to create our own layers for the hardware in order to support the package dependency. We have to be careful before creating a new layer that someone should not has already created a layer which having containing the Metadata we need.

(i)Creation of Directory**:** Creation of a directory involve our layer only. It is not needed, use prefix string meta. Ex: “meta-mylayer”, “meta-GUI_xyz”and “meta-mymachine”

(ii)Creating a Configuration File: Inside our new layer directory, we need to make a file of conf/layer.conf f. It is quite helpful to take an existing layer configuration then modify the file as per our need. We have a conf and classes directory that can be added to BBPATH

BBPATH. = ":${ LAYERDIR}" The config file and the class folder are appended to the file named as BBPATH. The recipe work flow diagram shows in the figure 18. Which represents general working flow of the selecting recipes and adding the layers to it.

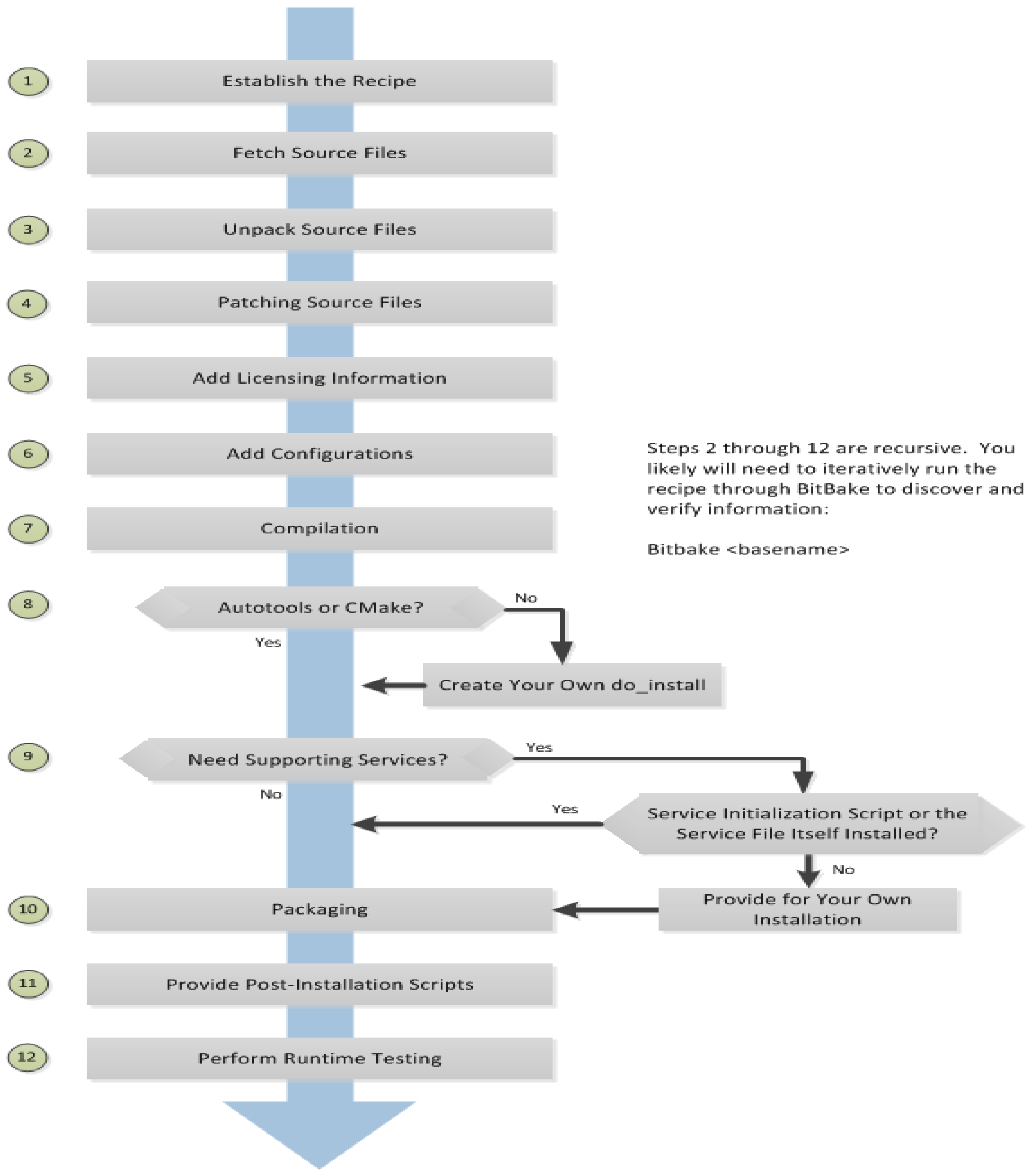
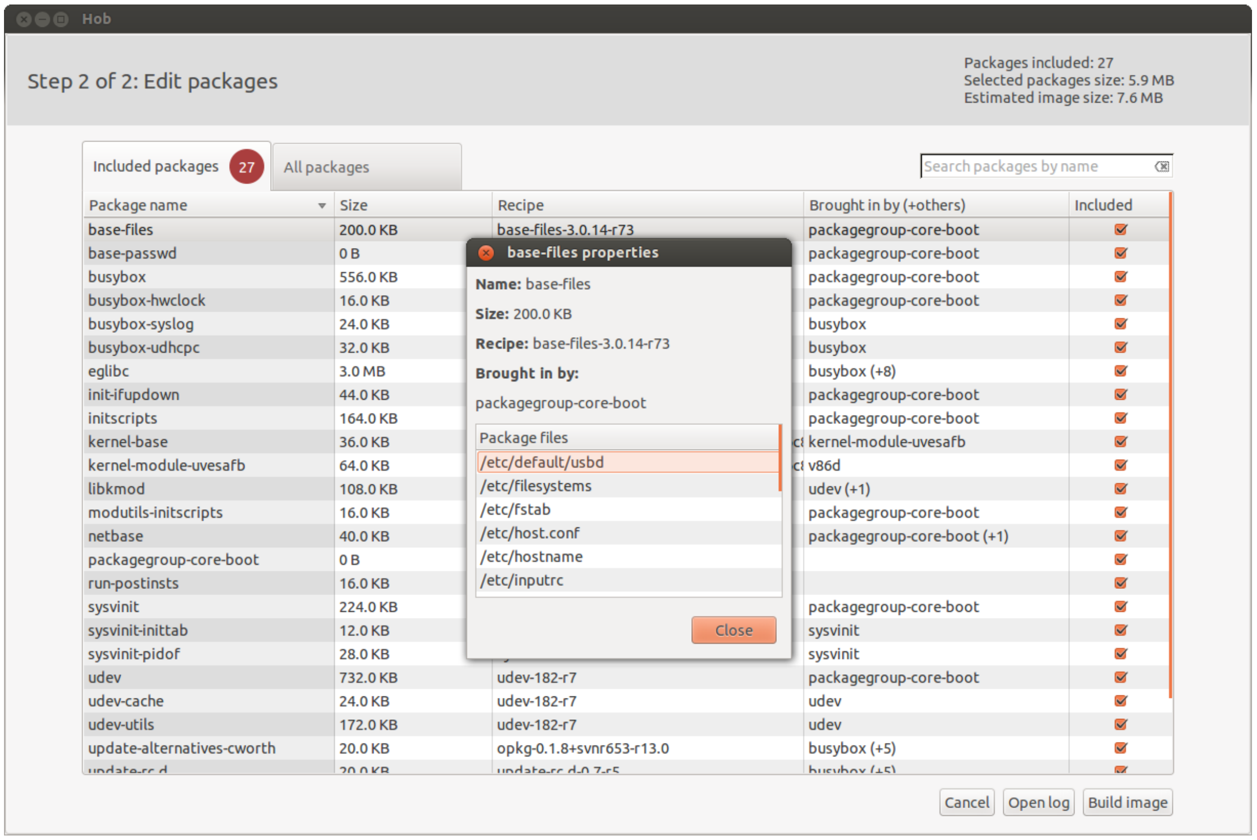

1
Establish the Recipe
2
Fetch Source Files
3
Unpack Source Files
4
Patching Source Files
5
Add Licensing Information
6
Add Configurations
7
Compilation
8
Autotools or CMake?
No
Yes
Create Your Own do_install
9
Need Supporting Services?
Yes
No
Yes
Service Initialization Script or the Service File Itself Installed?
No
Provide for Your Own Installation
10
Packaging
11
Provide Post-Installation Scripts
12
Perform Runtime Testing
Steps 2 through 12 are recursive. You likely will need to iteratively run the recipe through BitBake to discover and verify information:
Bitbake <basename>

Fig 15: Recipe work flow (source: http://www.yoctoproject.org/docs/1.6.1/kernel-dev/kernel-dev.html)

Note

As per our requirements are concerned we need Python, Pygame, libsdl, gcc (full version)

Wget, tar, nano, bash, openssh. As the packages get added to the recipes the size of the custom image increases.



Hob

Step 2 of 2: Edit packages

Packages included: 27
Selected packages size: 5.9 MB
Estimated image size: 7.6 MB

Included packages 27 | All packages

Search packages by name

Package name	Size	Recipe	Brought in by (+others)	Included
base-files	200.0 KB	base-files-3.0.14-r73	packagegroup-core-boot	✓
base-passwd	0 B		packagegroup-core-boot	✓
busybox	556.0 KB		packagegroup-core-boot	✓
busybox-hwclock	16.0 KB		packagegroup-core-boot	✓
busybox-syslog	24.0 KB		busybox	✓
busybox-udhcpc	32.0 KB		busybox	✓
eglibc	3.0 MB		busybox (+8)	✓
init-ifupdown	44.0 KB		packagegroup-core-boot	✓
initscripts	164.0 KB		packagegroup-core-boot	✓
kernel-base	36.0 KB		kernel-module-uvesafb	✓
kernel-module-uvesafb	64.0 KB		v86d	✓
libkmod	108.0 KB		udev (+1)	✓
modutils-initscripts	16.0 KB		packagegroup-core-boot	✓
netbase	40.0 KB		packagegroup-core-boot (+1)	✓
packagegroup-core-boot	0 B			✓
run-postinsts	16.0 KB			✓
sysvinit	224.0 KB		packagegroup-core-boot	✓
sysvinit-inittab	12.0 KB		sysvinit	✓
sysvinit-pidof	28.0 KB		sysvinit	✓
udev	732.0 KB	udev-182-r7	packagegroup-core-boot	✓
udev-cache	24.0 KB	udev-182-r7	udev	✓
udev-utils	172.0 KB	udev-182-r7	udev	✓
update-alternatives-cworth	20.0 KB	opkg-0.1.8+svnr653-r13.0	busybox (+5)	✓
update-rc.d	20.0 KB	update-rc.d-0.7-r5	busybox (+5)	✓

base-files properties

Name: base-files

Size: 200.0 KB

Recipe: base-files-3.0.14-r73

Brought in by:

packagegroup-core-boot

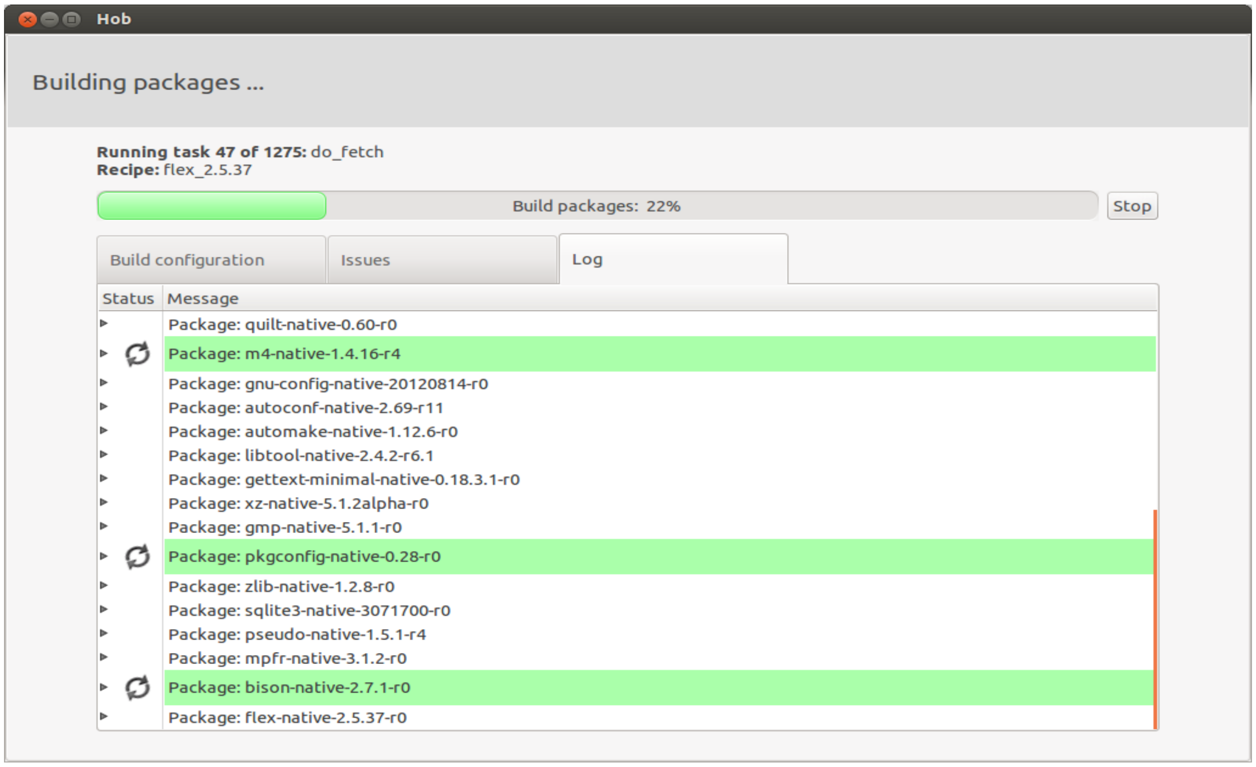
Package files
/etc/default/usbd
/etc/filesystems
/etc/fstab
/etc/host.conf
/etc/hostname
/etc/inputrc

Close

Cancel | Open log | Build image

Fig 16: Hob default packages for image

Once we are satisfied with the set of recipes, we can proceed by clicking the "Build packages" button. Or, we can cancel our changes and come to the main menu by click on the "Cancel" button. Here the figure shows is how the screen looks like during package building, when the packages are built, the image can be further modified by selecting what are the packages actually we want to include asper our requirement.



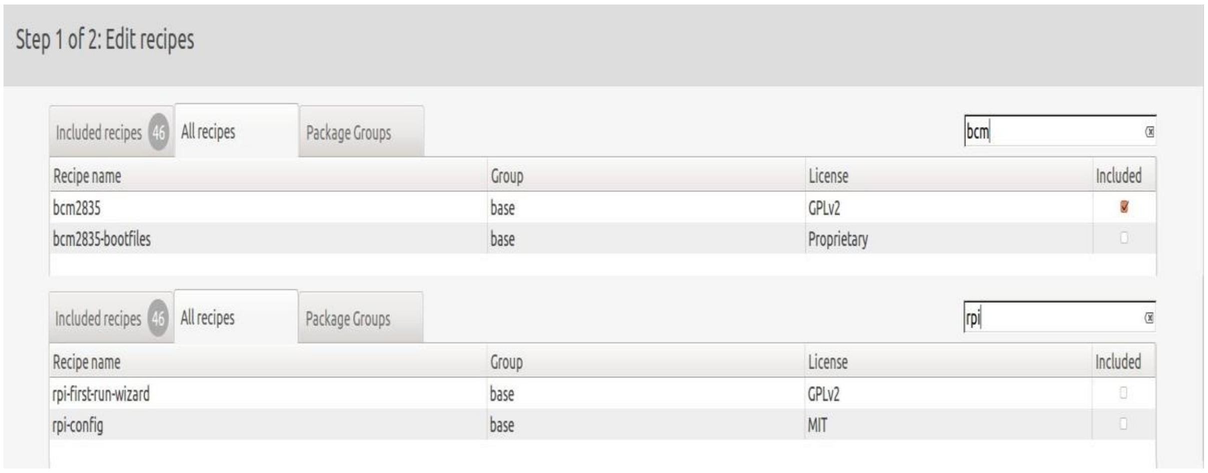
Running task 47 of 1275: do_fetch
Recipe: flex_2.5.37

Build packages: 22%

Status	Message
▸	Package: quilt-native-0.60-r0
▸	Package: m4-native-1.4.16-r4
▸	Package: gnu-config-native-20120814-r0
▸	Package: autoconf-native-2.69-r11
▸	Package: automake-native-1.12.6-r0
▸	Package: libtool-native-2.4.2-r6.1
▸	Package: gettext-minimal-native-0.18.3.1-r0
▸	Package: xz-native-5.1.2alpha-r0
▸	Package: gmp-native-5.1.1-r0
▸	Package: pkgconfig-native-0.28-r0
▸	Package: zlib-native-1.2.8-r0
▸	Package: sqlite3-native-3071700-r0
▸	Package: pseudo-native-1.5.1-r4
▸	Package: mpfr-native-3.1.2-r0
▸	Package: bison-native-2.7.1-r0
▸	Package: flex-native-2.5.37-r0

Figure 17: Hob while building packages

By Click on a package we can see some additional details about it. The information includes the list of files that the package will install into the root file system. Once you are satisfied with the set of packages. The packages can be further selected after building; if the respective packages are not available we can rebuild the image.



Recipe name	Group	License	Included
bcm2835	base	GPLv2	✓
bcm2835-bootfiles	base	Proprietary	☐

Recipe name	Group	License	Included
rpi-first-run-wizard	base	GPLv2	☐
rpi-config	base	MIT	☐

Figure 18: Editing recipes

The building of images takes time to compile depending on the native processor and internet speed. Approximately it take 10 to 12 hours for i5 processors. Our customized image is available in tmp/deploy/images/ directory having the name hob-image-raspberrypi.rpi-sdimg. Now it is ready to be flush in to SD-card. When the packages are not compatible according to our configuration the build stops automatically. So we have to be careful about the packages before we start building. Once build is done, we can flash our SD card with the custom image and insert it to Raspberry pi SD card socket. After giving the power supply we access the Yocto based Embedded LINUX using host terminal through **SSH,** acting as **root and no password**.

```
$ssh root@10.42.0.33
```

Where 10.42.0.33 is the IP address generated by nmap.

4.3 Hardware Connections:

The Raspberry Pi is interfaced with piTFT using 26 GPIO Pins. The GSM Modem SIM 900 A interfaced with Raspberry pi through serial communication using Tx and Rx pin. Tx, Rx pins of Raspberry pi have been connected to the Rx and Tx pins of GSM modem respectively having common ground. The audio input and output are interfaced to mini metal microphone and mini speaker from GSM modem through dedicated pins.

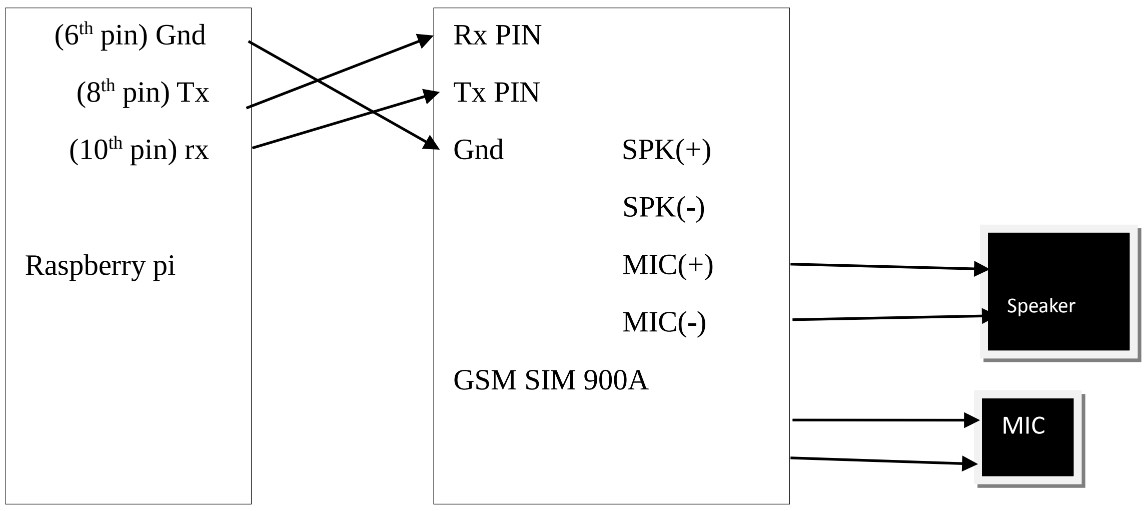


Fig 19: Hardware connections

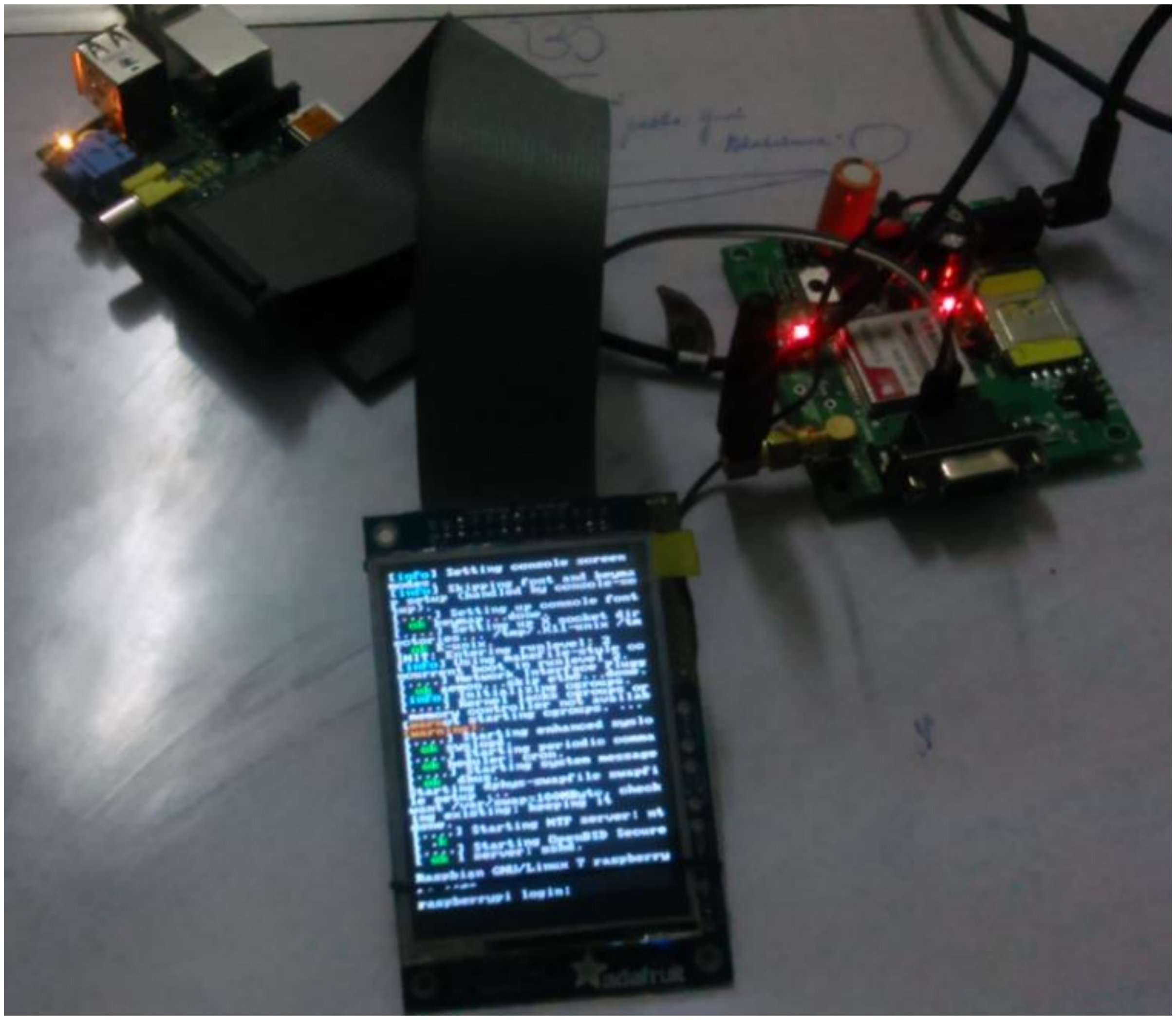

```
[info] Setting console screen
[illegible]
[info] Setting up console font
[illegible]
Starting enhanced syslo
[illegible]
Starting NTP server: nt
Starting OpenBSD Secure
[illegible]
Raspbian GNU/Linux 7 raspberry
raspberrypi login:
```

Fig 20: Physical Hardware interconnections

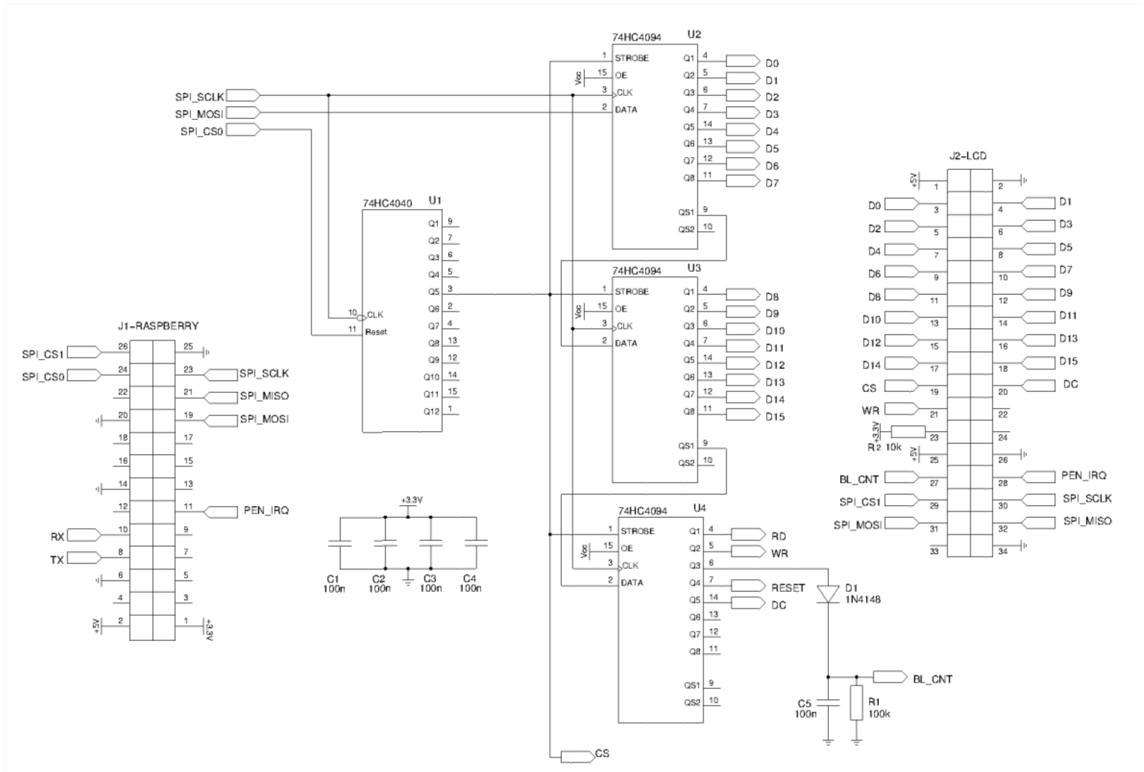
4.4 SPI in Raspberry pi

It is quite simple and easy to use serial protocols in embedded system. Less number of pins are used and reduce complex functionality. Focusing on the Serial Peripheral Interface (SPI), this paper explains the reasons to test and debug a SPI port

Serial Parallel interface can save the use of GPIO pins on raspberry pi. SPI uses five pins of Raspberry pi MISO (pin 09), MOSI (pin 10, CLK (pin 11), CE 0(pin 8), CE 1(pin 7).

The advantages, we will get

- ❖ Most of the GPIO pins will be free for other peripherals.
- ❖ The data transfer rate also increase in case of SPI compared to serial communication with the peripheral devices.
- ❖ Easy handling of data
- ❖ No limitations of 8 bit data
- ❖ Push – pull drivers give very good result with high speed



,

Fig 21: Serial to parallel conversion circuit

4.4.1 Serial to Parallel Interface:

The Devices able to communicate in master/slave mode when the master device initiates with the data frame. Many number of slave devices are allowed with individual slave selector lines. SPI is also called as four-wire serial bus, depending on the three-, two-, and one-wire serial buses. SPI is referred as SSI (Synchronous Serial Interface)

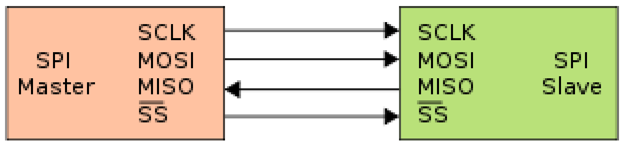


Fig 22: SPI bus with Single master single slave

The SPI bus has four logic signals are given below:

SCLK: Serial Clock (output of the master).

MOSI: Master Output, Slave Input (output from master to slave).

MISO: Master Input, Slave Output (slave to master).

NOTE We can interface Raspberry pi with ITEAD LCD b02 having 40 input/output pins .In this case we need SPI, except that we have used PI TFT 2.8 having 26 GPIO pins.

Fig 23: ITEAD B02 3.2" LCD

CHAPTER 5

RESULT AND DISCUSSION

After getting all the hardware components, we have interfaced pi TFT with Raspberrypi (Raspbain OS). With Yocto based Embedded LINUX, interfacing of piTFT could not be done yet (calling takes place via host terminal) since the piTFT driver is kept in staging mode by Yocto people and would be involved in meta-raspberrypi only in Linux 3.20 kernel and currently we are running Linux 3.18.11 kernel. In case, we want to use ITEAD B02 3.2 inch TFT, we need to implement SPI circuit since it has 40 GPIO pins and Raspberrypi have only 26 pins.

5.1 Interfacing piTFT with Raspberry pi:

Raspberry pi can be interfaced with piTFT 2.8, since it has 26 GPIO pins. The GPIO pins are connected through 26 pin serial Bus as shown in figure. Then the Raspbian OS starts booting and after login, the graphical x window displayed. The SD card contains the piTFT driver with Raspbian.

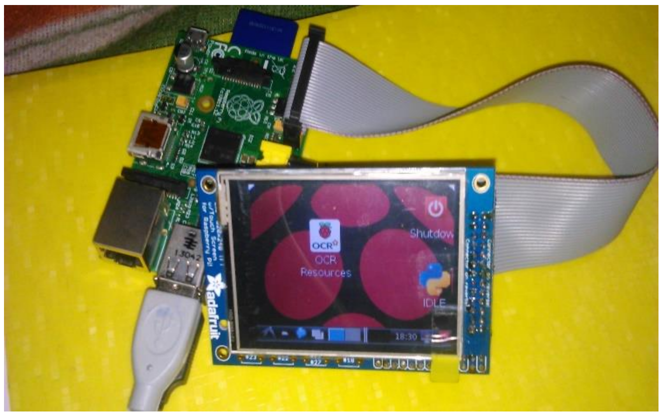


Fig 24: Pi TFT 2.8” interface with Raspberry pi from Adafruit

❖ We have used python for making GUI (Graphical User Interface) for number dialer.

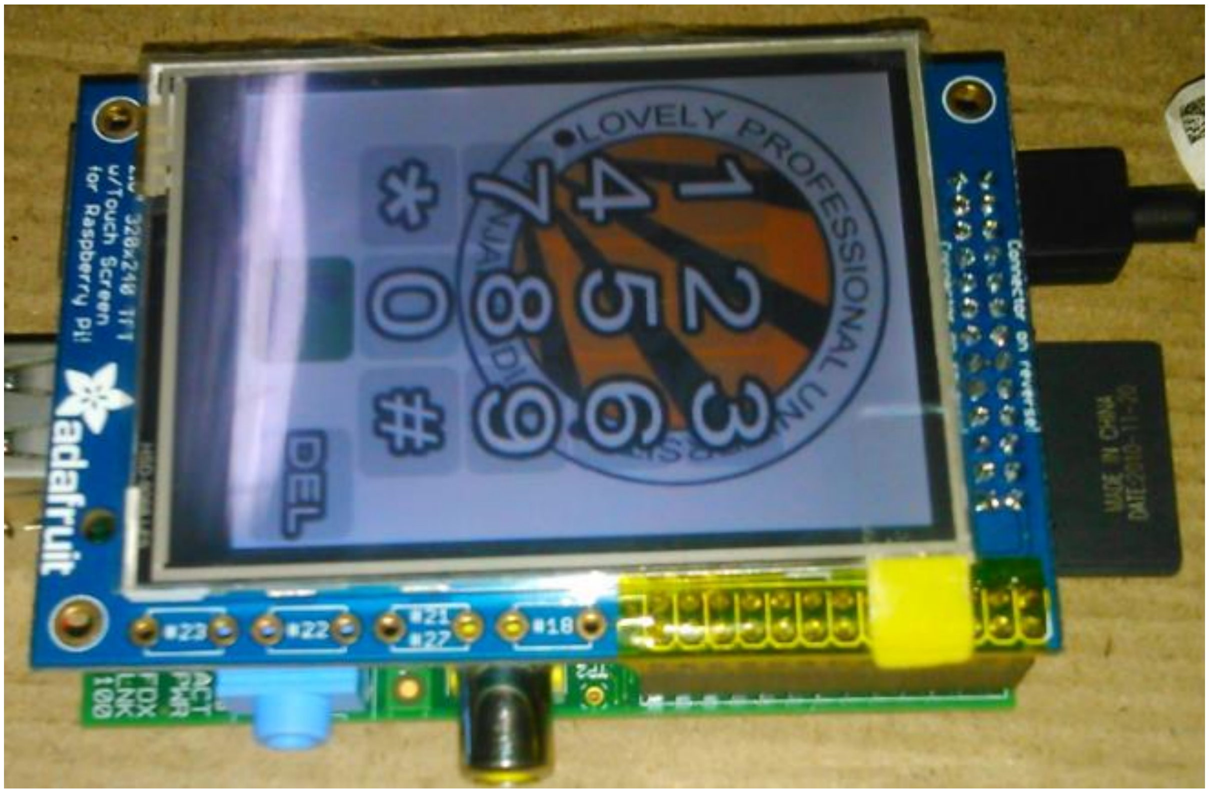


Fig 25: Python based number dialer on Raspberry pi

- ❖ Successful voice transmission between sending and receiving end with optimum voice clarity.
- ❖ Successful customization of Linux OS from 3.3 GB to 300 MB (without GUI & Python)
- ❖ The piTFT (2.8”) has been calibrated to perform touch sensitive and we have used this feature to write text, draw graphics
- ❖ Successful Yocto image build for raspberry Pi Basic image of 13 MB, 330MB(without GUI and python) and 650 MB(GUI + Python)

The number dialing and making call in device are shown in figures with LPU logo. The audio interface with GSM modem gives clear voice clarity at transmitter as well as receiver end.

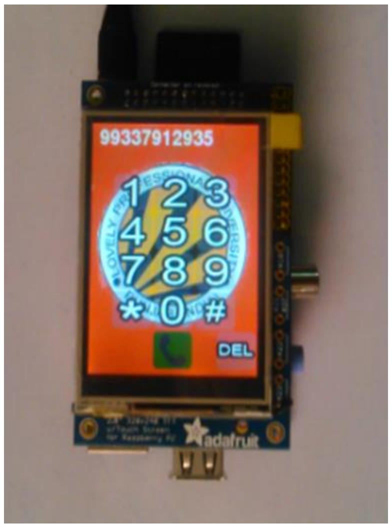
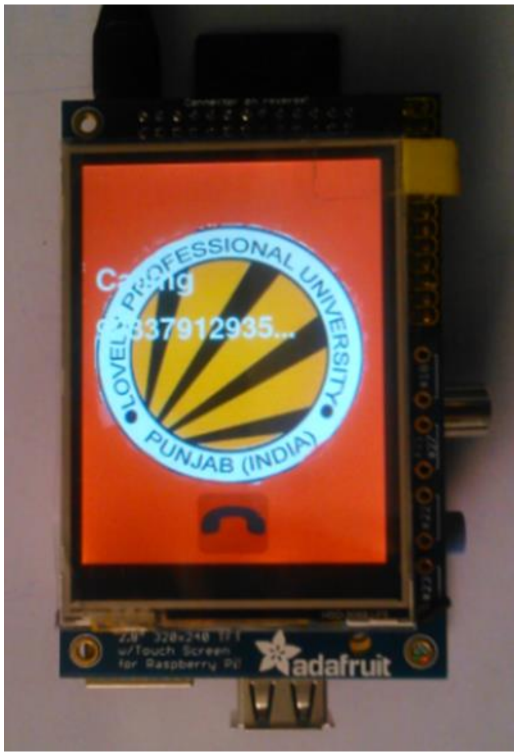


Fig 26: Number dialer pad (using python GUI)

The Python graphical user interface have been designed for call dialer pad which is touch sensitive. The graphical buttons designed are simple and responsive. There are two buttons on the GUI one is dedicated to dialing CALL and another one is for DEL button which is used to rectify as per the numbers of character dialed. The wallpaper decided for the device is having LPU logo. When the call button get pressed the python code starts to communicate with GSM SIM 900A and call get forwarded. The hanging button allows dropping the call after conversations get over. The custom images created by Yocto having some compatibility issue with piTFT 2.8" depending the Linux Kernel architecture armel and armhf. The Yocto image is having all the BSPs required to run the python GUI code. The result on the Yocto image would be similar to this.



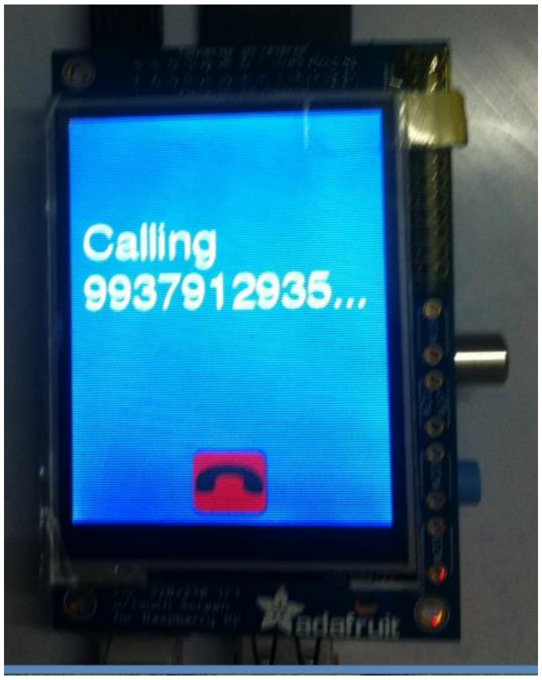


Fig 27: Device During voice call

CHALLENGES FACED

Here are the some challenges faced while developing this prototype:

- Interfacing of ITEAD LCD 3.2 (ITB02) with Raspberry pi using SPI communication protocol.
- Customization of LINUX Kernel with piTFT driver (for piTFT 2.8 from adafruit)
- Packages dependencies on ARM architecture "armel" and "armhf" in Raspberry pi.

CHAPTER 6

CONCLUSION & FUTURE SCOPE

Embedded Linux based voice calling device will produce a voice call alternative to mobile devices with its stationary appearance. Due to its stationary format, it will not suffer with battery drainage issue and may act as secondary voice calling device to any home, at time of emergency. The device has been customized at kernel and package level in order to get less memory foot print and application specific so that it can support more applications as required in future. The successfully voice transmission over the transmitter and receiver end has verified with optimum clarity which was our main objective. Due to limitation of funds, we have restricted to voice calling feature only.

FUTURE SCOPE

This device features are not restricted to voice calling only, some extra features can be added to enhanced performance of this device. We may also connect it to battery to make add portability to device. Since it supports Embedded LINUX, it could be loaded with other useful applications like web browser, games, music player, image processing application, etc. Other possible areas of uses of such devices could be home security surveillance, multifunctional device, and many more. This device features can be enhanced for better user experience. Ex: it can be used for Speech Recognition, WI pi server, video surveillance, robotics and automation and many more. To make it smart it can be connected to Wi-Fi dongle so that internet can be accessed on this device.

CHAPTER 7

REFERENCES

[1] Building Linux Kernel for Raspberry Pi Hannu Flinck, Nokia Solutions and Networks hannu.flinck@saunalahti.fi

[2] Using Phase Behavior in Scientific Application to Guide Linux Operating System Customization, Chandra Krintz Rich Wolski Computer Science Department University of California, Santa Barbara, IEEE computer society, 2011

[3] Porting the Linux Kernel to Arm System-On-Chip and Implementation of RFID Based Security System Using ARM, International Journal of Advanced Research in Computer Science and Software Engineering

[4] Studying Main Differences between Android & Linux Operating Systems, International Journal of Electrical & Computer Sciences IJECS-IJENS Vol:12 No:05. October 2012 [TA6] Android ology I: Architecture
http://www.android.com/about/videos.html#video=androidologyiarchitecture

[5] Porting the Linux Kernel to a New ARM Platform, Wookey and Tak-Shing, Aleph One • www.aleph1.co.uk

[6] Learn the INS and OUTs of Linux The operating System that runs Raspberry pi

By peter Membrey and David Hows

[7] Smart Projectors using Remote Controlled Raspberry Pi International Journal of Computer Applications (0975 – 8887)
(http://research.ijcaonline.org/volume82/number16/pxc3892250.pdf

[8]http://www.engineersgarage.com/embedded/raspberry-pi/controlling-hardware-using-gui-in-raspberry-pi

[9] https://learn.adafruit.com/ raspberry pi projects

[9] http://ozzmaker.com/tag/raspberry-pi-2/

[10] http://cagewebdev.com/index.php/category/raspberry_pi/

[11] http://www.raspberrypi.org/

[12]http://blog.iteadstudio.com/raspberry-pi-sim900-gsmgprs-module-adapter-kit/

[13] https://learn.adafruit.com/fona-tethering-to-raspberry-pi-or-beaglebone-black?view=all

[15]http://www.raspians.com/Knowledgebase/setting-up-a-remote-desktop-view-the-pi-on-your-windows-pc/

[16] http://www.shirwahersi.com/content/how-use-minicom-linux-serial-port-comunication

[17]https://help.ubuntu.com/community/Minicom

[18]http://engineering-with-a-rpi.blogspot.in/search/label/raspberrypi

[19]https://communities.intel.com/message/246077#24607

[20]http://www.tutorialspoint.com/python/tk_button.htm

[21]http://python-gtk-3-tutorial.readthedocs.org/en/latest/spinner.html

[22]http://www.icttm-iitd.in/paper-submission.html

[23]http://www.yoctoproject.org/docs/1.7.1/dev-manual/dev-manual.html#new-recipe-writing-a-new-recipe

[24]http://www.raspberry-projects.com/pi/programming-in-python/gui-programming-in-python/tkinter/tkinter-general

[25]http://inventwithpython.com/blog/2012/10/30/creating-a-button-ui-module-for-pygame/

[26]http://www.cnx-software.com/2012/07/31/84-mb-minimal-raspbian-armhf-image-for-raspberry-pi/

[27]http://www.linuxsystems.it/raspbian-wheezy-armhf-raspberry-pi-minimal-image/

[28]https://community.freescale.com/docs/DOC-94849

[29]http://www.python-course.eu/tkinter_buttons.php

[30]http://python-gtk-3-tutorial.readthedocs.org/en/latest/iconview.html

[31]https://www.raspberrypi.org/learning/python-for-vb-programmers/worksheet/

[32]http://www.averagemanvsraspberrypi.com/2014/07/how-to-set-up-adafruit-pitft-for.html

[33]http://www.cnx-software.com/tag/debian/page/9/