AUTOMATEING THE CRYOPROB BY RASPBERRY PI(RPi)

Major Project

Submitted in partial fulfillment of the requirements for the degree of

Master of Technology

in

Electronics & Communication Engineering
(Embedded Systems)

By

YOGESH BADOLE
(12MECE29)

Electronics & Communication Engineering Branch

Electrical Engineering Department

Institute of Technology

Nirma University

Ahmedabad-382 481

May 2014

AUTOMATEING THE CRYOPROB BY RASPBERRY PI(RPi)

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

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

By

YOGESH BADOLE
(12MECE29)

Under the guidance of

External Project Guide:

Mr. S.G Trivedi
Managing Director
Maharshi Electronic And Systems,
Ahmedabad.

Internal Project Guide:

Dr. Yogesh N. Trivedi
Associate Professor, EC Dept,
Institute of Technology,
Nirma University, Ahmedabad.

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

May 2014

Declaration

This is to certify that

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

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

YOGESH BADOLE

Certificate

This is to certify that the Major Project entitled "*Automating the Cryoprobe by Raspberry Pi (RPi)*"submitted by *Yogesh Badole* (*12MECE29*), towards the partial fulfillment of the requirements for the degree of Master of Technology in Electronics and Communication Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

_____	_____
Internal Guide	Dr. N.P Gajjar
Dr. Y.N Trivedi	PG Co-ordinator (Embedded Systems)
_____	_____
Dr. P.N Tekwani	Dr K Kotecha
rofessor & Head, EE	Director, IT-NU

Date: _____ **Place: Ahmedabad**

Acknowledgements

With immense pleasure, I would like to present this report on the dissertation work related to "Automating Cryoprob By RPi". I am very thankful to all those who helped me for the successful completion of the first phase of the dissertation and for providing valuable guidance throughout the project work.

I would first of all like to offer thanks to **Mr. S. G. Trivedi** , Guide & Managing Director Of Maharshi Electronic Systems and **Mr. Jay Shah** Guide & "Project Engineer ", whose keen interest and excellent knowledge base helped me to finalize the topic of the dissertation work. His constant support and interest in the subject equipped me with a great understanding of different aspects of the required architecture for the project work. He has shown keen interest in this dissertation work right from beginning and has been a great motivating factor in outlining the flow of my work.

My sincere thanks and gratitude to **Prof. Y.N. Trivedi**, Internal Guide and Associate Professor,Institute of Technology, Nirma University, Ahmedabad for his continual kind words of encouragement and motivation throughout the Dissertation work.

I am thankful to Nirma University for providing all kind of required resources. I would like to thank The Almighty, my family, especially my mother and my brother, for supporting and encouraging me in all possible ways. I would also like to thank all my friends who have directly or indirectly helped in making this dissertation work successful.

- YOGESH BADOLE

12MECE29

Abstract

Cryoprobes are probes of Cryogenic probe station. Cryogenic probe stations are high performance instruments, which are used for creating environment for testing devices and semiconductor wafers.

Cryogenic probe station can create affordable vacuum , temperature and cryogenic probing of wafer and devices. To reduce human efforts and prevent human error, it is required to automate this instrument . An embedded system is necessary for automating this instrument, which can facilities flexibility to analyze the working and performance of devices and semiconductor wafers.

To full fill that purpose, it requires an automation and control system.To achieve this objective, we use Raspberry Pi(RPi) Board provided by Raspberry pi foundation. RPi is based on "SoC"(System On Chip)concept. It is a small size CPU, able to provide interactive services to the user. RPi used Broadcom BCM2835 chip. BCM2835 contains ARM1176JZFS and Graphical processor unite(GPU). RPi is very small in size, it's cost is low and it provides high performance. RPi also support different operating systems. Most efficient operating system for RPi is Raspbian "wheezy".Rasbian is a customized version of the Debian OS for RPi.

Combination of Raspbian OS and RPi makes it possible to use python language to create interactive applications as per benchmarking of RPi GPIO. According to benchmarking one can achieve 44 KHz of frequency which is enough for automation and Python language provides Application Peripheral Interface(API) to control GPIO and create Graphical User Interface(GUI).

These interactive applications provides the user an easy control on Croprobes. Cryoprobes require a high resolution motion control to keep on tracing all the motion

of every probe and keep that data for future analysis. And also one can access it from remote distance. In this hardware control system we also require temperature measurement system for every probe to create graph between time and temperature.

This application also provide service to store data which occurs during practical, for future analysis. In automation, through stepper motor with 1.8 degree of step angle, 98% accuracy is achieved.

To achieve remote access shell script is used. Performing text processing and file operation remote access task is achieve.

Abbreviation Notation and Nomenclature

ASIC ..... Application Specific Integrated Chip
CV ..... Capacitance and voltage
CSI ..... Camera Serial Interface
DMA ..... Direct Memory Access
DSP ..... Digital Signal Processor
FPGA ..... Field Programmable Gate Array
GPIO ..... General Purpose Input Output
HDMI ..... High Definition Multimedia Interface
IV ..... Current And Voltage
LAN ..... Local Area Network
LED ..... Light Emitting Diode
OS ..... Operating System
PSoC ..... Programmable System On Chip
PWM ..... Pulse Width Modulator
RPi ..... Raspberry Pi
RTOS ..... Real Time Operation System
SoC ..... System On Chip
SPI ..... Serial Peripheral Interface
TTM ..... Time To Market
USB ..... Universal Serial Bus

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Cryogenic Probe Station

The Cryogenic probe stations are high performance research instruments designed to provide affordable vacuum and cryogenic probing of wafers and devices.In this thesis we using ST-500 series probe station.[6] The proven ST-500 cryostat is the platform for these probe stations, and includes low vibration technology (originally designed for high spatial resolution optical microscopy) to provide outstanding sample positional stability. Researchers around the world are using these systems to conduct research in a wide variety of fields, including MEMS, nanoscale electronics, superconductivity, ferroelectrics, material sciences, and optics.

Fig1.1 shows the cryoprobe in the cryogenic probe station. Probe station has four probe, those are able to motion in the X-axis, Y-Axis and Z-Axis.[6]

1.2 Features and Capabilities

The ST-500 probing station has the following features and capabilities:

- Very smooth X-Y-Z- travel stages for all monoscope system assemblies

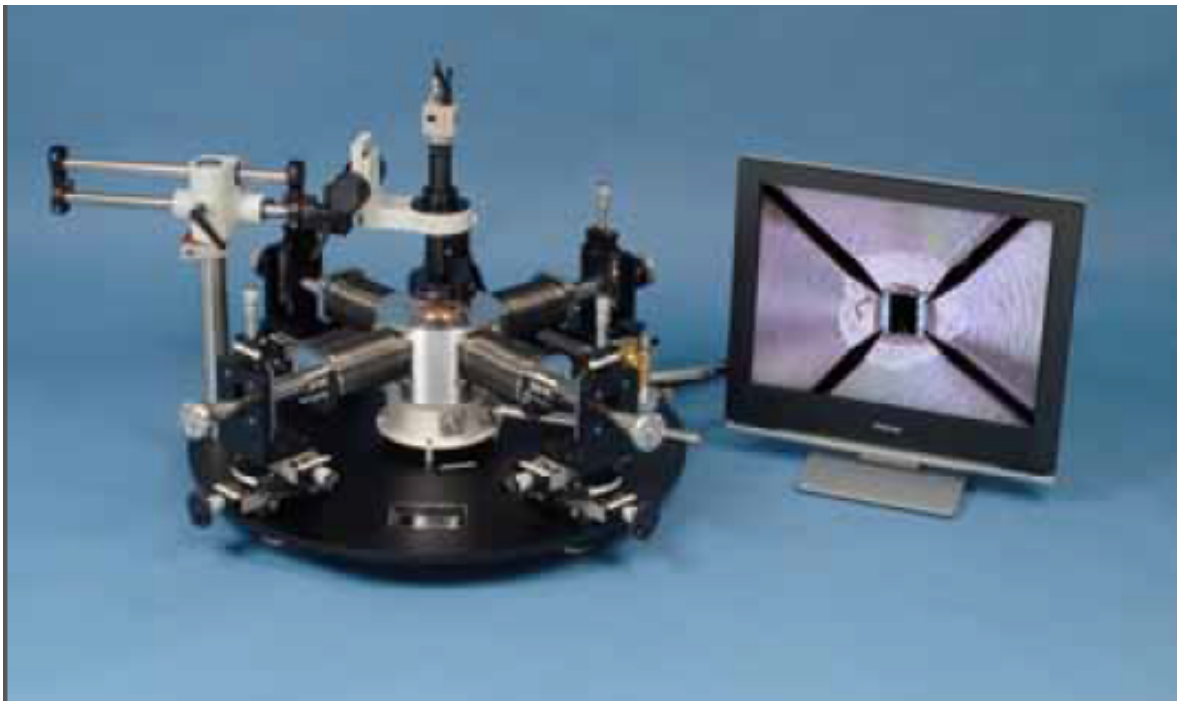

Figure 1.1: Cryogenic Probe Station
[6]

- Low vibration level and the positional drift
- Accommodates up to 2" (51 mm) diameter wafers (optional: up to 8" [203 mm])
- Temperature range from 3.5 K to 475 K (optional: 8 K to 675 K)
- Works either with liquid nitrogen or liquid helium
- Helium consumption less than 1 liter/hour
- Up to seven cooled, easily interchangeable, micro-manipulated probe arms
- Very low triaxial probe arms leakage current of just a few fA
- Multi-tip probes
- Optional optical access through the sample mount for transmission measurements

- Additional electrical feedthroughs with cables and wires to sample area
- System customization options

1.3 Probe Stander Specification

Table 1.1: Probe Station Specification [6]

Vibration level:	25 nm
Positional drift:	150 nm in 30 minutes
emperature range:	3.5 K to 475 K (8 K to 675 K optional)
Cryogenic consumption:	Helium: less than 1 liter/hr; Nitrogen: less than 0.1 liter/hr
Temperature stability:	50 mK
Cooling time (for standard 2" diameter sample mount):	30 min to 10 K, 60 min to 5 K
Warming up time:	45 min with quick warming up option (4 hours without)
Sample mounts: 2"(51 mm) diameter standard up to 8"(203 mm) optional	Ground Chuck Electrically Isolated Chuck with Bias Voltage Coaxial Cable Special Chuck for Light Transmission Experiments Leadless Chip Carrier (LCC) Holder

Interchangeable Probe Arms:	DC/LF probes: DC to 20 MHz with following tips available: - Tungsten tips with 0.1 to 200 micron tip radius (optional gold plating) - Special Tungsten bendable shank and tip (cat whisker) tips - Beryllium Copper soft tip with low contact resistance Coaxial or triaxial (a few fAmps leakage currernt) wiring Microwave probes: 0-40 GHz 0-50 GHz 0-67 GHz Fiber probes: Single Mode UV-VIS or VIS-IR Multimode Non-contact, non-destructive Kelvin probes Multi-tip probes
Monoscopes with LCD 19" monitor or USB camera and smooth travel stages	7.1 zoom, 5 microns resolution(216X magnification) 12.5:1 zoom, 3.4 microns resolution (508X magnification) 16.1 zoom 2.2 microns (626X magnification)
Probe travel:	**ST-500-1**
X-axis:	1" (25 mm)
Y-axis:	1" (25 mm) (15 mm with MW Probes)
Z-axis:	10 mm (18 mm optional)
X-, Y-, Z-axes probe translation (incremental units of graduation):	10 microns

1.4 Miro-Manipulation of Cryogenic Probe Systems

1.4.1 Micromanipulated Translation Stages

In this cryogenic probe translation instrument provide eight stage of X,Y,Z- Axis precise control over the motion of the probe.In every stage include gradual increment of 10-12.5 microns, with typical useful resolution of 5 to 6.25 micron.Stage used for microwave probes include theta rotation adjustment for polarization of the probe.

1.4.2 Probe Option

Cryoprobe Instrument can be used for wide variety of different applications so that variety of probe option available for the researcher. This option can categorize as test for low frequency, high frequency, microwave frequency experiment and test with fiber optics probe.Typical configuration include:

Low Frequency(DC) Probe

Tip diameter and radios on probe, material used in experiment can be specified by the user according to the requirement of the experiment.

- CX: Coaxially shielded low frequency probe
- TX: Triaxially shielded low frequency probe
- KEL: Kelvin probe it is used for low resistance measurement in order to eliminate cable resistance from the probe

High Frequency Probe

- MW: Microwave probe for high frequency measurement.MW probe are available in 40, 50 and 60 GHz range for performing experiment on different wafer of devices.

1.5 Application Of Cryogenic Probe Station

- Typical applications include sampling IV and CV curves over a wide range of temperatures measuring
- Microwave and electro-optical responses characterizing
- Magneto-transport properties in variable magnetic fields
- Hall-effect measurements to understand carrier mobility
- Variety of material studies

1.6 Motivation

To automate an mechanical device and make an efficient embedded system. Cryogenic probe station is very delicate instrument.Many researcher and scientist use these type of instruments to find out the IV and CV characteristics of a particular wafer and provide the probing system The Cryoprobe in that instrument is controlled manually which is very sharp work and take more time ,energy and attention of a researcher. Due to the limitation in the human attributes for long time work. We have to create an embedded system which give more flexibility, easiness and comfort in work. So that researcher pay more attention on their actual work, observation and analyses. To achieve all that goals we have to achieve some small goals and objective.
Those objectives are:

First, a low cost, high performance and high degree of motion control system through the RPi board and stepper motor. An GUI application which provide more interaction with the user.

Second, Development of Temperature tracer which can show temperature of each bar and GUI shows the graph between time and temperature.

Thirdly, make that system Real time and limit the processes execute in it. Prepare a library for GPIO access and manage more number of devices control by board. Prepare an wireless system by board which can control the device at distance.

.

1.7 Objective

Apart from the above mentioned goals and vision, this report addresses the specific objective . Hence, the objective of the dissertation work can be summarized as:

Study Cryogenic Probe station and find out the specification related to motion of probes in X and Y direction.What is the resolution of translation? How to achieve high much resolution? Study the feature and the specification given for cryoprobe.

Study and selection of the hardware on the basis of specific criteria and provide more facility and services. Criteria based on cost, performance, size and availability. RPi meet all this criteria regarding the project.Find out limitation and services of the board. Ports available on board. Find out video, audio and different hardware connections on board.

The next objective is, study, selection and porting of Open source OS on it. OS which give the efficient performance with the RPi. OS which is customize according to the RPi. Raspbian "wheezy" OS meet all the criteria. Explore the application, services and facilities OS support.Find out the limitation of OS. Explore different application like editor support, language support etc.

The next objective is, study about the GPIO access through board. Which type of Buses supported by RPi. Can we use GPIO as a bus interface terminal? How to

access GPIO for control external hardware like: DC motors, Stepper Motors? Find out extra circuitry required for external hardware(motor) control? Can we control GPIO through GUI application? Which language support GPIO control through interactive GUI application ? How to manage more number of hardware if GPIO pins are less? How to make library for GPIO access in python or C language?

The next objective is, to test the performance of the motion system and temperature tracer. Find out can we use wireless connection between the hardware and the RPi Board?

Finally, the objective is to build the high resolution, low cost, easily available, easily expendable customize and interactive motion control system and temperature tracer.

1.8 Scope of Work

In the experimental setup and to full-filling these objectives of the work, selection of the hardware and software depend on some parameter. Hardware selection on the basis of some parameter like cost, size, performance and availability of the hardware. RPi board for the the hardware support which meet all the criteria. This board is also support the Linux OS "Raspbian" which is a open source. Combination of Both the hardware and OS provide you huge flexibility. RPi is small size CPU which can control the four stepper motors and also trace the temperature.To Automate the Cryoprobe, lowest step angle stepper motor is required which give the high resolution motor control and a temperature sensor.An interactive GUI motion and temperature control system is the part of the thesis. RPi and Raspbian OS also become part of the thesis. Thesis include detailed information about GPIO access of the board and study of the library for GPIO access and control it with the GUI.

Chapter 2

Literature Survey

2.1 Embedded System

An embedded system is a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a specific function. It is important to point out that a general-purpose computer is itself made up of numerous embedded systems. For example, my computer consists of a keyboard, mouse, video card, modem, hard drive, floppy drive, and sound card-each of which is an embedded system.Each of these devices contains a processor and software and is designed to perform a specific function.

2.2 Consideration in designing embedded system

Embedded systems are within every industry, from aerospace to consumer applications. With the new advances in embedded systems design, more complex applications may be implemented. During the development of an embedded system certain process models are followed. These models usually include the development of a working prototype of the final system. Embedded systems are single-functioned systems which are tightly constrained by power and cost, and are reactive and real-time. Embedded problems can be solved using different approaches. Approaches that are used in

practice are as follows.

- The designer can use a combined hardware/software approach that contains some custom hardware and an embedded processor core integrated within the custom hardware.
- The designer can create custom software that runs on an off-the-shelf embedded processor.
- The designer uses another type of processor besides a general purpose embedded processor, such as a digital signal processor, and a custom software.

In order to design a near optimal system, the following need to be considered besides the functionality and safety of the system.

- Cost
- Performance
- Power
- Maintainability
- Size
- Time-to-market

Many embedded systems have substantially different design constraints than desktop computing applications. A single characterization cannot apply to the diverse spectrum of embedded system, and the considerations are weighed differently, based on the type of application and consumers.

The cost of the embedded system is a very important factor during the embedded system design process. The affordability of the product by many consumers and the profit that can be generated by the device is important while designing.

Performance is a factor that is always considered in systems. An embedded system should perform its functions and complete them quickly and accurately. High performance is especially emphasized in many embedded systems.

Low power is an important requirement for embedded systems. The embedded systems usually run on batteries and should last a long time before those batteries need to be changed. An ultra-low power design needs to be developed for long-term battery operation.

In many cases embedded systems must be repairable in a few minutes to a few hours, which imply that spare components and maintenance personnel must be located close to the system. A fast repair time may also imply that extensive diagnosis and data collection capabilities must be built into the system, which may affect the goal of keeping production costs low. A system self-test can be created in the design to lower the maintenance and diagnosis costs that might be incurred later.

Typically, embedded computers are physically located within some larger device or casing. Therefore, their shape and size may be dictated by the space available and the connections to the mechanical components. Time-to-market (TTM) is the length of time from the product idea conception until it is available for sale. TTM is important in industries where products are outdated quickly, such as the technology industry. The market window, shown in Figure 1, is crucial to deploying a product in the embedded systems technology industry. The typical TTM is eight months. The company needs to deploy when the peak revenue can be attained.

2.3 Architecture Of embedded System

Design of embedded systems has evolved from the transistor level to gate level and register transfer level. Having a higher level of abstraction is beneficial when implementing complex hardware systems. Programming at an even higher level known as the system level, the designer can be concerned with the functionality of the system being designed. At this higher level of abstraction, the designer can specify the functionality of the system using a procedural language, such as the C language.

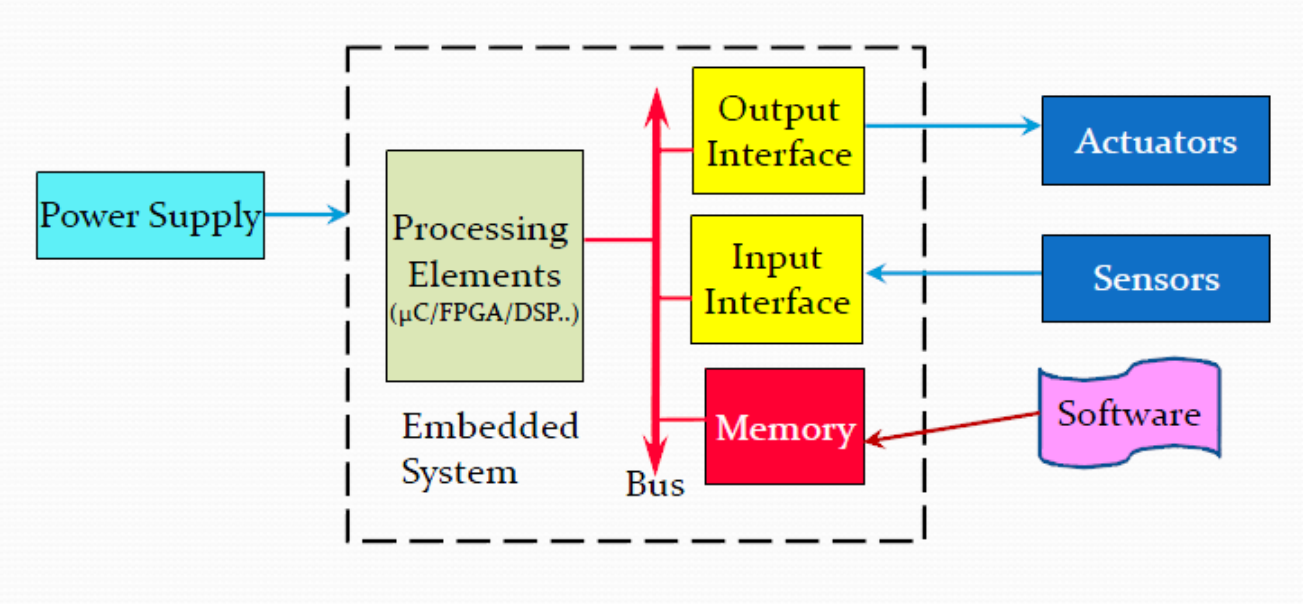

Figure 2.1: Architecture Of Embedded System

2.3.1 Embedded System Element

- I/O(Peripheral) Interface
- Processing Element (Microcontroller or Microprocessor / FPGA / DSP / ASIP, etc)
- Memory (Program and Data)

- Bus
- Software

I/O(Peripheral) Interface

Peripherals consist of inputs and outputs to the embedded system. The design of the system relies heavily on the understanding of the communication interfaces in and out of the device. There are different types of devices that can be used as inputs and outputs. Sensors, LCD displays, speakers, keyboards, and infrared devices are used to communication with the outside world.

There are many different types of sensors. Sensors are designed for many physical quantities, such as water, image, pressure, infrared, sound, and biometrics. [3] Smart systems can only be developed due to the use of sensors. The other communication interfaces include serial, USB, and Ethernet. The communication interfaces can be categorized as wireless media, optical media, and wires. The wireless media includes radio frequency and infrared signal based communication.

The communication interfaces are essential when the embedded system connect to other devices to communicate data for processing. If the system is independent from other devices the user interfaces are mainly used and the serial and USB communication are encapsulated within the system.

- Input Interface:
  - External sub-system for data /command reception
  - Sensor (Thermistor, accelerometer ,etc)
- Output Interface:
  - Actuators (Valve, motors(DC or stepper), etc)

  - External Sub-system for data/status transmission

- Electrical Interfaces:
  - Simple Single ended CMOS/TTL
  - Analog input from sensor
- Protocols
  - Simple Parallel data
  - Serial interfaces
    - Commercial protocols like UART, SPI, I2C,etc
    - Application Specific protocols

2.3.2 Processing Element

Processing element is brain of embedded systems.It provide platform on which software executes.It has resource for Arithmetic & Logical operation, conditional operation and sequential task etc.

Types Of Processing Element

- Micro-controller & Microprocessor

  A large number of processors used in embedded systems are in fact microcontrollers. Microcontrollers can be used easily in a design. The difference between a microprocessor and microcontroller is that a microprocessor is contained within a microcontroller. A microcontroller is essentially a computer system on a chip and it contains a processor core, memory, and programmable input/output peripherals.

- FPGA(Field Programmable Gate Array)

  The FPGA can be used as the main processor in an embedded system, a coprocessor, or a processor used for quick prototyping. The FPGA is capable of

implementing very large and complex functions and it can also perform DSP, ASIC, and Microcontroller functions. The flexibility offered by FPGAs is incredible because the design can be easily changed without much effort from the designer. This flexibility isnt available when designing with ASICs and DSPs. Microcontrollers offer some flexibility but only with regard to the software of the system, the hardware cannot be changed.

- ASIC(Application Specific Integrated Circuits)
  An ASIC can be easily used for an embedded system that has a single purpose.Gate arrays are based on basic cells consisting of a collection of unconnected transistors and resistors. The vendor determines the optimum mix to be provided in a basic cell.

- DSP(Digital Signal Processor)
  Digital Signal Processors or DSP are best suited for handling digital signal processing applications. Digital signal processing is the branch of electronics that is concerned with the representation and manipulation of signals in digital form. When the requirement of the embedded system is to mainly perform signal processing, such as wireless signals, DSPs should be chosen. Besides DSPs, there are different methods of implementing the signal processing application. The alternative choices are to use a general purpose microprocessor, dedicated ASIC hardware, dedicated FPGA hardware. DSPs are superior to these alternatives because the chip is designed to perform digital signal processing tasks much faster, more efficiently, and lower cost.

- PSoc(Programmable System On chip) PSoC Integrates configurable analog and digital peripheral functions, memory and a microcontroller on a single chip. It contain different IP(Intellectual Properties) for different modules.

2.3.3 Memory

The data and program needs to be stored in some kind of memory. The storage needs to be completed in an efficient way. The run-time, code-size, and energy efficiency needs to be considered. A good compiler and compression techniques assists in achieving code-size efficiency. Memory hierarchies can be utilized to achieve good run-time and energy efficiency.

2.3.4 Bus

Buses are used for communication between two devices. According to the requirement, situation, length of communication and speed bus system and protocols are changed.

2.3.5 Embedded Software

In embedded software there are choice of select programming model or programming structure hardware required. Choice are as follow:

- Assembly language
- Embedded Programming Language
  (High level programming language)
- Embedded Operating System

2.4 Automation

In the robotics area, when ever automation come in picture regarding motion control(position control) motor required. Motors are of different type like:

- DC motor
- Servo motor

- Stepper motor

According to project requirement, there is need of stepper motor to control motion. Stepper motor are of two type:

- Unipolar

  Unipolar motors use 6 wires and require a unipolar driver. In addition to the A and B phases, there are two extra wires called the common wires: Current always flows in one direction: from the phases, through the common wires. In addition, only one portion of the motor is energized at a time. Unipolar stepper motors can be used in three modes namely the Wave Drive, Full Drive and Half Drive mode.

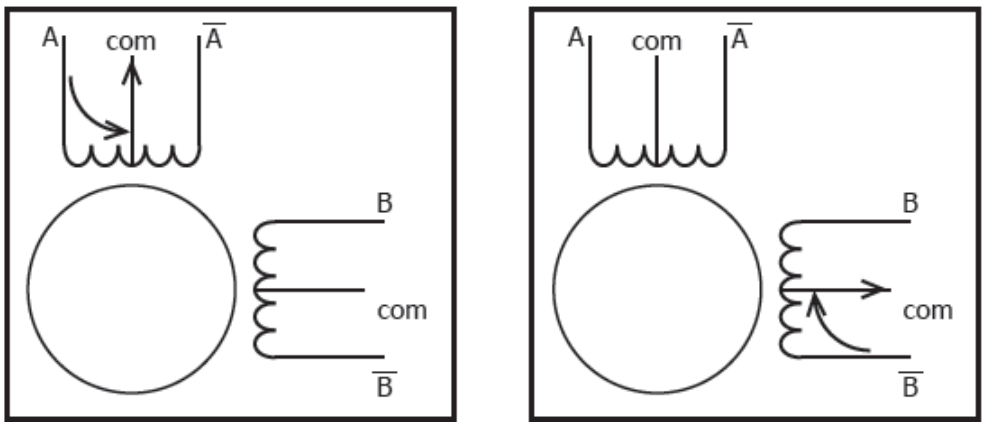

Figure 2.2: Unipolar Stepper Motor

  – Wave Drive:

    n this mode only one electromagnet is energized at a time. Generated torque will be less when compared to full drive in which two electromagnets are energized at a time but power consumption is reduced. It has same number of steps as in the full drive. This drive is preferred when power consumption is more important than torque. It is rarely used.

Table 2.1: Wave Drive

Step	A	B	C	D
Step1	1	0	0	0
Step2	0	1	0	0
Step3	0	0	1	0
Step4	0	0	0	1

– Full Drive:

In this mode two electromagnets are energized at a time, so the torque generated will be larger when compared to Wave Drive. This drive is commonly used than others. Power consumption will be higher than other modes.

Table 2.2: Full Drive

Step	A	B	C	D
Step1	1	1	0	0
Step2	0	1	1	0
Step3	0	0	1	1
Step4	1	0	0	1

– Half Drive

In this mode alternatively one and two electromagnets are energized, so it is a combination of Wave and Full drives. This mode is commonly used to increase the angular resolution of the motor but the torque will be less, about 70% at its half step position. We can see that the angular resolution doubles when using Half Drive.

Table 2.3: Half Drive

Step	A	B	C	D
Step1	1		0	0
Step2	1	1		0
Step3	0	1	0	0
Step4	0	1	1	0
Step5	0	0	1	0
Step6	0	0	1	1
Step7	0	0	0	1
Step8	1	0	0	1

- Bipolar Bipolar motors use 4 wires and require a bipolar drive. Common wires are not used. Current can flow in two directions. In addition, two phases can be energized at one time. Above all three mode work for bipolar also.

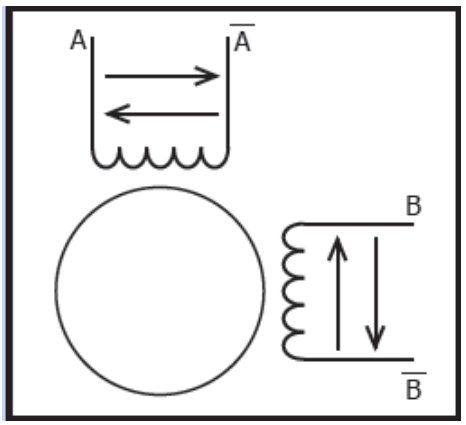

Figure 2.3: Bipolar Stepper Motor

- Microstepping

  All step motors can be microstepped (microstepping is a function of the driver). By sending different amounts of currents to each phase, it forces the motor to

make a step in between its natural full step positions. In order to half step a motor, instead of sending 100% current to both coils, it switches off and sends A Phase 100% and B Phase 0 current or vise versa. If motor has step angle 1.8. It take 200 step per revolution.

2.5 Design and Development Process

In this section, we will begin by presenting a general approach to modular design. In specific, we will discuss how to organize software blocks in an effective manner. The ultimate success of an embedded system project depends both on its software and hardware. Computer scientists pride themselves in their ability to develop quality software. Similarly electrical engineers are well-trained in the processes to design both digital and analog electronics. Manufacturers, in an attempt to get designers to use their products, provide application notes for their hardware devices. The main objective of this is to combine effective design processes together with practical software techniques in order to develop quality embedded systems.

These software skills include: modular design, layered architecture, abstraction, and verification. Writing good software is an art that must be developed, and cannot be added on at the end of a project. Just like any other discipline (e.g., music, art, science, religion), expertise comes from a combination of study and practice. Good software combined with average hardware will always outperform average software on good hardware. One of the hardest steps when designing a system is "where do I start?" This section will address this critical aspect of design.

2.5.1 Product Life Cycle And Requirement

In this section, we will introduce the product development process in general. As we learn software/hardware development tools and techniques, we can place them into the framework presented in this section. As illustrated in Figure 2.4, the development of a product follows an analysis−design−implementation−testing−deployment cycle. For complex systems with long life−spans, we transverse multiple times around the life cycle. During the analysis phase, we discover the requirements and constraints

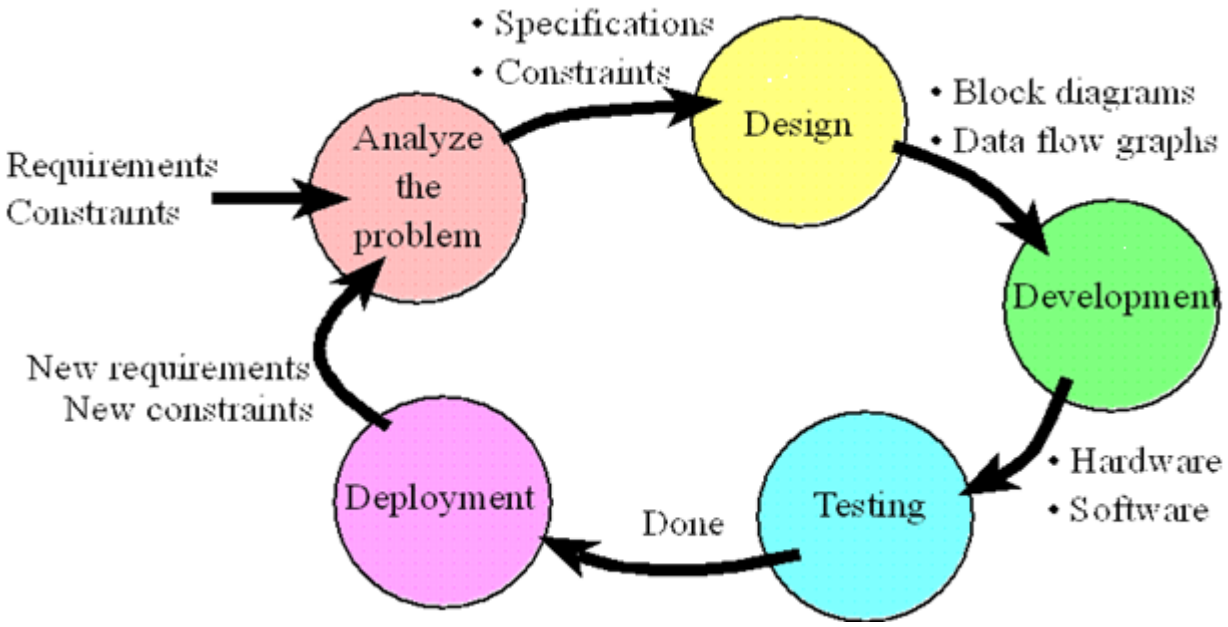

Figure 2.4: Product Life Cycle

for our proposed system. We can hire consultants and Interview potential customers in order to gather this critical information. A requirement is a specific parameter that the system must satisfy. We begin by rewriting the system requirements, which are usually written in general form, into a list of detailed specifications. In general, specifications are detailed parameters describing how the system should work. For example, a requirement may state that the system should fit into a pocket, whereas a specification would give the exact size and weight of the device. For example, suppose we wish to build a motor controller. During the analysis phase, we would

determine obvious specifications such as range, stability, accuracy, and response time.

There may be less obvious requirements to satisfy, such as weight, size, battery life, product life, ease of operation, display readability, and reliability. Often, improving the performance on one parameter can be achieved only by decreasing the performance of another. This art of compromise defines the trade−offs an engineer must make when designing a product. A constraint is a limitation, within which the system must operate. The system may be constrained to such factors as cost, safety,compatibility with other products, use of specific electronic and mechanical parts as other devices,interfaces with other instruments and test equipment, and development schedule.

The following measures are often considered during the analysis phase of a project:

Safety: The risk to humans or the environment

Accuracy: The difference between the expected truth and the actual parameter

Precision: The number of distinguishable measurements

Resolution: The smallest change that can be reliably detected

Response time: The time between a triggering event and the resulting action

Bandwidth: The amount of information processed per time

Maintainability: The flexibility with which the device can be modified

Testability: The ease with which proper operation of the device can be verified

Compatibility: The conformance of the device to existing standards

Mean time between failure: The reliability of the device, the life of a product

Size and weight: The physical space required by the system

Power: The amount of energy it takes to operate the system

Nonrecurring engineering cost : The one-time cost to design and test

Unit cost: The cost required to manufacture one additional product

Time-to-prototype: The time required to design, build, and test an example system

Time-to-market: The time required to deliver the product to the customer

Human factors: The degree to which our customers like/appreciate the product

2.5.2 Requirements Document

The following is one possible outline of a Requirements Document. IEEE publishes a number of templates that can be used to define a project (IEEE STD 830−1998). A requirements document states what the system will do. It does not state how the system will do it. The main purpose of a requirements document is to serve as an agreement between you and your clients describing what the system will do. This agreement can become a legally binding contract. Write the document so that it is easy to read and understand by others. It should be unambiguous, complete, verifiable, and modifiable.

When we begin the design phase, we build a conceptual model of the hardware/software system. It is in this model that we exploit as much abstraction as appropriate. The project is broken into modules or subcomponents. During this phase, we estimate the cost, schedule, and expected performance of the system. At this point we can decide if the project has a high enough potential for profit.

A data flow graph is a block diagram of the system, showing the flow of information. Arrows point from source to destination. The rectangles represent hardware

components, and the ovals are software modules. We use data flow graphs in the high−level design, because they describe the overall operation of the system while hiding the details of how it works. Issues such as safety (e.g., Isaac Asimov‘s first Law of Robotics “A robot may not harm a human being, or, through inaction, allow a human being to come to harm ”) and testing (e.g., we need to verify our system is operational) should be addressed during the high-level design.

A data flow graph for a simple position measurement system is shown in Figure 2.5. The sensor converts position in an electrical resistance. The analog circuit converts resistance into the 0 to +3V voltage range required by the ADC. The ADC converts analog voltage into a digital sample. The ADC driver, using the ADC and timer hardware, collects samples and calculates voltages. The software converts voltage to position. Voltage and position data are represented as fixed−point numbers within the computer. The position data is passed to the OLED driver creating ASCII strings, which will be sent to the organic light emitting diode (OLED) module. A

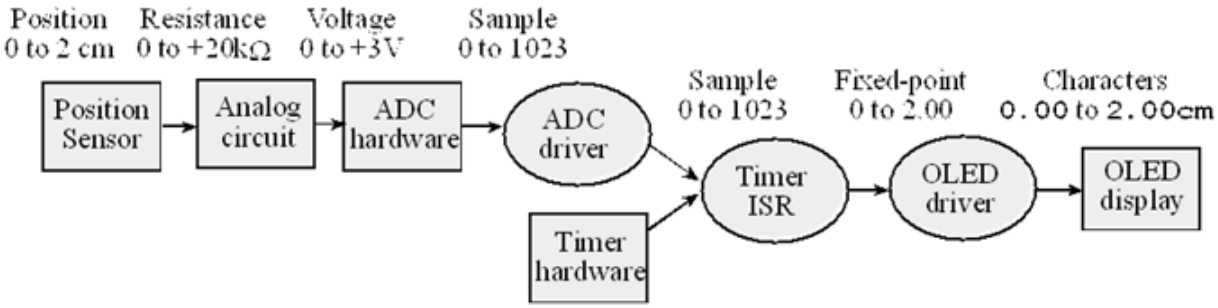

Figure 2.5: A data flow graph showing how the position signal passes through the system

preliminary design includes the overall top-down hierarchical structure, the basic I/O signals, shared data structures, and overall software scheme. At this stage there should be a simple and direct correlation between the hardware/software systems and the conceptual model developed in the high−level design. Next, we finish the

top−down hierarchical structure and build mock-ups of the mechanical parts (connectors, chassis, cables etc.) and user software interface. Sophisticated 3−D CAD systems can create realistic images of our system. Detailed hardware designs must include mechanical drawings. It is a good idea to have a second source, which is an alternative supplier that can sell our parts if the first source can't deliver on time.

Call graphs are a graphical way to define how the software/hardware modules interconnect. Data structures, which will be presented throughout the class, include both the organization of information and mechanisms to access the data. Again safety and testing should be addressed during this low-level design.

A call graph for a simple position measurement system is shown in Figure 2.6. Again, rectangles represent hardware components, and ovals show software modules. An arrow points from the calling routine to the module it calls. The I/O ports are organized into groups and placed at the bottom of the graph. A high-level call graph, like the one shown in Figure 2.6, shows only the high−level hardware/software modules. A detailed call graph would include each software function and I/O port. Normally, hardware is passive and the software initiates hardware/software communication, it is possible for the hardware to interrupt the software and cause certain software modules to be run. In this system, the timer hardware will cause the ADC software to collect a sample. The timer interrupt service routine (ISR) gets the next sample from the ADC software, converts it to position, and displays the result by calling the OLED interface software. The double−headed arrow between the ISR and the hardware means the hardware triggers the interrupt and the software accesses the hardware.

The next phase involves developing an implementation. An advantage of a top−down

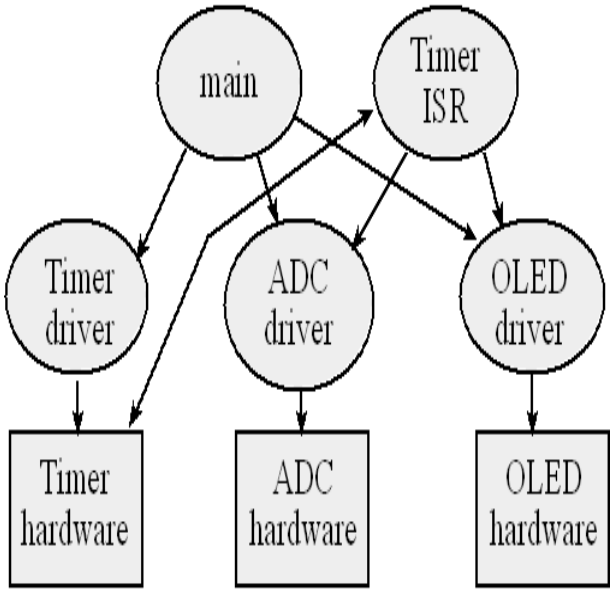

Figure 2.6: A call graph for a simple position measurement system

design is that implementation of sub−components can occur simultaneously. During the initial iterations of the life cycle, it is quite efficient to implement the hardware/software using simulation. One major advantage of simulation is that it is usually quicker to implement an initial product on a simulator versus constructing a physical device out of actual components. Rapid prototyping is important in the early stages of product development. This allows for more loops around the analysis−design−implementation−testing−dep cycle, which in turn leads to a more sophisticated product.

Recent software and hardware technological developments have made significant impacts on the software development for embedded microcomputers. The simplest approach is to use a cross−assembler or cross-compiler to convert source code into the machine code for the target system. The machine code can then be loaded into the target machine. Debugging embedded systems with this simple approach is very difficult for two reasons. First, the embedded system lacks the usual keyboard and display that assist us when we debug regular software. Second, the nature of embed-

ded systems involves the complex and real−time interaction between the hardware and software. These real−time interactions make it impossible to test software with the usual single-stepping and print statements.

The next technological advancement that has greatly affected the manner in which embedded systems are developed is simulation. Because of the high cost and long times required to create hardware prototypes, many preliminary feasibility designs are now performed using hardware/software simulations. A simulator is a software application that models the behavior of the hardware/software system. If both the external hardware and software program are simulated together, even though the simulated time is slower than the clock on the wall, the real−time hardware/software interactions can be studied.

During the testing phase, we evaluate the performance of our system. First, we debug the system and validate basic functions. Next, we use careful measurements to optimize performance such as static efficiency (memory requirements), dynamic efficiency (execution speed), accuracy (difference between expected truth and measured), and stability (consistent operation.)

Maintenance is the process of correcting mistakes, adding new features, optimizing for execution speed or program size, porting to new computers or operating systems, and reconfiguring the system to solve a similar problem. No system is static. Customers may change or add requirements or constraints. To be profitable, we probably will wish to tailor each system to the individual needs of each customer. Maintenance is not really a separate phase, but rather involves additional loops around the life cycle.

Figure 2.4 describes top−down design as a cyclic process, beginning with a problem statement and ending up with a solution. With a bottom−up design we begin with solutions and build up to a problem statement. Many innovations begin with an idea, "what if?"In a bottom−up design, one begins with designing, building, and testing low−level components. The low−level designs can be developed in parallel. Bottom−up design may be inefficient because some subsystems may be designed, built, and tested, but never used. As the design progresses the components are fit together to make the system more and more complex. Only after the system is completely built and tested does one define the overall system specifications. The bottom−up design process allows creative ideas to drive the products a company develops. It also allows one to quickly test the feasibility of an idea. If one fully understands a problem area and the scope of potential solutions, then a top−down design will arrive at an effective solution most quickly. On the other hand, if one doesn‘t really understand the problem or the scope of its solutions, a bottom-up approach allows one to start off by learning about the problem.

2.5.3 Debugging Theory

Functional debugging involves the verification of input/output parameters. Functional debugging is a static process where inputs are supplied, the system is run, and the outputs are compared against the expected results. Four methods of functional debugging are presented in this section, and two more functional debugging methods are presented in the next chapter after indexed addressing mode is presented.

There are two important aspects of debugging: control and observability. The first step of debugging is to stabilize the system. In the debugging context, we stabilize the problem by creating a test routine that fixes (or stabilizes) all the inputs.

In this way, we can reproduce the exact inputs over and over again. Stabilization is an effective approach to debugging because we can control exactly what software is being executed. Once stabilized, if we modify the program, we are sure that the change in our outputs is a function of the modification we made in our software and not due to a change in the input parameters. When a system has a small number of possible inputs (e.g., less than a million), it makes sense to test them all. When the number of possible inputs is large we need to choose a set of inputs. There are many ways to make this choice. We can select values: Near the extremes and in the middle. Most typical of how our clients will properly use the system. Most typical of how our clients will improperly attempt to use the system. That differ by one.You know your system will find difficult, Using a random number generator. To stabilize the system we define a fixed set of inputs to test, run the system on these inputs, and record the outputs.

Debugging is a process of finding patterns in the differences between recorded behavior and expected results. The advantage of modular programming is that we can perform modular debugging. We make a list of modules that might be causing the bug. We can then create new test routines to stabilize these modules and debug them one at a time. Unfortunately, sometimes all the modules seem to work, but the combination of modules does not. In this case we study the interfaces between the modules, looking for intended and unintended (e.g., unfriendly code) interactions.

Many debuggers allow you to set the program counter to a specific address then execute one instruction at a time. The debugger provides three stepping commands Step, StepOver and StepOut commands. Step is the usual execute one assembly instruction. However, when debugging C we can also execute one line of C.StepOver will execute one assembly instruction, unless that instruction is a subroutine call, in which case the debugger will execute the entire subroutine and stop at the instruction following the subroutine call.StepOut assumes the execution has already entered

a subroutine, and will finish execution of the subroutine and stop at the instruction following the subroutine call.

A breakpoint is a mechanism to tag places in our software, which when executed will cause the software to stop. Normally, you can break on any line of your program.

One of the problems with breakpoints is that sometimes we have to observe many breakpoints before the error occurs. One way to deal with this problem is the conditional breakpoint. To illustrate the implementation of conditional breakpoints, add a global variable called Count and initialize it to 32 in the initialization ritual. Add the following conditional breakpoint to the appropriate location in your software. Using the debugger, we set a regular breakpoint at bkpt. We run the system again (you can change the 32 to match the situation that causes the error.)

Notice that the breakpoint occurs only on the 32nd time the break is encountered. Any appropriate condition can be substituted. Most modern debuggers allow you to set breakpoints that will trigger on a count. However, this method allows flexibility of letting you choose the exact conditions that cause the break

The use of print statements is a popular and effective means for functional debugging. One difficulty with print statements in embedded systems is that a standard printer may not be available. Another problem with printing is that most embedded systems involve time-dependent interactions with its external environment. The print statement itself may be so slow, that the debugging process itself causes the system to fail.In this regard, the print statement is intrusive rely on the availability of a standard output device.

Every programmer is faced with the need to debug and verify the correctness of his or her software. A debugging instrument is hardware or software used for the

purpose of debugging. In this section, we see hardware−level probes like the logic analyzer, oscilloscope, and Joint Test Action Group (JTAG standardized as the IEEE 1149.1); software-level tools like simulators, monitors, and profilers; and manual tools like inspection and print statements. Nonintrusiveness is the characteristic or quality of a debugger that allows the software/hardware system to operate normally as if the debugger did not exist. Intrusiveness is used as a measure of the degree of perturbation caused in system performance by the debugging instrument itself. For example, a print statement added to your source code is very intrusive because it significantly affects the real−time interaction of the hardware and software. It is important to quantify the intrusiveness of an instrument.

In a real microcomputer system, breakpoints and single−stepping are intrusive, because the real hardware continues to change while the software has stopped. When a program interacts with real−time events, the performance can be significantly altered when using intrusive debugging tools. On the other hand, dumps with filter, and monitors (e.g., output strategic information on an LED or LCD) are much less intrusive. A logic analyzer that passively monitors the activity of the software is completely nonintrusive. Interestingly, breakpoints and single−stepping on a mixed hardware/software simulator are often nonintrusive, because the simulated hardware and the simulated software are affected together.

Although, a wide variety of program monitoring and debugging tools are available today, in practice it is found that an overwhelming majority of users either still prefer or rely mainly upon “rough and ready ”manual methods for locating and correcting program errors. These methods include desk−checking, dumps, and print statements, with print statements being one of the most popular manual methods. Manual methods are useful because they are readily available, and they are relatively simple to use. But, the usefulness of manual methods is limited: they tend to be highly intrusive, and they do not provide adequate control over repeatability, event selection, or event

isolation. A real−time system, where software execution timing is critical, usually cannot be debugged with simple print statements, because the print statement itself will require too much time to execute.

Debugging Dumps

To solve these limitations, we can add a debugging instrument that dumps strategic information into an array at run time. We can then observe the contents of the array at a later time. One of the advantages of dumping is that the JTAG debugger allows you to visualize memory even when the program is running. So this technique will be quite useful in systems with a JTAG debugger. Assume happy and sad are strategic 8-bit variables. The first step when instrumenting a dump is to define a buffer in RAM to save the debugging measurements.

One problem with dumps is that they can generate a tremendous amount of information. If you suspect a certain situation is causing the error, you can add a filter to the instrument. A filter is a software/hardware condition that must be true in order to place data into the array. In this situation, if we suspect the error occurs when another variable gets large, we could add a filter that saves in the array only when the variable is above a certain value.

Chapter 3

Raspberry Pi Board(RPi)

3.1 Introduction

Raspberry pi is a small size board.Size of the board is approximately size of the credit card. Educational and official name of the Raspberry Pi is RPi. RPi board is developed by UK the Raspberry Pi foundation in Cambridge university[8, 9].Basic objective of RPi board is for education purpose. But now this board is used in many area like robotics, automation and at experimental purpose.RPi board is very low cost which is affordable for the students and other user of it.It's availability is good.It is like a general CPU and RPi also support Linux operating system which make it more usable.Due to it's low cost and support open source OS and also availability of it attack use and student towards it.It is developed by a group of Cambridge university professors and students.

3.2 Component Technologies

3.2.1 CPU

The Central Processing Unit is the "brains" of the computer, processing all commands and instructions that make your computer perform tasks and run programs.

3.2.2 GPU

The Graphics Processing Unit processes and converts all the data needed to output video to your monitor. This can be over various media types such as HDMI, VGA, Display Port etc.

3.2.3 RAM

Random Access Memory is used by the system to store data that is currently being accessed or cached for quick retrieval. RAM is known as volatile memory because any data stored here will be lost when the power is turned off.

3.2.4 Hard Disk

The Hard Disk is a type of long term storage, non-volatile memory, because data stored here is not lost when the power is switched off. Hard Disks usually consist of glass disks coated in a magnetic substance that stores the data as bits. These can be read from and written to using a small magnet on an arm that hovers very closely to the disk surface. Hard Disks are traditionally used for storing the Operating System and for your documents, pictures and videos.

3.2.5 SD Card

Raspberry Pi does not come with a Hard Disk for storing the OS instead it uses another form of non-volatile memory known as a Secure Digital Card. This card is a form of flash memory that can be written to and read from and will retain data when the power supply is stopped. SD Cards come in a large array of physical formats and storage capacities allowing them to be used in a range of devices including Mobile Phones, Digital Cameras and removable storage in a PC and now the Raspberry Pi.

3.2.6 Motherboard

The Motherboard acts as the backbone of the computer allowing all of the separate components such as CPU, RAM, Hard Disk etc, to communicate with each other.

3.2.7 Network Interface Card

The Network Interface Card, or NIC, is used to connect your Raspberry Pi to a LAN. The Raspberry Pi uses a standard RJ45 port that accepts an Ethernet cable, technically called shielded/ unshielded twisted pair. The other end is connected to a switch/hub or your home router. Once the network settings have been configured you will be able to access other resources on your LAN or the Internet.

3.2.8 USB Port

The USB port is used to connect many types of peripherals such as a keyboard and mouse to interface with the Pi or a USB PenDrive/Hard Disk to add more storage capacity.

3.2.9 Micro USB

The Micro USB port is used to power the Raspberry Pi, you can typically use a mobile phone charger to do this.

3.2.10 Composite Video

This video out port is used when HDMI is unavailable on the monitor or television the user wishes to display output on. Almost all televisions have an input for this medium, called composite in and is usually found with two other identical ports. These are red/white for left/right audio and yellow for video. The disadvantage to using this video out is the poorer resolution when compared with HDMI.

3.2.11 Audio Out

The audio out used on the Raspberry Pi is a standard 3.5mm analogue jack commonly found on MP3 players and mobile phones. The HDMI port is also capable of outputting audio to a compatible device.

3.2.12 GPIO

The General Purpose Input/Output is a fully user programmable interface used to connect almost anything that has a compatible receiving connection.

3.3 Description of RPi Board

The RPi Board contain many facilities on a small board.It contain processor and graphics chip, program memory(RAM), and various interfaces and connector for external devices.Some of these devices are essential and some are optional. RPi operate in the same way as any stander PC.

It also requires mass-storage, but a hard disk drive of the type found in a typical PC is not really in keeping with the miniature size of RPi. Instead we will use an SD Flash memory card normally used in digital cameras, configured in such a way to look like a hard drive to RPis processor. RPi will boot (load the Operating System into RAM) from this card in the same way as a PC boots up into Windows from its hard disk.

RPi board description is as follow:

RPi board has following chips, ports and GPIO (terminal available on board)

- BROADCOM BCM2835: It is an SoC which contain ARM1176JZF-S and peripherals which safely be controlled by ARM processor.
  Peripherals are as follow[8, 9] :

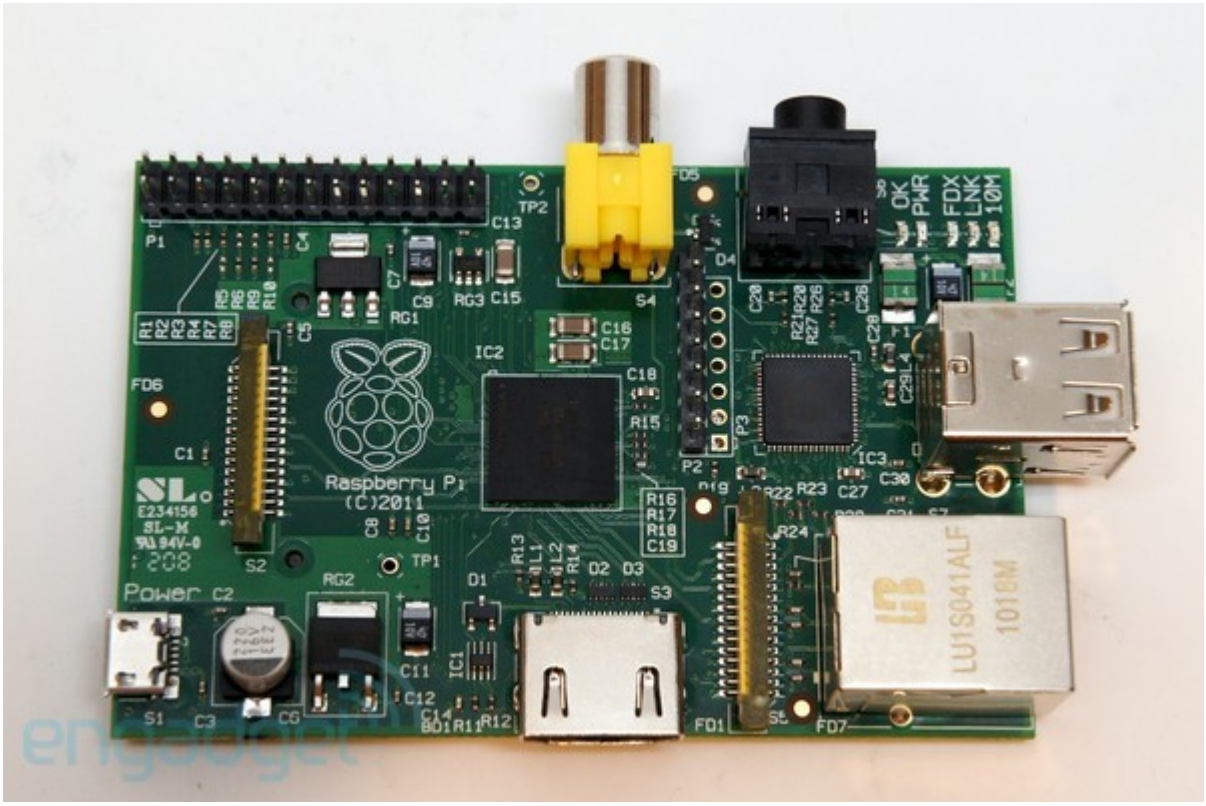

Figure 3.1: Raspberry Pi Board

– Timers
– Interrupt controller
– GPIO
– USB
– PCM/I2S
– DMA controller
– I2C master
– I2C / SPI slave
– SPI0, SPI1, SPI2
– PWM
– UART0, UART1

- LAN controller: Control and stabilize the Ethernet connection
- CSI connector camera: For connect the camera through RPi
- Status LED: For show the LAN connection status

Port Available on RPi:

- HDMI Port: Provide connection to the monitor screen and enable display
- RCA video output: If HDMI monitor not available you can use this
- Audio jack
- USB 2.0 port: For connect Mouse, key board, Pendrive and USB hub
- Ethernet Port : For LAN and WiFi connection
- Micro USB Port: For provide power supply
- GPIO Header: To access GPIO pins for control external hardware
- JTAG header: For provide connection to the JTAG connection device
- DSI connector Display: For another display purpose
- SD Card connector: For connect SD card which is external memory contain OS like Hard Disk in CPU

3.4 Specification of RPi Board

There are some specification regarding RPi board which help to know for proper utilization of board. It shows the facility, flexibility and limitation of board.Current and voltage specification for the GPIO pins.[10]

Table 3.1: Board Specification

Parameter	Model B
Price Of Board	US$ 35
SoC	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, and single USB port)
CPU	700 MHz ARM1176JZF-S core (ARM11 family, ARMv6 instruction set)
GPU	Broadcom VideoCore IV @ 250 MHz,OpenGL ES 2.0 (24 GFLOPS), 1080p30 h.264/MPEG-4 AVC high-profile decoder and encoder, MPEG-2 and VC-1
SDRAM	512 MB (shared with GPU)
USB 2.0 ports	2
Video input	A CSI input connector allows for the connection of a RPF designed camera module
Video outputs	Composite RCA (PAL and NTSC), HDMI, raw LCD Panels via DSI
Audio outputs	3.5 mm jack, HDMI
Onboard storage	SD / MMC / SDIO card slot (3.3V card power support only)
Onboard network	10/100 Ethernet (8P8C) USB adapter on the third port of the USB hub for WiFi

Low-level peripherals	8 × GPIO, UART, IC bus, SPI bus with two chip selects, IS audio, +3.3 V, +5 V, ground and Maximum Current Draw from 3.3V is 50mA
Power Rating	700mA(3.5 W)
Power Source	5 volt via MicroUSB or GPIO header
Size	85.60 × 53.98 mm (3.370 × 2.125 in)
Weight	45 g
Operating systems	Raspbian OS, Debian GNU/Linux, Fedora, RISC OS and many

3.5 BCM2835 SoC

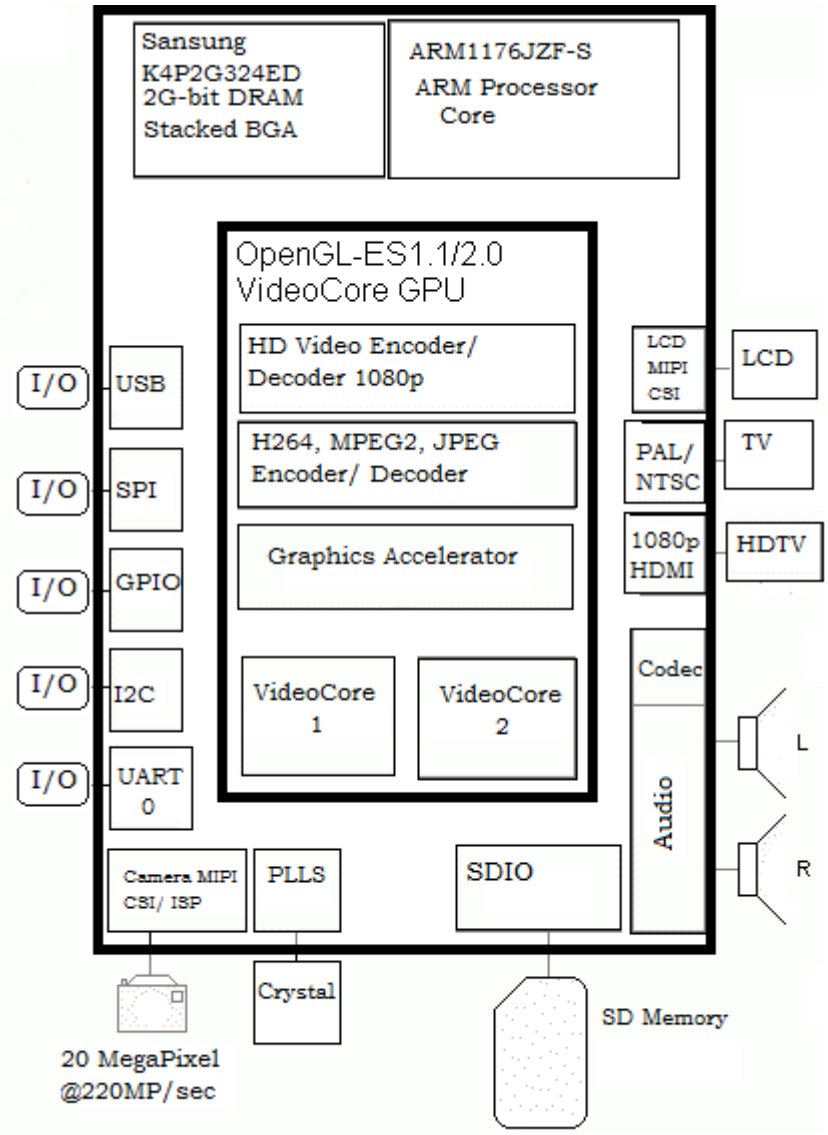

Figure 3.2: Block Diagram Of BCM2835

The Raspberry Pi does not have a separate CPU (Central Processing Unit), RAM (Random Access Memory) or GPU (Graphics Processing Unit). Instead they are all squeezed into one component called a System on Chip or SoC unit. This is essentially the entire computer on one chip.

The Raspberry Pi uses an ARM1176JZF−S 700MHz CPU which is also installed in a wide variety of mobile phones, hand held games consoles and eReaders. This CPU is single core, however it does have a co-processor to perform floating point calculations. Many calculations required by a program involve whole numbers (integers). These are easier for the CPU to handle. Integer calculations produce accurate results. Floating point or real numbers have a fractional part e.g. 1.5. They are more demanding for the CPU to process.

The Model B Raspberry Pi has 512MB SDRAM (Synchronous Dynamic RAM). This is working memory that is used to store programs that are currently being run in the CPU.

The ARM CPU has 32KB of Level 1 cache memory for instructions and 32KB for data. It also has 128KB of Level 2 cache memory. Cache memory is important in improving the performance of the system because it stores recently used program lines copied from RAM ready to be used again if needed. Most processors have levels of cache. Level 1 is the smallest size but is closest to the CPU core. Level 2 is larger but is situated slightly further away from the CPU core. The CPU is based on 32 bit architecture and has 32 bit registers. It also uses a 32 bit words. A word is a complete piece of information that the CPU can execute. The Arithmetic Logic Unit is the part of the CPU where instructions are executed.

3.6 Generl Purpose Input / Output(GPIO)

There are 54 genaral-purpose I/O(GPIO) lines in RPi.These 54 lines split into two banks.All GPIO pins have at least two alternative function within BCM. The alternate function are usually peripheral I/O and a single peripheral may appear in each bank to allow flexibility on the choice of I/O voltage.

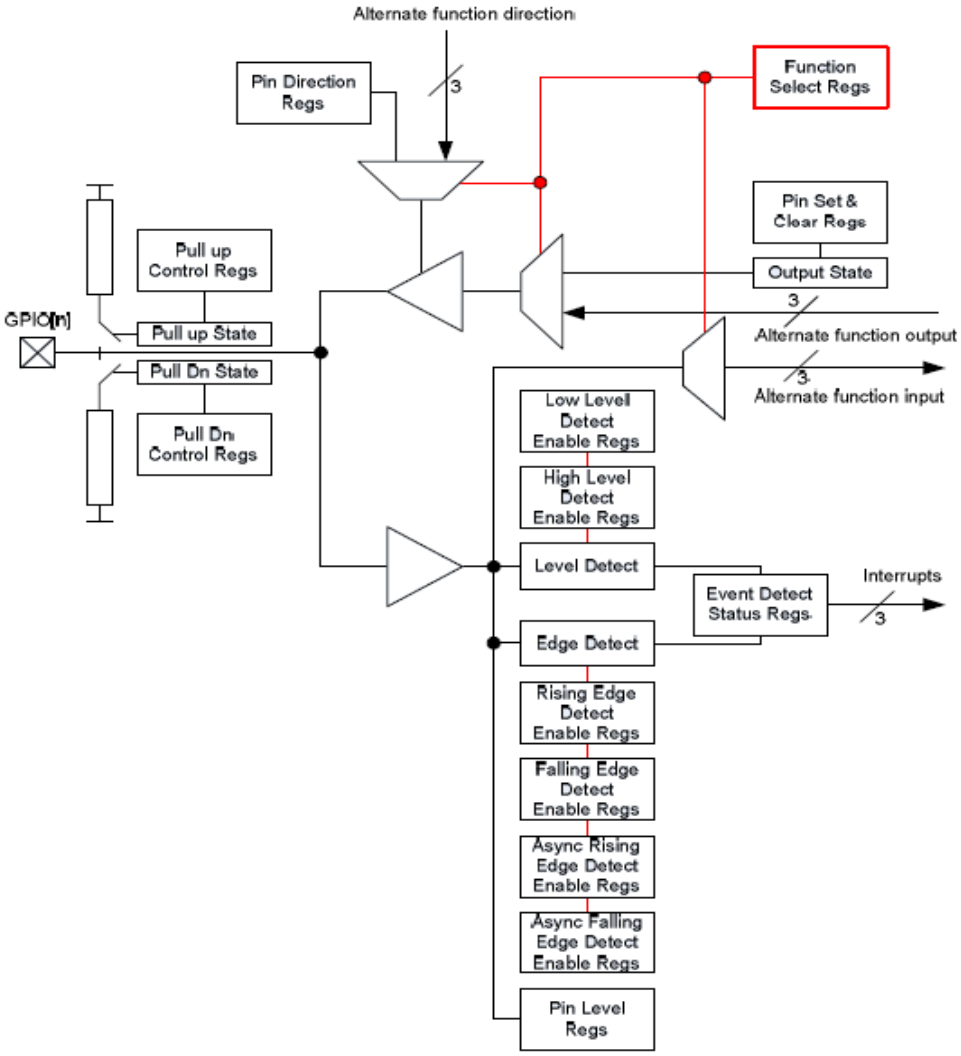

Figure 3.3: GPIO Block Diagram
[18]

3.6.1 Registers Organization Of GPIO

GPIO Function Select Registers The function select registers are used to define the operation of the general-purpose I/O pins. Each of the 54 GPIO pins has at least two alternative functions.All unused alternative function lines are tied to ground and will output a "0 "if selected. All pins reset to normal GPIO input operation.

GPIO Pin Output Set Registers The output set registers are used to set a GPIO pin. The field defines the respective GPIO pin to set, writing a "0 "to the field has no effect. If the GPIO pin is being used as in input (by default) then the value in the field is ignored.

GPIO Pin Output Clear Registers The output clear registers are used to clear a GPIO pin.

GPIO Pin Level Registers The pin level registers return the actual value of the pin.It could be GPIO Level Register 0 and GPIO Level Register 1.

GPIO Event Detect Status Registers The event detect status registers are used to record level and edge events on the GPIO pins. The relevant bit in the event detect status registers is set whenever:

1. an edge is detected that matches the type of edge programmed in the rising/falling edge detect enable registers.
2. a level is detected that matches the type of level programmed in the high/low level detect enable registers. The bit is cleared by writing a "1 "to the relevant bit.

The interrupt controller can be programmed to interrupt the processor when any of the status bits are set. The GPIO peripheral has three dedicated interrupt lines. Each GPIO bank can generate an independent interrupt. The third line generates a single interrupt whenever any bit is set.

GPIO Rising Edge Detect Enable Registers The rising edge detect enable registers define the pins for which a rising edge transition sets a bit in the event detect status registers (GPEDSn). When the relevant bits are set in both the GPRENn and GPFENn registers, any transition (1 to 0 and 0 to 1) will set a bit in the GPEDSn registers. The GPRENn registers use synchronous edge detection. This means the input signal is sampled using the system clock and then it is looking for a 011 pattern on the sampled signal. This has the effect of suppressing glitches.

GPIO Falling Edge Detect Enable Registers (GPRENn) The falling edge detect enable registers define the pins for which a falling edge transition sets a bit in the event detect status registers (GPEDSn). When the relevant bits are set in both the GPRENn and GPFENn registers, any transition (1 to 0 and 0 to 1) will set a bit in the GPEDSn registers. The GPFENn registers use synchronous edge detection. This means the input signal is sampled using the system clock and then it is looking for a 100 pattern on the sampled signal. This has the effect of suppressing glitches.

GPIO High Detect Enable Registers The high level detect enable registers define the pins for which a high level sets a bit in the event detect status register (GPEDSn). If the pin is still high when an attempt is made to clear the status bit in GPEDSn then the status bit will remain set.

GPIO Low Detect Enable Registers (GPLENn) The low level detect enable registers define the pins for which a low level sets a bit in the event detect status register (GPEDSn). If the pin is still low when an attempt is made to clear the status bit in GPEDSn then the status bit will remain set.

GPIO Asynchronous rising Edge Detect Enable Registers (GPARENn) The asynchronous rising edge detect enable registers define the pins for which a asynchronous rising edge transition sets a bit in the event detect status regis-

ters (GPEDSn). Asynchronous means the incoming signal is not sampled by the system clock. As such rising edges of very short duration can be detected.

GPIO Asynchronous Falling Edge Detect Enable Registers (GPAFENn) The asynchronous falling edge detect enable registers define the pins for which a asynchronous falling edge transition sets a bit in the event detect status registers (GPEDSn). Asynchronous means the incoming signal is not sampled by the system clock. As such falling edges of very short duration can be detected.

GPIO Pull-up/down Register (GPPUD) The GPIO Pull-up/down Register controls the actuation of the internal pull-up/down control line to ALL the GPIO pins. This register must be used in conjunction with the 2 GPPUDCLKn registers. Note that it is not possible to read back the current Pull-up/down settings and so it is the users responsibility to remember which pull-up/downs are active. The reason for this is that GPIO pull-ups are maintained even in power-down mode when the core is off, when all register contents is lost. The Alternate function table also has the pull state which is applied after a power down.

GPIO Pull-up/down Clock Registers (GPPUDCLKn) The GPIO Pull-up/down Clock Registers control the actuation of internal pull-downs on the respective GPIO pins. These registers must be used in conjunction with the GPPUD register to effect GPIO Pull-up/down changes.

3.6.2 Pin Configuration Of RPi GPIO Headers

There are 26 pin on header P1 and 8 pin on header P5(only model B of RPi Board) for low level peripheral on RPi board. Pin configuration is as in figure:

Name and number of the GPIO pin as per BCM2835 are shown in the figure. But as per wiring pi library, number of the GPIO pins is different.So for proper GPIO name and number see AppendixC.

Figure 3.4: RPi GPIO
[18]

Chapter 4

Rapbian OS and Software

RPi support different operating system. In this thesis Raspbian "wheezy" operating system is used. It is a customize version for raspberry pi. It's performance and compatibility is well compare to other operating system[13].

Compare to other Linux OS, raspbian also has the same feature. It support the Hierarchy file system, directory and sub-directory system.

4.1 Raspbian File System Hierarchy

Root is main directory. The Root Directory of linux file system contains the following sub-directory:

- /bin
- /sys
- /boot
- /dev
- /etc

- /home
- /lib
- /lost+found
- /media
- /mnt
- /opt
- /proc
- /root
- /sbin
- /usr
- /srv
- /tmp
- /var
- /selinux
- /run

Now let us see the purpose of each folder given above:

4.1.1 /bin

It contains several useful commands that are of use to both the system administrator as well as non-privileged users. Usually contains the shells like bash, csh, etc.... and commonly used commands like cp, mv, rm, cat, ls.

4.1.2 /boot

It contains everything required for the boot process except for configuration files not needed at boot time and the map installer. It stores data that is used before the kernel begins executing user-mode programs.It may include the system kernel (under symbolically linked)

4.1.3 /dev

This sub-directory the location of device files. A device and a file both can be read from and written to this directory. So config a device is same with edit a file.
EX: sending data to /dev/ttyS0 that means you are sending data to a communication device, such as a modem. 'block devices' are devices that store or hold data. 'character devices' can be thought of as devices that transmit or transfer data.

4.1.4 /etc

It Contains all system related configuration files. Local file used to control the operation of a program. Those files must be static and cannot be an executable binary. In this directory there is a sub-directory network. In network sub-directory there is a interface file, which is use for set the IP address setting for LAN and WAN.

4.1.5 /home

The user home directories. Accessible only to its owner and the system administrator. It contains the user?s personal configuration files. It is quite large to be used as User's Documents Space.

4.1.6 /lib

It contains kernel modules and those shared library images (the C programming code library) needed to boot the system and run the commands in the root filesystem, ie.

by binaries in /bin and /sbin. Windows equivalent to a shared library would be a DLL (dynamically linked library) file.

4.1.7 /lost+found

It contains the files which were recovered after an unexpected event, such as a proper shutdown. It is try to move each file back to its original location.

4.1.8 /media

It contains sub-directories which are used as mount points for removable media such as pen-drive, disks, CD-ROM and zip disks.

4.1.9 /mnt

This is a generic mount point under mounted (mount is to make a filesystem available to the system) the filesystems or devices. When a filesystem no longer needs to be mounted, it can be unmounted with umount.

4.1.10 /opt

This directory is reserved for all the software and add-on packages that are not part of the default installation.

4.1.11 /proc

It is virtual filesystem, provide runtime system information (e.g. system memory, devices mounted, hardware configuration, etc). The most of them have a file size of 0. To view, use ?cat?. Use ?vi? to edit.

4.1.12 /root

The home directory of the System Administrator, 'root'. Why not in '/home'? Because '/home' is often located on a different partition or even on another system and would thus be inaccessible to 'root' when - for some reason - only '/' is mounted.

4.1.13 /sbin

/sbin should contain only binaries essential for booting, restoring, recovering, and/or repairing the system in addition to the binaries in /bin.

4.1.14 /usr

The largest share of data on a system. The most important directories in the system as it contains all the user binaries, their documentation, libraries, header files, etc.... X and its supporting libraries, and User programs like telnet, ftp, etc.... as well, can be found here.

4.1.15 /var

It contains variable data, files and directories the system must be able to write to during operation, like system logging files, mail and printer spool directories, and transient and temporary files

4.1.16 /srv

Srv is a serve folder. It holds site specific data to be served by the system for protocols such as, ftp, rsync, www, cvs etc.

4.1.17 /tmp

Tmp is a temporary storage folder. Anything that is to be temporarily stored goes here.It is recommended that you don't delete these manually

4.2 Protocols Support

There are several protocol used by RPi operating system. Some network protocol, bus protocol audio and video protocol. Some protocol is as follow:

4.2.1 TCP

TCP or Transmission Control Protocol is an OSI Layer 4 protocol that ensures connection reliability through the use of Acknowledgment (ACK) packets.

4.2.2 UDP

UDP or User Datagram Protocol is similar to TCP but is considered connection-less and does not ensure that packets are received correctly at the destination but does have advantages such as higher throughput due to lower overhead and is generally used when guaranteed delivery is not essential such as DNS look ups.

4.2.3 FTP

FTP or File Transfer Protocol, is a lightweight protocol using TCP ports 20 and 21 enabling the transferring of files across a network.

4.2.4 TFTP

TFTP or Trivial File Transfer Protocol uses UDP port 69, this is a very lightweight protocol that lacks many of the features of FTP such as user authentication and listing of directories but does allow greater transfer speeds due to the lower overhead.

4.2.5 HTTP

Hyper Text Transfer Protocol is the protocol used by web browsers such as Firefox, Internet Explorer, Chrome, etc to request web pages from a web server on the Internet. HTTP uses the well known TCP port 80.

4.2.6 HTTPS

HTTPS is the same as the HTTP protocol but operates over SSL to provide end to end encryption, you will commonly see this when logging into a secure site such as Internet Banking, Web based Email etc. HTTPS operates over TCP port 443.

4.2.7 SSL

Secure Socket Layer is an OSI layer 6 protocol that provides strong encryption for protocols used by applications such as HTTP and FTP.

4.2.8 SSH (Secure Shell)

Secure Shell is a method of accessing the Command Line Interface of your Raspberry Pi, or any UNIX based OS, over an encrypted connection using TCP port 22

4.2.9 Telnet

Telnet is one of the original protocols developed to allow a user to access a host command line remotely over a LAN or a WAN such as the internet. Telnet is not used extensively anymore due to a lack of security features such as encryption and users now prefer to use SSH. Telnet uses TCP port 23.

4.2.10 DHCP

Dynamic Host Configuration Protocol is a popular protocol used to automatically configure the network settings on a host. Most home routers will provide this service as standard. DHCP uses UDP ports 67 and 68.

4.2.11 DNS

The Domain Name System is used to resolve the URL you type into your browser to an IP address. DNS uses either TCP or UDP port 53.

Apart form above protocol supported for networking. It is also support bus protocols for SPI, I2C, USB, 802.3 for LAN, 802.11g for USB wireless, WiFi and protocol for CSI etc.

4.3 Program and Application Support

It support different application or program.
Some of them are as follows:

- nano editor:
  This application for edit any file normal or administrative.

- LXTerminal:
  It is same as DOS prompt on windows or Terminal window on Linux. Here command line activity can perform. With that command line many utilities can be used. It also provide facility to install new application software through command line and many other services.

- Midori:
  It is a internet browser like Google chrome. For excess internet and use services of internet.

- IDLE and IDLE 3:
  It is the IDE(Integrated development environment) for python language.

- WiFi config:
  It is for config WiFi adopter.

- vi editor: It is a editor. It help to write code in c language and for edit files.

Apart from that it consist many application for video, audio, image and so on.

4.4 Benchmarking Raspberry Pi GPIO Speed

The basic test setup was to toggle one of the GPIO pins, namely the GPIO and see what frequency square wave could be achieved. This basic test setup give idea for any signaling. One can say is basically the upper limit for any signaling with the GPIO pins.

Table 4.1: GPIO Benchmark

Language	**Library**	**Version**	**Square wave**
Shell	/proc/mem access	not applicable	3.4 kHz
Python	RPi.GPIO	0.3.0	44 kHz
Python	wiringPi	github	20 kHz
C	Native library	not applicable	14-22 MHz
C	BCM 2835	1.3	4.7 5.1 MHz
C	wiringPi	not available	6.9 7.1 MHz
Perl	BCM 2835	1.0	35 kH

Different test code with different languages is as follow:

4.4.1 Shell script

The easiest way to manipulate the RPi GPIO pins is via terminal. Here‘s a basic shell script to toggle the GPIO 4 without any delay between on and off:

! /bin/sh echo “4 ”> /sys/class/gpio/export

echo “out ”> /sys/class/gpio/gpio4/direction

while true

do

echo 1 > /sys/class/gpio/gpio4/value

echo 0 > /sys/class/gpio/gpio4/value
done

The performance of this implementation is not as good as expected : A 3400 Hz square wave can be generated using this method. For turnings things on and off this is enough, but no signalling and hardly even LED PWM is feasible.

4.4.2 Python

One of the simplest ways to access the GPIO with a "real programming language"is with the RPi.GPIO Python library. Test script for benchmarking is simple as well:
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(4, GPIO.OUT)
while True:
GPIO.output(4, True)
GPIO.output(4, False

While the 0.2.0 version performance was terrible, the newer 0.3.0 version has significantly improved performance: 44 kHz square wave could be generated. The diagram below is updated for version 0.3.0.

Another alternative for Python are the wiringPi Python bindings. With the following simple test program, a square wave of 19.5 kHz was generated about half the speed of the updated RPi.GPIO library:
import wiringpi

io = wiringpi.GPIO(wiringpi.GPIO.WPI_ MODE_ PINS)

io.pinMode(7,io.OUTPUT)

while True:
io.digitalWrite(7,io.HIGH)
io.digitalWrite(7,io.LOW)

4.4.3 Native C

The Raspberry Pi Wiki gives a nice C code example for true hardware − level access to the GPIO. The interfacing is slightly more difficult, but code isn‘t too bad. I took the example program and simplified the main method after setup_ io() to this:
\\Set GPIO pin 4 to output
INP_ GPIO(4); \\must use INP_ GPIO before we can use OUT_ GPIO
OUT_ GPIO(4);

while(1)
GPIO_ SET = 1<<4 ;
GPIO_ CLR = 1<<4 ;

The measurements for these high frequency signals are now completely redone. Program compiling with the −O3 flag gives even more impressive results: 21.9 MHz square wave. We can conclude that using C, signaling at several MHz speeds should be achievable.

4.4.4 BCM2835 C library

Mike McCauley has made a nice C library called bcm2835 that can also be used to interface with the GPIO pins using C. Its installation was also quite easy: download, run the standard configure \make \make install commands and you‘re good to go. Compiling the code is done with the −l bcm2835 compiler flag to include the library. Benchmark code looked like this (note that in Broadcom numbering, GPIO 4 is P1_ 07):

```
# include <bcm2835.h>
# define PIN RPI_ GPIO_ P1_ 07 \\GPIO 4

    int main(int argc, char  argv[])
if(!bcm2835_ init())
return 1;

    \\Set the pin to be an output
bcm2835_ gpio_ fsel(PIN, BCM2835_ GPIO_ FSEL_ OUTP);

    while(1)  \\Blink
bcm2835_ gpio_ write(PIN, HIGH);
\\delay(500);
bcm2835_ gpio_ write(PIN, LOW);
\\delay(500);


    return 0;
```

The performance is not far beyond the earlier C example: A solid 4.7 MHz which

could be bumped to 5.1 MHz with the use of −O3 optimization flag. Definitely enough for most applications.

4.4.5 C with WiringPi

Gordon Henderson has written an Arduino-like wiringPi library using C. Here‘s the simple test program:

```
# include <wiringPi.h>
# include <stdio.h>
# include <stdlib.h>
# include <stdint.h>
int main()
if (wiringPiSetup () == −1) exit (1) ;
pinMode(7, OUTPUT);

    while(1)
digitalWrite(7, 0);
digitalWrite(7, 1);

return 0 ;
```

With the normal GPIO access method, the library already clocks an impressive 6.9 MHz square wave. The picture below has also been updated for more accurate waveform: There‘s also a GPIO access method which involves calling wiringPiSetupGpio() instead of wiringPiSetup(), and using the normal GPIO numbering instead of wiringPi native renumbering system, so 7 becomes 4 in the above code. The performance is increased slightly to 7.1 MHz. Also, a /proc/sys based access method was

provided, but it was a lot slower, running at 200 kHz on average. The wiringPi also comes with a command gpio that can be used to access the GPIO, but the performance is very poor. The program below achieved a 80 Hz square wave:

```
gpio −g mode 4 out
while true
do
gpio −g write 4 1
gpio −g write 4 0
done
```

Based on these simple benchmarks, One can conclude that shell and Python access to GPIO is enough for any automation tasks. For actual signaling applications, C seems like the only choice.

Chapter 5

Implementation and Result

5.1 Design Phase

There are some parameter which come in design phase like requirement, specification and constrains.

In the design four stepper motor required those are able to control cryogenic probe station. There are two motor required for to and fro motion and two motor for side−wise motion.

To understand flow, 3−D model is created shown in figure 5.1 below

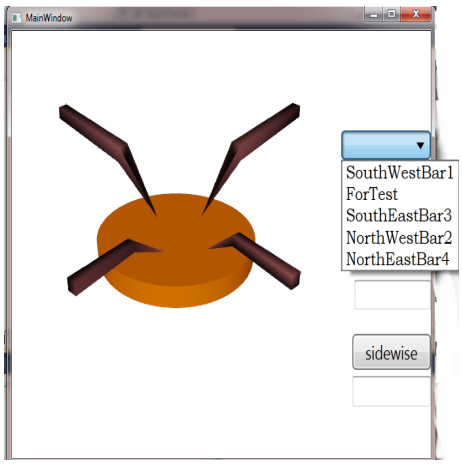

Figure 5.1: 3D presentation Diagram

This 3D presentation according to the flow chart given below. For specification refer D.12 C Cost and Size are two most important parameter. Application for automation should be error free. GPIO pins available To implement an interactive GUI application through Raspbian Operating system is used. Python language is used for creating interactive application.

In this implement process python language is used. In python both the libraries of GUI and GPIO available. To implement GUI tkinter module used. RPi.GPIO module used for controlling GPIO of the board.

5.2 Flow Chart

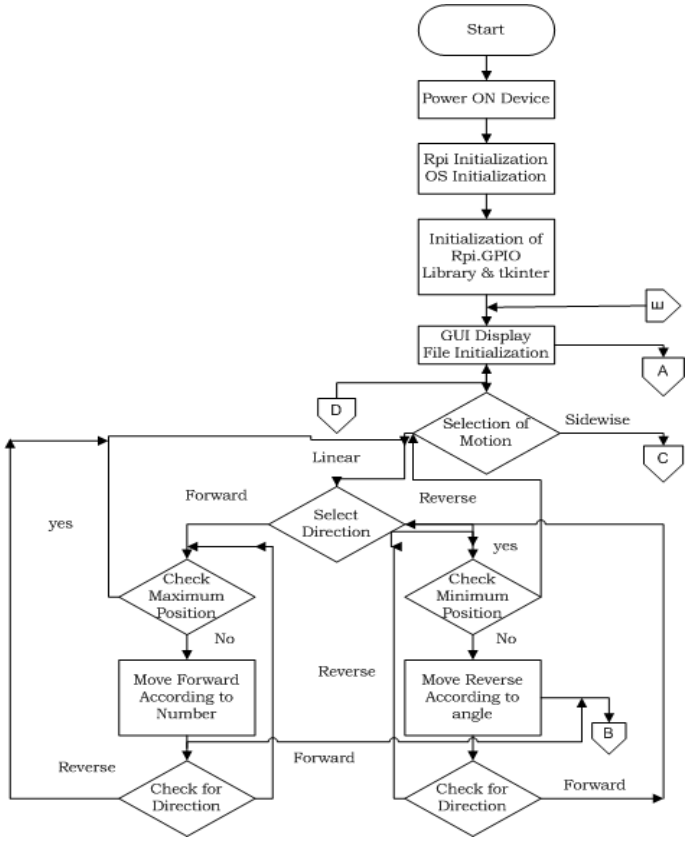

Figure 5.2: Main Program Flow

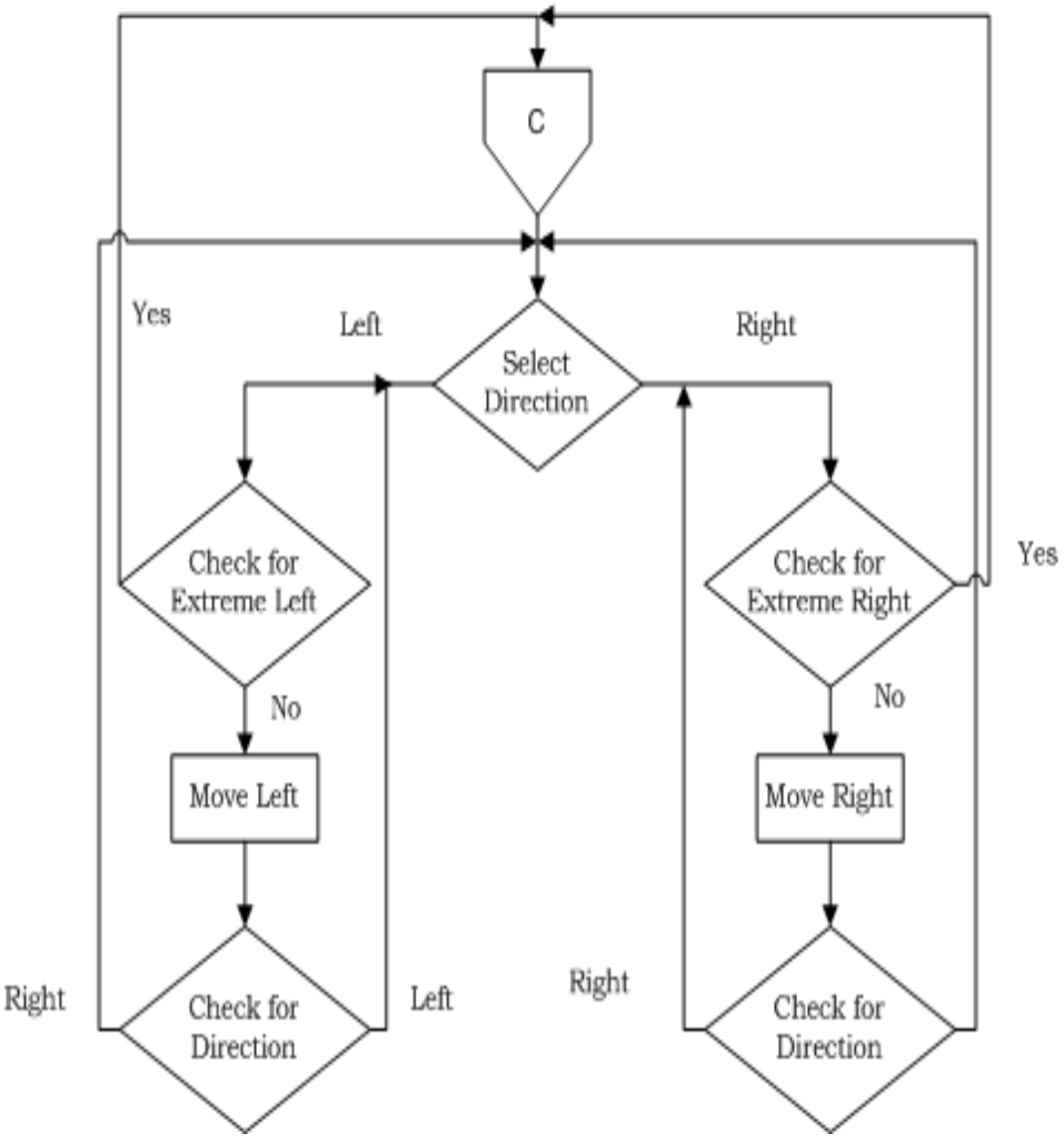

Figure 5.3: Side Wise Motion

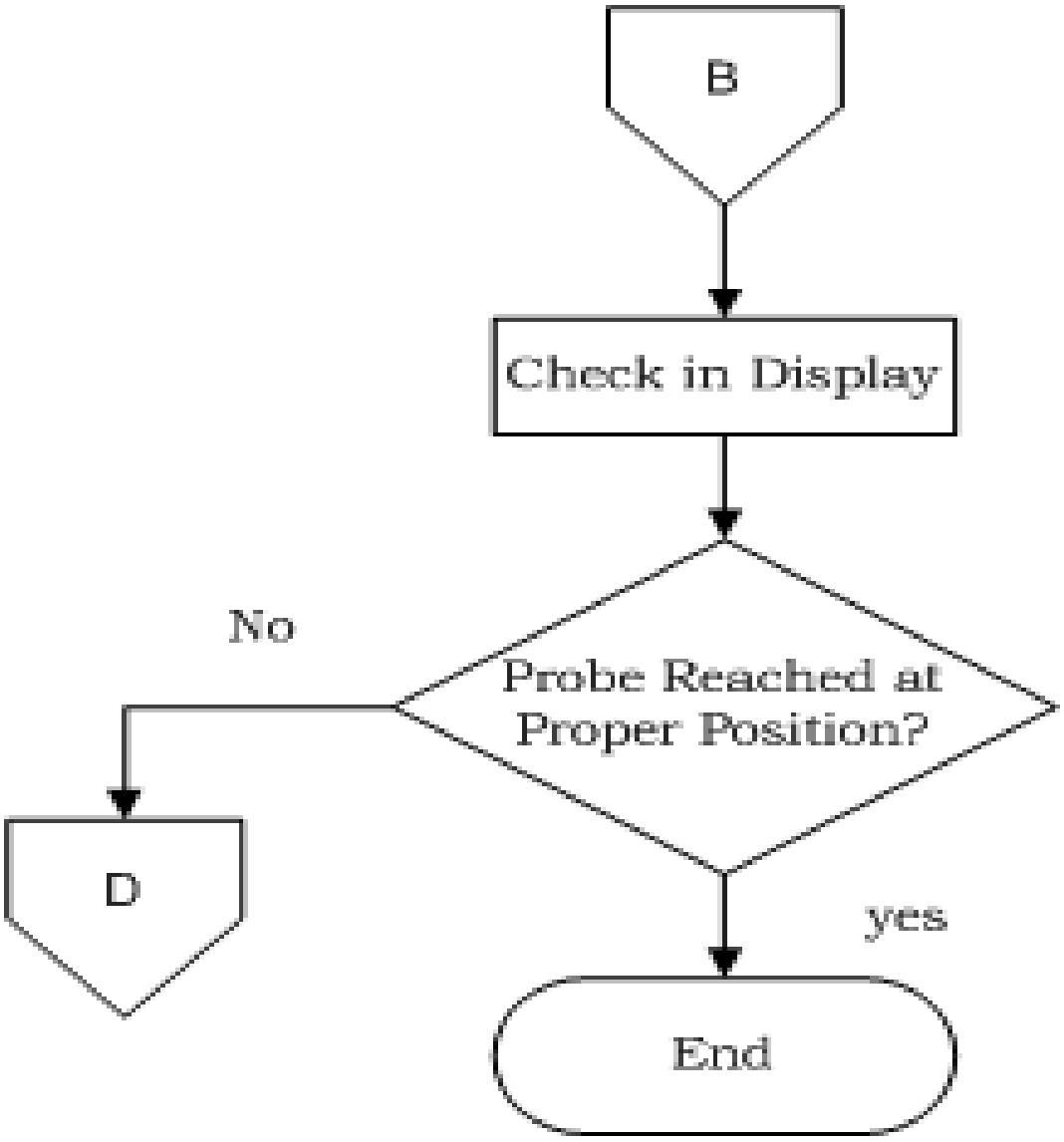

Figure 5.4: Subroutine of Flow Chart

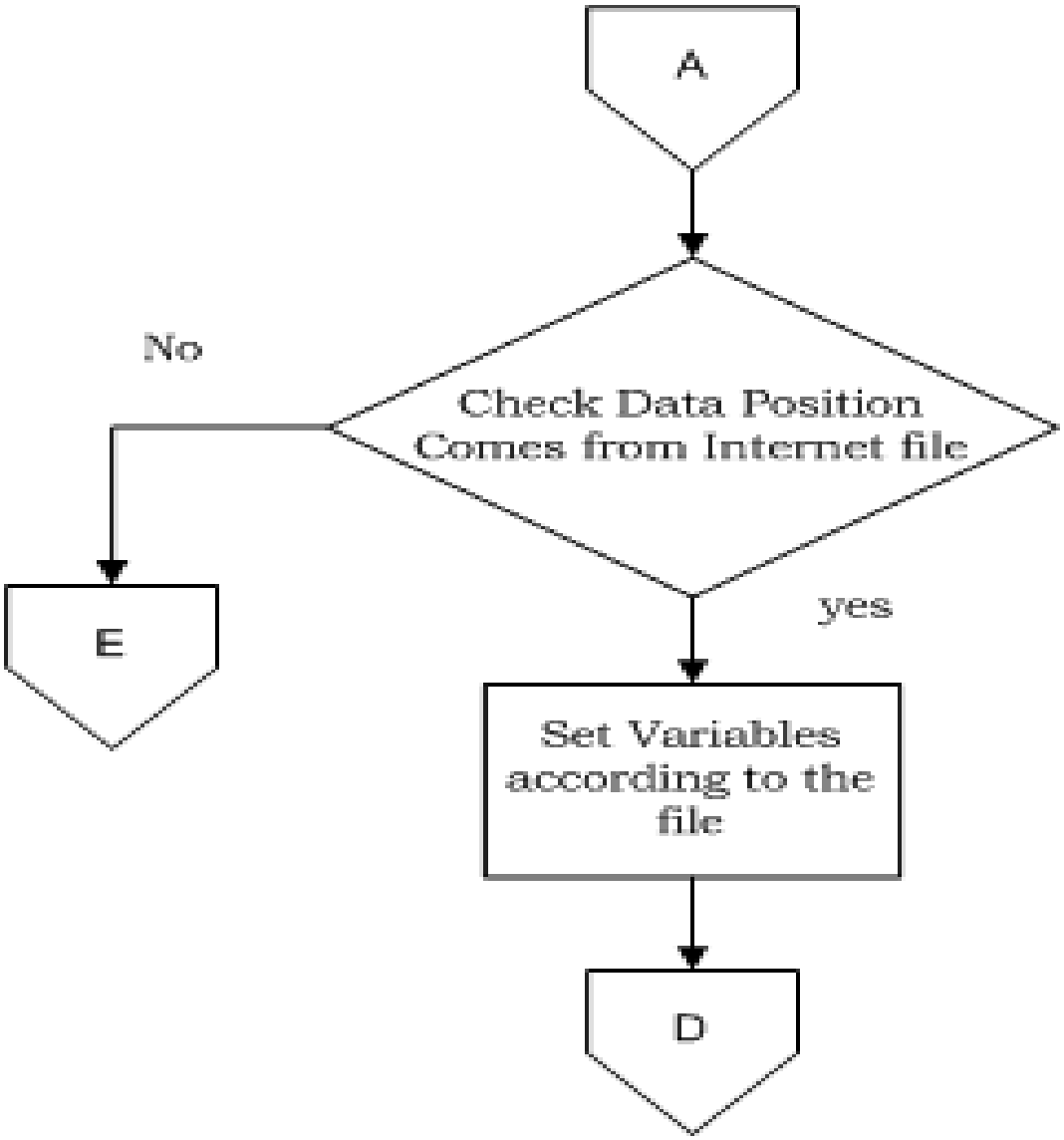

Figure 5.5: Subroutine of Flow Chart

5.3 GUI Application Flow

User case diagram and class diagram for the GUI application. Those diagrams are self explanatory. This application help user to interact with the in a proper way.

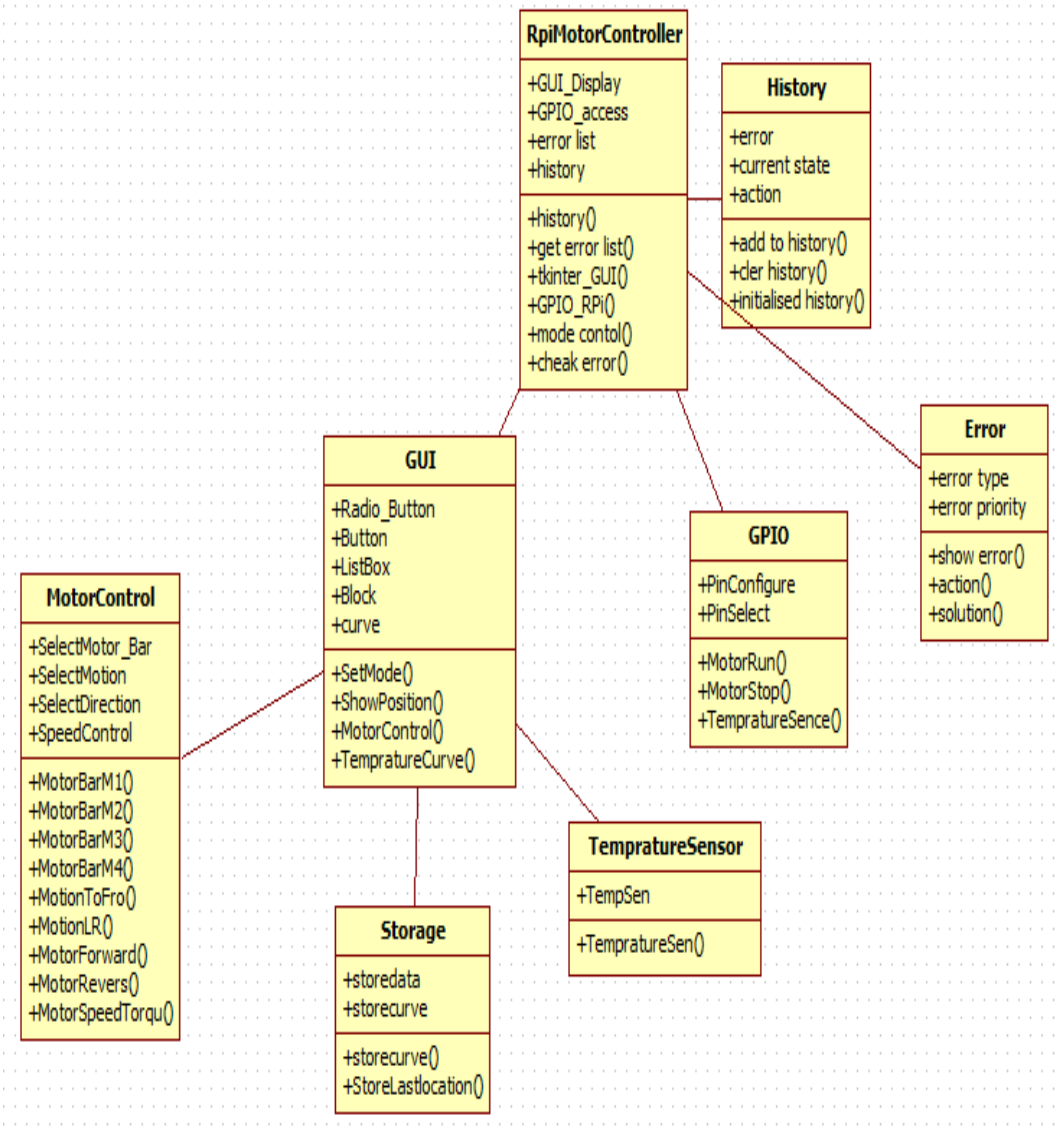

Figure 5.6: Class Diagram For GUI

Figure 5.7: User Case Diagram For GUI Application

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this project entitled as "Automate the Cryoprob By Raspberry Pi", we automate the cryoprobes with the help of RPi and achieve 98% accuracy. Also enable application through remote place with the help of internet web site. Remote Access is achieve and it is secure and reliable. Problem due to sudden power off was overcome by use file operation with save the position and motion related information into the SD card. One more solution for that problem was always start from initial position but it is not profitable for user. Hardware selection as RPi is good choice to make an interactive application, because RPi is a mini computer. It supports Linux OS and many other applications.For creating GUI and accessing GPIOs libraries are available there in Raspbian OS, which provides easiness for the developer to develop an interactive embedded system. But there are very less GPIOs available on board for low level peripheral access, which is a drawback of RPi. For us to overcome this drawback we require another external hardware which can control four stepper motors and a sensor. This external hardware is controlled by many ways like Bus protocol available on RPi or use simple GPIO,it is also alternate way to achieve external hardware control. In the GPIO header pins there is a particular pin for SPI bus through which we can

control other hardware.Here the stepper motors alone can not provide the required resolution(in the order of micro meter)for the cryoprob.so we use a gear system to increase the resolution. Here the combination of RPi and the peripheral hardware provides a strong embedded system to automate the cryoprob .

6.2 Future Work

Future objective is, to create an external hardware with the help of micro-controller. Connect it with RPi and control by SPI bus. External hardware will control the stepper motor and other sensor. Create GUI in RPi which provide interaction with user and this GUI control SPI which control external hardware and external hardware control stepper motor.

Appendix A

Python code for Stepper motor control through GUI

```
import time
import RPi.GPIO as GPIO
import tkinter
from tkinter import
root = Tk()
window = tkinter.Tk()
window.title("Stepper Motor")
lable = Label(window, text="Click here to run motor")
lable.grid(row=0)
lb = Listbox(window)
lb.insert("end", "A list entry")
for item in ["M1","M2","M3","m4"]:
lb.insert("end",item)
GPIO.setmode(GPIO.BCM)
StepPins = [24,25,8,7]
for pin in StepPins:
```

```
print ("Setup pins")
GPIO.setup(pin,GPIO.OUT)
GPIO.output(pin, False)
StepCounter = 0
WaitTime = 0.5
StepCount1 = 4
Seq1 = [(1,0,0,0),(0,1,0,0),(0,0,1,0),(0,0,0,1)]
StepCount2 = 8
Seq2 = [(1,0,0,0),(1,1,0,0),(0,1,0,0),(0,1,1,0),
(0,0,1,0),(0,0,1,1),(0,0,0,1),(1,0,0,1)]
Seq = Seq2
StepCount = StepCount2
class Motor_Run:
StepCounter = 0
WaitTime = 0.5
def __init__(self):
print("It just initiate the class %i" %Motor_ Run.StepCounter)
def Motor_ Move(self):
for pin in range(0, 4):
xpin = StepPins[pin]
if Seq[Motor_ Run.StepCounter][pin]!=0:
print (" Step %i Enable %i " %(Motor_ Run.StepCounter,xpin))
GPIO.output(xpin, True)
else:
GPIO.output(xpin, False)
Motor_ Run.StepCounter +=1
if (Motor_ Run.StepCounter==StepCount):
Motor_ Run.StepCounter = 0
time.sleep(Motor_ Run.WaitTime)
```

```
def MotorRun():
MotorRun = Motor_ Run()
MotorRun.Motor_ Move() def selectBar(event):
global value
widget = event.widget
selection = widget.curselection()
value = widget.get(selection[0])
print("Selection:", selection, ": %s" %value) if (value == "M1"):
print("Motor Bar 1 is selected")
def MotorRun():
MotorRun = Motor_ Run()
MotorRun.Motor_ Move()
MotorRun()
elif (value == "M2"):
print("Motor Bar 2 is selected")
def MotorRun():
print("Code is not prepared for Bar 2")
MotorRun()
elif (value == "M3"):
print("Motor Bar 3 is selected")
def MotorRun():
print("Code is not prepared for Bar 3")
MotorRun()
else:
print("Motor Bar 4 is selected")
def MotorRun():
print("Code is not prepared for Bar 4")
MotorRun()
lb.bind("<Double-Button-1>", selectBar)
```

```
lb.grid(row=1, column=0)
def Motor_ Stop():
for pin in range(0,4):
xpin = StepPins[pin]
GPIO.output(xpin, GPIO.LOW)
print("Motor has stopped")
M=Message(root,text="Motor is Stop")
M.pack()
MotorSwitch = Button(window, text="Motor_ Switch",command=MotorRun)
MotorSwitch.grid(row=0,column=1)
MotorStop = Button(window, text= "Motor_ Stop ",command=Motor_Stop) Mo-
torStop.grid(row=1,column=1)
Quit = Button(window,text="Quit",command=window.quit)
Quit.grid(row=2,column=1)
window.mainloop()
```

Appendix B

Writing an SD Card Image

B.1 Writing an SD card from from Windows

1. Download the Win32DiskImager program.
2. Download the 2013-09-25-wheezy-raspbian.zip image file of Raspbian OS from o http://www.raspberrypi.org/downloads.
3. Download the SDFormatterv4.zip program for format the SD card.
4. Insert the SD card in your reader and note the drive letter that pops up in Windows Explorer.
5. Open Win32DiskImager and select the Raspbian disk image.
6. Select the SD cards drive letter, then click Write. If Win32DiskImager has problems writing to the card, try reformatting it in Windows Explorer.
7. Eject the SD card and put it in your Raspberry Pi.

Note Please make sour SD card is properly formatted. If SD card show 15.9 MB space after format then use SDFormatterv4. SD card must show full space. When SD card formatted then use above mount procedure.

B.2 Writing an SD card from from Linux

1. Open your a new shell and without the card in the reader, type df -h to see which disk volumes are mounted.

2. Now insert the SD card and run df -h again.

3. Look at the list of mounted volumes and determine which one is the SD card by comparing it to the previous output. Find the device name, which should be something like /dev/sdd1. Depending on the configuration of your computer, this name may vary. Write the cards device name down.

4. To write to the card you will have to unmount it first. Unmount it by typing umount /dev/sdd1 (using the device name you got from the previous step instead of /dev/sdd1). If the card fails to unmount, make sure it is not the current working directory in any open shells.

5. Next youll need to figure out the raw device name of the card, which is the device name without the partition number. For example, if your device name was /dev/sdd1, the raw device name is /dev/sdd.

6. It is really important that you get the raw device name correct! (use "dmesg | tail -10" command for check the name) You can overwrite your hard drive and lose data if you start writing to your hard drive instead of the SD card. Use df again to double check before you continue.

7. Make sure that the downloaded image is unzipped and sitting in your home directory. Youll be using the Unix utility dd to copy the image bit by bit to the SD card. Below is the command:

   sudo dd bs=1M if= /2013-09-25-wheezy-raspbian.img of= /dev/sdd

This command tells dd to run as root and copy the input file (if) to the output file (of).

8. It will take a few minutes to copy the whole disk image. Unfortunately dd does not provide any visual feedback, so you will just have to wait. When its done it will show you some statistics. It should be ok to eject the disk, but just to make sure it is safe, run sudo sync, which will flush the filesystem write buffers.

9. Eject the card and insert it in your Raspberry Pi.

Appendix C

Pin configuration and Number

Table C.1: Pin Numbering

wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin
–	–	3.3v	1|2	5v(Power)	–	–
8	R1:0/R2:2	SDA	3|4	5v(Power)	–	–
9	R1:1/R2:3	SCL	5|6	0v(gnd)	–	–
7	4	GPIO7	7|8	TxD	14	15
–	–	0v(gnd)	9|10	RxD	15	16
0	17	GPIO0	11|12	GPIO1	18	1
2	R1:21/R2:27	GPIO2	13|14	0v(gnd)	–	–
3	22	GPIO3	15|16	GPIO4	23	4
–	–	3.3v	17|18	GPIO5	24	5
12	10	MOSI	19|20	0v(gnd)	–	–
13	9	MISO	21|22	GPIO6	25	6
14	11	SCLK	23|24	CE0	8	10
–	–	0v(gnd)	25|26	CE1	7	11

Appendix D

Procedure To Perform Action

To implement an interactive GUI application through Raspbian Operating system is used. Python language is used for creating interactive application. Implementation of the GUI to control GPIO in RPi is as follows:

- Step 1:

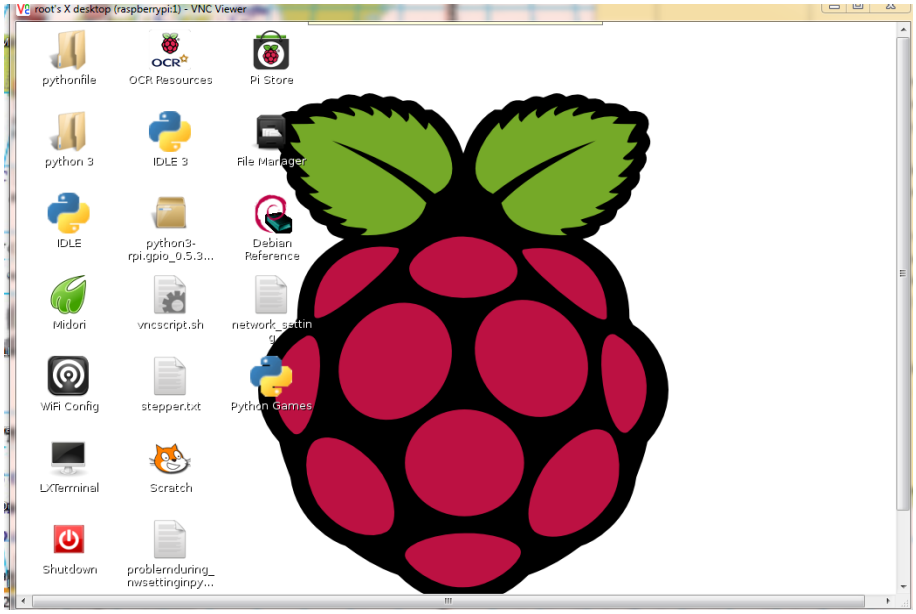

Figure D.1: Desktop Of Raspbian OS

- Step 2:

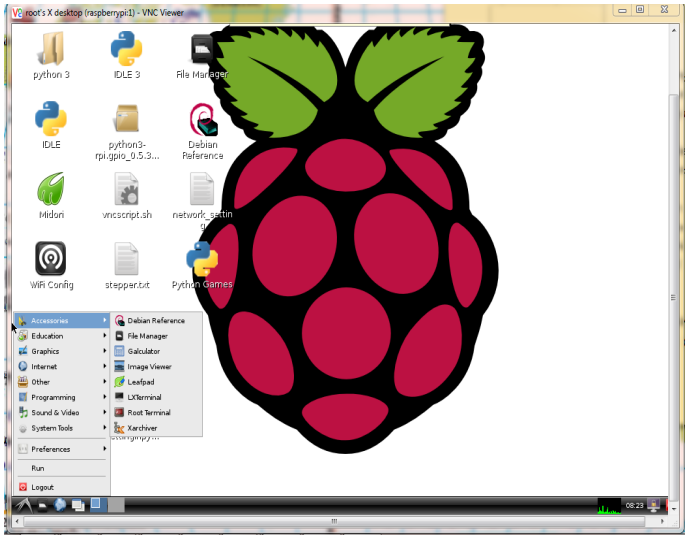

Figure D.2: Applications on OS

- Step 3:

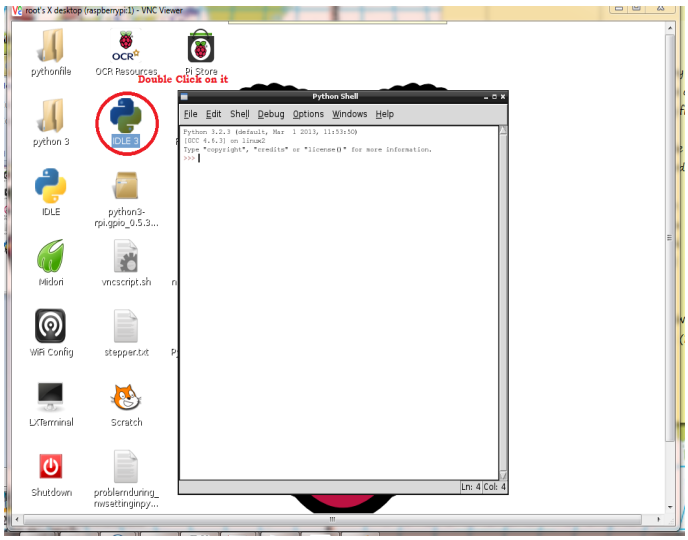

Figure D.3: Python3.2 IDE

- Step 4:

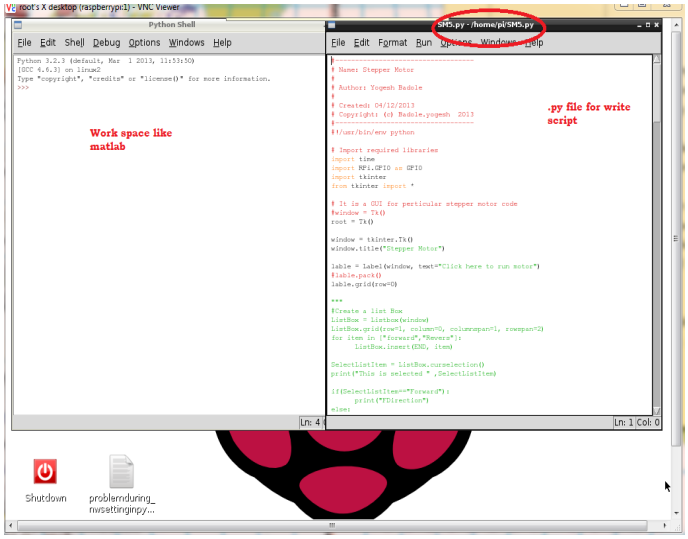

Figure D.4: Python Script

- Step 5:

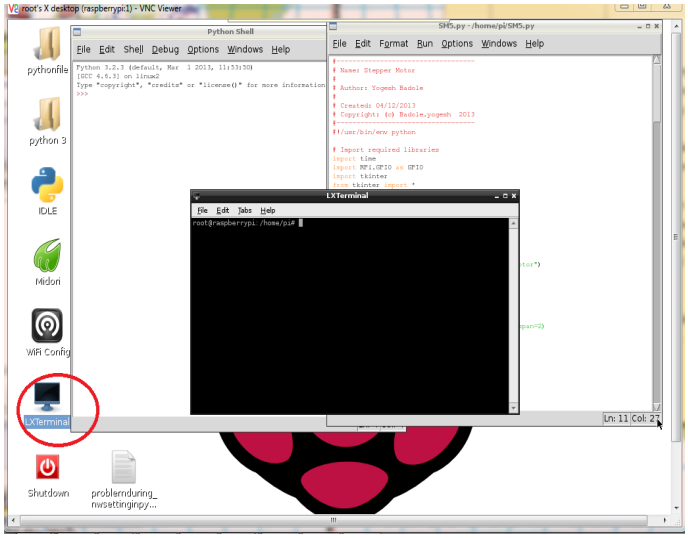

Figure D.5: LXteminal

- Step 6:

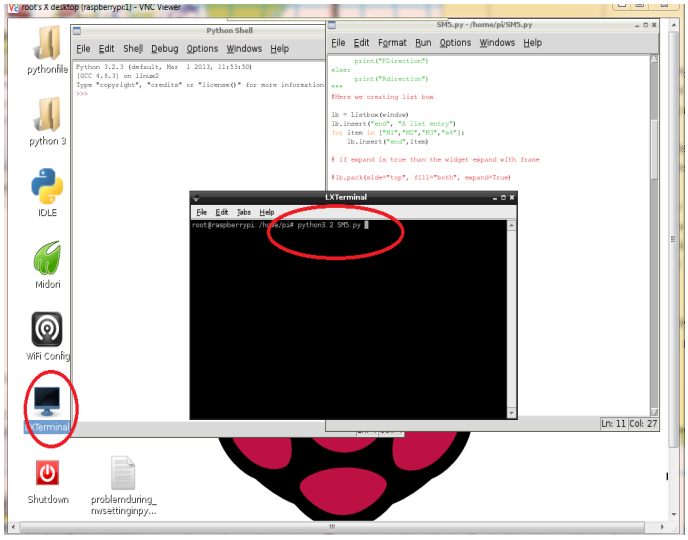

Figure D.6: Compile Python File

- Step 7:

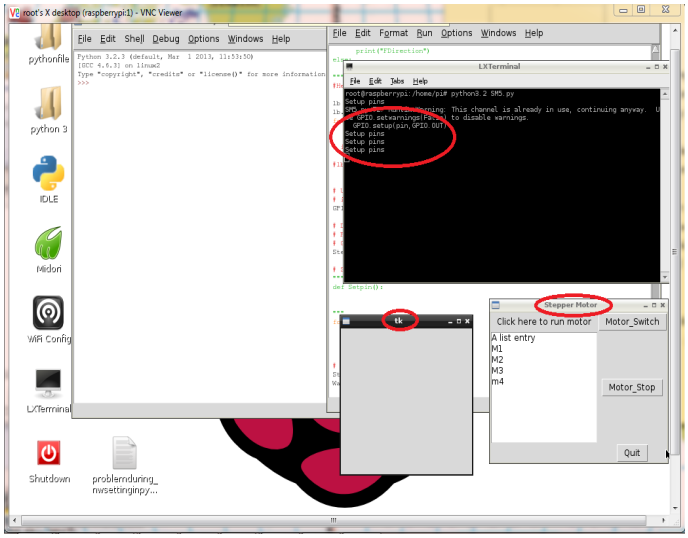

Figure D.7: Display GUI

- Step 8:

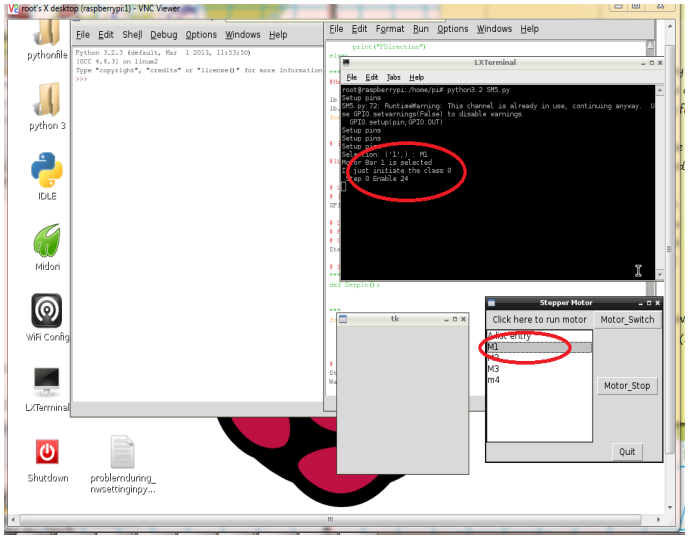

Figure D.8: Select Motor

- Step 9:

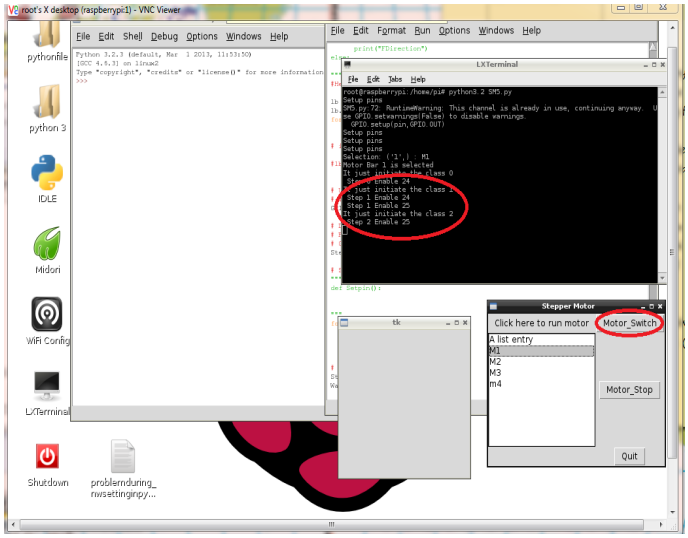

Figure D.9: Run Motor

- Step 10:

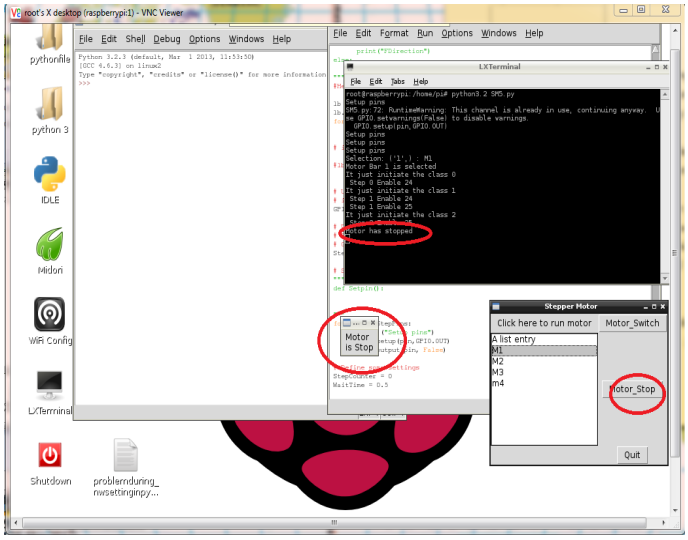

Figure D.10: Motor Stop

- Step 11:

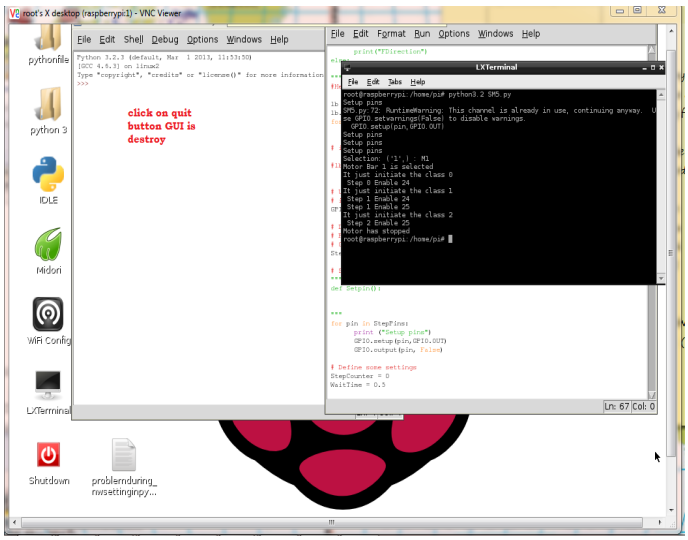

Figure D.11: Click Quit Button

- Step 12:

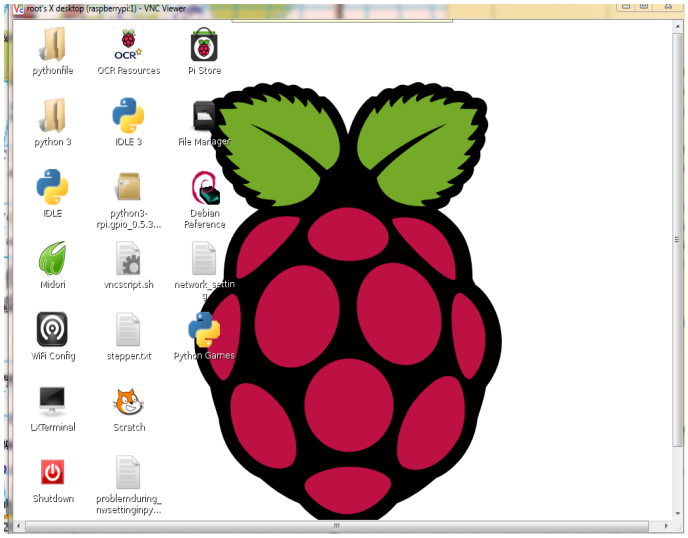

Figure D.12: Desktop Of Raspbian OS

In this implement process python language is used. In python both the libraries of GUI and GPIO available. To implement GUI tkinter module used. RPi.GPIO module used for controlling GPIO of the board.

References

[1] Mayumi Itakura Kamata and Tetsuo Tamai "How Does Requirements Quality Relate to Project Success or Failure? ", in IEEE International Requirements Engineering Conference,Jul. 2007, pp.69-78

[2] Software Engineering Standards Committee "IEEE Recommended Practice for Software Requirements Specications ", in IEEE Std 830-1998,Revision of IEEE Std 830-1993),Reaffirmed 9 December 2009 Approved 25 June 1998

[3] Matt Richardson and Shawn Wallace "Getting Started with Raspberry Pi ", Published by OReilly Media,,Dec. 2012,

[4] "Raspberry Pi Getting Started Guide Raspberry Pi User Manual",RS Components Vsn 1.0 3/2012

[5] "BCM2835-ARM-Peripherals Data Sheet", Broadcom Corporation.,2012

[6] "Cryogenic Micromanipulated Probe Stations from Janis Research", `http://www.janis.com/ProbeStations_Home_KeySupplier.aspx`

[7] "Cryogenic Probe Stations", `http://www.lakeshore.com/products/cryogenic-probe-stations/pages/cryogenic-probe-stations.aspx`.

[8] "The Raspberry Pi Education Manual", http://www.raspbian.org/RaspbianDocumentation

[9] "Debian Raspbian Wheezy operating system", http://downloads.raspberrypi.org/Raspberry_Pi_Education_Manual.pdf

[10] http://www.raspberrypi.org/wp-content/uploads/2012/12/quick-start-guide-v1.1.pdf

[11] http://www.raspberrypi.org/phpBB3/viewforum.php?f=26

[12] http://http://elinux.org/RPi_VerifiedPeripherals

[13] "Raspian Wheezy system image on SD card", http://www.raspberrypi.org/downloads

[14] http://www.raspberrypi.org/archives/tag/hardware

[15] http://www.tech-fruits.com/archives/category/raspberry-pi-hardware.

[16] "RPi Low-level peripherals-eLinux.org", elinux.org/RPi_Low-level_peripherals

[17] "BCM2835 ARM Peripherals - Raspberry Pi", www.raspberrypi.org/wp-content/.../BCM2835-ARM-Peripherals.pdf

[18] "RPi Low-level peripherals-eLinux.org- Raspberry PI Community", www.raspians.com/wp-content/uploads/.../RPi-Low-level-peripherals.pdf