B.E. PROJECT ON

"विज्ञान हथेली पे"

SCIENTIFIC APPLICATIONS USING SINGLE BOARD COMPUTERS

Submitted by:

Anshuman Mishra (032/EC/13)

Gaurav Tyagi (058/EC/13)

Under the guidance of:

Prof. Dhananjay V. Gadre

A Project in partial fulfillment of requirement for the award of

B.E. in

Electronics and Communication Engineering

Division of Electronics and Communication Engineering

NETAJI SUBHAS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF DELHI

NEW DELHI-110078

2017

नेताजी सुभाष प्रौद्योगिकी संस्थान
NETAJI SUBHAS INSTITUTE OF TECHNOLOGY
(An Institution of Govt. of NCT of Delhi-Formerly, Delhi Institute of Technology)
Azad Hind Fauj Marg, Sector-3, Dwarka, New Delhi - 110 078
Telphone : 25099050; Fax : 25099025, 25099022 Website : http://www.nsit.ac.in

CERTIFICATE

This is to certify that the project titled, "**विज्ञान हथेली पे : SCIENTIFIC APPLICATIONS USING SINGLE BOARD COMPUTERS"** submitted by **Anshuman Mishra (032/EC/13)** and **Gaurav Tyagi (058/EC/13)** is a record of bonafide work carried out by them in the Department of Electronics and Communication Engineering, Netaji Subhas Institute of Technology, New Delhi, under my supervision and guidance in partial fulfilment of requirement for the award of the degree of Bachelor of Engineering in Electronics and Communication Engineering, University of Delhi in the academic year 2016-2017.

The results presented in this thesis have not been submitted to any other university in any form for the award of any other degree.

Dhananjay V. Gadre
Associate Professor, ECE Division
Coordinator, Centre for Electronics Design and Technology
Director, TI Centre for Embedded Product Design
Netaji Subhas Institute of Technology
Sector-3, Dwarka. New Delhi 110078
website: http://www.dvgadre.com, http://www.cedtnsit.in
e-mail: dvgadre@nsit.ac.in , dvgadre@gmail.com

`

ACKNOWLEDGEMENTS

Feeling of accomplishment is the best of all emotions that a human being experiences in his life. It is not just the person who wins and succeeds, there are the efforts of the people who made this possible and they deserve every bit of gratitude we can ever express.

The Guru is not just an adjective used for a person, he is an emotion, who holds our hands to guide us through all the storms and springs. He is the one who will pick up the leash to get us back on the track when we tend to lose it. The person who made us learn not just electronics but the righteousness needed in life, whose anecdotes relating electronics to life, act as boosters each time we need one. He who made us a better student and stood like an unbreakable wall, Professor Dhananjay V. Gadre, is not just a mentor for us. From pin to plane, he has always been there, the constant throughout our journey.

To our Parents, whose selfless love and belief on us made us strong to face every situation with grit and might, we are blessed to be your children.

We will also like to extend our gratitude to the people who helped us with the best of their resources and capabilities, teaching us the lessons of brotherhood. Mr. Nikhilesh Prasannakumar's baby weighing scale implementation inspired our 'Talking Weighing Scale" project. Prof. Harish Parthasarathy for authorizing us to use the Telescope and Mr. Vaibhav Choudhary to put in efforts in getting it to us for the "Photometer" Project.

Lastly, To the CEDT Lab and her caretaker, Mr. Vinod Kumar, whose logistical support made every project possible for us, words won't be enough to express the gratitude. Varun's, Rahul's, Akshay's, Riddhi's and Anmol's assistance will never be overlooked whenever and wherever **विज्ञान हथेली पे** will be mentioned.

Anshuman Mishra **Gaurav Tyagi**

032/EC/13 **058/EC/13**

`

ABSTRACT

विज्ञान हथेली पे is an initiative to make scientific applications portable, efficient and affordable using Raspberry Pi 3. This project is based on a framework approach, wherein the designer needs to select the peripherals from the given set. This approach strikes the much required balance between complexity and manageability.

This project is a result of motivation taken up from our own experiences that how can the scientific instrumentation be brought down to grass root level where the users of the instrument can be its creators too. It aims this noble cause, of making people experiment with what they have been trying to do but could not, due to lack of resources gathered at one spot. This implementation can be used as an educational aid to assist kids and students at different levels in what they learn through text books.

विज्ञान हथेली पे makes it much more interactive and supportive to learn the required skills in a short period of time with the onboard peripherals like 5-way joystick, a user switch, a 2.2" Color Graphics display and a LED connected on to a PWM channel of Raspberry Pi. This all with the evergreen compatibility with TI launchpads and maintaining a standard by adopting the Boosterpack standards to create application specific boosterpacks.

It also gives us a platform to explore the Talking paradigm where we can enable our gadgets to speak.

Lastly, we demonstrate that how we used the same approach to converge over multiple scientific applications, justifying this project as what it is really capable of.

`

TABLE OF CONTENTS

`

`

`

LIST OF FIGURES

`

`

LIST OF TABLES

`

CHAPTER 1: INTRODUCTION

Every scientific achievement started off with an idea, an idea which just struck the thinkers, the doers, the makers, the builders and the ones who strived continuously for improving the existing situations. This case is not different.

We always see a discovery as an end result, the equations mentioned in the textbooks, the laws taught to us in the high school, the principles governing the necessities of human life sustaining on this planet, as basic as the movement. But it would have been more interesting to see the journey of the idea, starting from the brains of humans like us, to the paper which we hold in our hands today in the form of books, journals, magazines etc.

विज्ञान हथेली पे is one such idea where the journey is as exciting as the end result itself. The point from where it emanated, first draft of things on the paper, thoughts shooting like asteroids and then disappearing in the friction of atmosphere (read as the feasibilities). How it made us see the present situations from an entirely new perspective, it made us answer all those questions which every engineer must answer when asked about the problems, which he aimed to solve. To make someone appreciate a journey, it is very important to answer specific questions without losing the track.

This Report is an effort to make the reader understand our motivation which kept us moving throughout the Project, justifications of our decisions and how did we made it into a reality.

1.1 THE MOTIVATION

"A battle is lost when the warrior is exhausted, mentally and not physically. It is the mind which loses the battle first, even before the body does."

This quote explains that how psychology is of utmost importance while continuing to do everything, there would be obstructions, undoubtedly, but what makes someone converge what he started are the driving forces. Everyone calls it Inspiration. Even we had our part of forces, to keep us inspired for the causes which need to be solved. This section deals with all the causes which sparked the idea which later on converged as विज्ञान हथेली पे|

1.1.1 PRESENT SCIENTIFIC INSTRUMENTATION

From our early school days, I remember the time when we were introduced to science labs as kids. The first thought that hit me was that why were we made to sit in the classrooms when such facilities existed in the school. This thought faced a harsh reality that if the school authorities made every student sit in such rooms, either the fee will shoot up or the school will go bankrupt. Such rooms, which we were forced to call as LABS, a term which has been associated with delicate and expensive equipment, always kept out of bounds of kids, have never been accepted whole heartedly as a regular curriculum.

`

The niche section of our early school life didn't stop there and we realized that this has got its roots so deep into the education system that the same "do-not-touch" approach towards the Labs exists in the professional colleges as well.

This, was our first hit as engineers. Specialization in Electronics had a role to be played later. The intensity of this problem can be approximated by the fact that the students eventually develop the tendency of not touching anything which has a potential to go wrong. Kids won't even volunteer for changing a damaged electric bulb. Never getting "their hands dirty" while doing things and experimenting. If such situation exists, we cannot expect from the coming generation to have an open mind, daring enough to experiment.

So, what can be a solution? Well, the core problem lies with the cost of present scientific instrumentation, make it so cheap that kids can have a lab setup at their home, the ease of availability and usability.

Moving further, the kid somehow managed to have his interest going on in science and ended up in an engineering college. When we entered into the college, school problem remained same but now a new issue awaited. The experimentation around such sophisticated equipment had a great deal of human intervention, leading to errors and undesired results. But that is something one can ignore, right? No. The attitude towards each and every observation while performing an experiment should be treated as a moral and ethical value for a budding engineer. But turns out that even if these moral values are imparted well into the budding engineers, the scope of human errors still exists, inevitably.

Again, what can be a solution to this inevitable problem?

Alright, as the life progressed and the new engineer ended up into the Industry and when he actually got an opportunity to explore, the whole system is packaged into an envelope, which is out of bounds to him. Also, this instrumentation is mostly made in some other country and we are mere users. Making him rest under the boundaries of possible learning through experimenting and spreading it further.

All this seemed like a great deal but when we started thinking of possible solutions, three terms flashed in front of our eyes: Low-Cost, Computers and Open-Source.

1.1.2 REQUIREMENT OF AN EDUCATIONAL AID

The entire curriculum ranging from schooling to graduation then to post graduation, comes with an unseen envelope around them, called the Textbook learning, which prevents any enthusiast to tweak his ways to new possibilities. Also, think about the time when you thought of controlling the lighting of your home, automatically closing and opening the doors and what not that you imagined to build outside the boundaries of your textbook? True, you may object as the books are the best friends of a student, even we aren't denying this, but having friends who motivate you to play the right shot without giving you the bat shows that something needs to be changed.

We need friends, not just for motivation and talking about the dimensions of the ground, weight of the bat and what shot did Sunil Gavaskar used to play as his favorite, but the

`

ones who can actually take you to the field, give you the bat and allow you to experiment all those nasty imaginative shots you have been thinking all day long.

Question is, how to have a friend who can guide us to the field, give us the bat and let us free to do whatever we wish to?

Adding up to the series of thoughts flashed in the previous section, we saw the need of an aid to present education system.

This friend is the one who will make you learn all those things which a book would have been unable to. After all, smooth seas have never made a good sailor. Reading books and keeping the hands in the pocket will never result in good engineers. Period.

1.1.3 EXPLORING THE TALKING PARADIGM

This section has got a story in itself. The idea of making a device talk started years back, when people in the CEDT tried to develop assistive technologies for visually impaired. I believe that every one of us have come across such sci-fi technologies where an electronic device would be verbally interacting with the users. Who forgets R2-D2 from star-wars? The sweet electronically modulated voices making them appear more robotic then they were really. Kids got a real time fascination by seeing something like Jarvis, an AI assistant to none other than Tony Stark, the Iron Man. But, think when the devices we make start to speak, interact, obey commands in the real life, they would be a boon for the visually impaired, an assistive technology for increasing the efficiency of the workers by giving an audio feedback. Thought?

Fascinating, isn't it?

But, the deal is to develop such an approach which can make this reel-life fascination a reality, hence making us talk about the Talking Paradigm.

We made you see what we saw as issues, addressing these issues formed the central nervous system of our entire journey. Now let us take you to the solution which we see as a one stop shop for everything mentioned in this section.

1.2 PROPOSED SOLUTION

Now, listing the terms which flashed in front of us in the same order, we were looking out for something, which can be cheap, involves 'computers', is open sourced and can be treated as an educational aid. In addition to these, can we utilize it to explore the Talking Paradigm?

We will show you the bigger picture and how does the solution which we are going to propose satisfies all the needs.

`

1.2.1 SCIENTIFIC INSTRUMENTATION USING COMPUTERS

After talking about the limited use of scientific instrumentation since our school days and then the limitless human intervention while performing the experiments, we are in full tempo to present the solution, but before that, we request you to believe, believe that computers are not just the ones kept on the table or the laptop which you are using to read this sentence right now. Computers are everywhere, check out for the one which is making your high-end Casio watch ticking or that 'intelligent lighting system of your car'.

After requesting you to believe it and trying to exploit your present knowledge about the processing capabilities of computers and their ability to replace humans to make a system automated, we started thinking like a normal person. Well, why a normal person? Try having a chat with someone over a cup of tea that how can we speed up a very basic process of handling long queues in front of a bill deposition office.

"Oh! That's simple! Computers can solve this problem, making things online is a sure shot solution!"

This would be the very obvious response of a normal person, but we being engineers cannot stop till making a statement, we will figure out the specifications of the computer required to solve the problem and to what extent, does it need any human assistance or not? Does it need to be a fully automated system or not? If there is a problem then its need to be solved efficiently will be our first priority, putting our knowledge and skill to test and use.

In our case too, 'computers' will be the problem solvers. But there is a problem. Computers don't know the density of the liquid we measured while observing a ball drift down the beaker full of the liquid, the speed of the wave travelling in a slinky, a fork when vibrated makes sound, the length of the day, the speed of earth's rotation, the resistance of your wet hand or that graphite lead pencil lying over there on the table., we won't have stopped here but then we leave it your imagination.

Computer are dumb? Are you thinking that despite their inability to understand all this, how can a computer be the solution to the issues we quoted?

Definitely computers don't understand any of the above mentioned physical attributes and phenomena, then who does? This task of making computers "smarter" and more human-like is done by 2 experts known as Sensor and Analog-to-Digital Converter Tough to understand? No issues. Consider your brain to be a computer which just knows how to deal with numbers. Now, how would you know that your mom is calling you? Let us see this part by part. You will know this by her voice. But, your mom's voice reached you after travelling a medium, the air pressure around your ear changed and it caused the ear drum inside your ear to vibrate, these vibrations were now converted to signals to be sent to your brain. Your brain processed those signals, matched them with the stored ones and when the match was deciphered as your mom's voice, this was the moment when you realized that who is calling you. Imagine, that you lost your ear, would you be able to respond to your mom's call now? Obviously Not.

`

The ear drums acted as sensors and the nerves acted as the Analog-to-Digital converter. One responded to the change in air pressure by vibrating and producing an output which was then converted and transmitted as signals to the brain by nerves. Easy, isn't it? Now, moving on the same lines, for any scientific instrumentation which can be done using computers, one needs to have an application specific sensor and apparatus, connected to an analog-to-digital converter, selected on the basis of various parameters like interval between two successive observations, how many observations to be taken up in parallel, what is the required accuracy and precision of the readings and so on. Point is to make the computer smart with the help of an environment created around it to perform a scientific application.

Becoming technical? Sit back and relax as we shoot you with the harsh reality that why your sweet PC is not the one we have been talking about all this while.

1.2.2 WHY NOT A DESKTOP COMPUTER?

Travel back in time to the same spot where we started, the lab in the school and assume that "Scientific Instrumentation using PC" exited back then. Remember all the experiments which you used to perform there and now try to place a PC besides every apparatus. Imagined? Alright. Move on to the possibility of performing those experiments at home, done? Then, seek for the possibility of asking your friends to have one setup each for themselves, done? What have been your responses to this little imaginative exercise we did? So, it would be positive for first two points, a PC for every experiment can be imagined because it is a school and it can afford it and also you would have a PC at your home, making it good for home based exercises. But, it is not necessary that all of your friends can afford a PC. Although PCs have become relatively cheaper these days but still not cheap enough to be affordable.

Secondly, what about all those experiments you thought of performing out of the boundaries of home and labs, in the ground, under the sun or the night sky. Are you thinking of carrying the bulky PC out there under the sky? By the way PC would need a UPS to run in that situation. Got the point? Portability should be another asset of our proposed solution.

With the advent of computing technology, modern world computers have been compressed to the size of a grain of rice, called embedded computers, the ones which are making your high-end Casio watch ticking or that 'intelligent lighting system of your car'. But, they need not give a Graphical programming interface as that of your PC, neither will they have display driving capabilities as your normal PC does nor the audio output port. Include keyboard and mouse in the list of things to be added for this system to be called a computer according to you. So is there any small, portable, cheap alternative to PC? Yes, there are. They are called Single Board Computers or SBCs in short. The next chapter is all about our decision and parameters based over what we chose one out of all the SBCs present today!

Merging up these points altogether and seeing what we now have as विज्ञान हथेली पे.

`

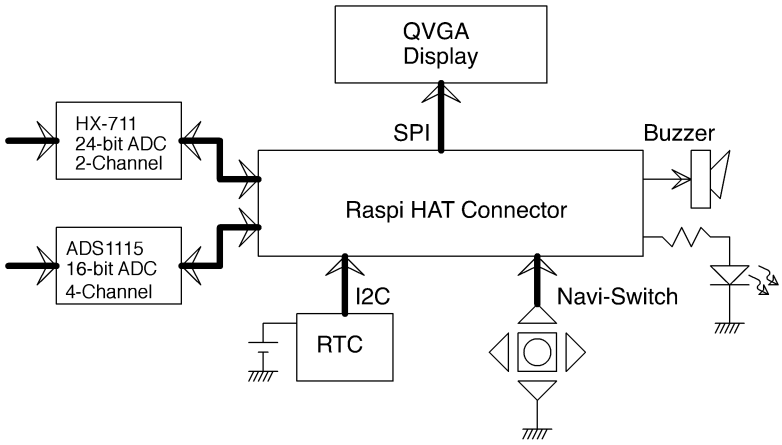


FIGURE 1: BLOCK DIAGRAM OF विज्ञान हथेली पे

1.3 THE JUSTIFICATION

"Our words should justify our actions."

There would have been infinite solutions to the issues we stated but why did we follow the one which resulted into विज्ञान हथेली पे ? Read further to find out.

1.3.1 NEED FOR RAPID PROTOTYPING

Leave electronics for a moment and think about all those points of time in your life when you stepped back due to the lack of resources. Let it be a creative papier Mache idea which was left for another time just because you didn't achieve the right consistency.

Every project, art or craft, electronics or mechanical, has a phase of development. No idea reached a final stage of completion in a day. A factory for producing large quantity of any commodity will not be setup by just having an idea of it. From where it all starts, is something known as prototyping. This is the stage where one can approximate the feasibility of any idea. It is the phase where most of the ideas die. Our intention behind what we proposed was definitely to overcome this die-out phase of any idea, which helps the experimenter to flow through his imagination without waiting for the availability of the apt resources. Since these resources also include the time to learn a particular skill, we had to take into account the beginners who have an idea of a scientific application but not the right amount of skill to even build a prototype.

A Low-Cost Open Source Scientific Instrumentation Framework built around a SBC is what we culminated at.

Framework? What is it?

`

1.3.2 THE FRAMEWORK APPROACH

Mentioning the beginners, the complexity in implementing their ideas can be a demotivating factor for them, so a need of an approach arises which helps them to realize their ideas without getting complicated at any step, even if they are stuck somewhere, the manageability of the approach helps them in getting out of the situation.

In technical terms, it would be a hardware, consisting of finite number of inputs and outputs. When the user is in a learning stage, the limited but ample resources of a framework would make it easier for him to be learn at a faster pace without getting into the complexities at the very initial level.

Justifying the use and need of A Low-Cost Open Source Scientific Instrumentation Framework built around a SBC, it's time that we talk about the basic most model of every electronics system, including विज्ञान हथेली पे which is a 6-block model, simplifying the understanding of the system (to be read as lots of electronic components, flying here and there). Next section talks about the 6 bock model.

1.4 THE SIX BLOCK MODEL

Some things in this world are beautiful enough to explain themselves without speaking a word, this figure being one of them.

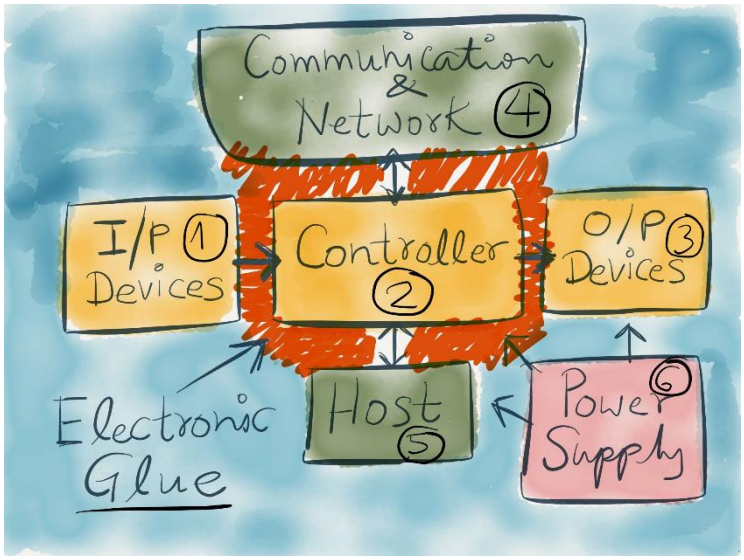


FIGURE 2: THE SIX BLOCK MODEL

We know that the term "host" created doubts for you, for that just understand that a system need not have a single controller, may be that the controller shown in the figure has a 'papa controller', to whom he reports everything. May be there is no host at all but being a generic block diagram, all the maybes have to be accounted. This image shows the atomic part of a system under consideration. This model will be referred throughout the report for circuit explanation and describing each block individually.

`

1.5 FLOW OF REPORT

Defining the flow of a program is as important as declaring and initializing the variables. All the sections of this chapter were not less than declaring the variables. Now, since we move onto the next phase of our journey, the variables will be initialized with explaining the entire flow of command.

That was enough of coding jargon, the next chapter will deal with our decision of choosing Raspberry Pi (I know it sounds tasty) over all others as 'THE' computer to meet our needs. Next we will quickly jump for the lunch where we will meet the chef विज्ञान हथेली पे, the Low-Cost Open Source Scientific Instrumentation Framework, which would help us in tasting various recipes (read as scientific applications, experiments and instrumentation) cooked using Raspberry Pi as the main ingredient.

The subsequent chapters will enable the reader to taste the dishes served readily (read as beginner's experiments using on-board peripherals), understand and learn what we cooked using this approach and lastly, to cook their own recipes, from the comfort of their homes.

Happy Cooking!

`

CHAPTER 2: RASPBERRY PI, WHY?

In the previous chapter, we gave you a peek to those teeny, tiny yet powerful devices known as Single Board Computers (SBCs). What are they? Single Board Computers (SBC) are miniature, typically palm sized devices which contain processors, memories, I/O ports and run an operating system such as Linux. Basically everything you would expect a normal Desktop Computer to have, at a very affordable cost. These computers are designed to be portable and consume very little power, running even from power banks! One of the most important feature that distinguishes SBCs from your run-of-the-mill computers is the provision of GPIOs and other connections provided on the SBCs that allow the user to interface any desirable hardware and program according to one's requirement.

Now you can easily imagine the scenario, where you connect your instruments to one of these devices and head on out to perform experiments!

2.1 SOME SINGLE BOARD COMPUTERS

In the recent times, many types of SBCs from different manufacturers, having their own unique feature sets have been launched, each competing with the others for dominance and popularity. Before we try explaining why we chose Raspberry Pi for विज्ञान हथेली पे, let us tell you a little bit more about three of the more popular SBCs in the market.

2.1.1 BEAGLEBONE BLACK

The Beagleboard Foundation launched a 1GHz AM335x (ARM Cortex A8) based single board computer named Beaglebone Black (BBB). It is a low power, open source platform with 512MB RAM, a NEON Floating point accelerator and two additional 32-bit microcontrollers (PRU) [1]. Priced at about $60 it is a powerful single board computer.

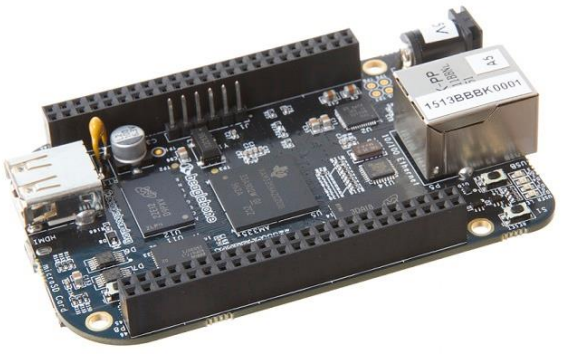


FIGURE 3: BEAGLEBONE BLACK

However, the community and support for Beaglebone Black is relatively small. As a result the platform hasn't been explored and utilized to its full potential. This platform hence, is more suitable for people with experience rather than complete beginners. In addition to this, audio and video support is not as good as contemporary SBCs.

`

	Feature	
Processor	Sitara AM3358BZCZ100 1GHz, 2000 MIPS	
Graphics Engine	SGX530 3D, 20M Polygons/S	
SDRAM Memory	512MB DDR3L 800MHZ	
Onboard Flash	4GB, 8bit Embedded MMC	
PMIC	TPS65217C PMIC regulator and one additional LDO.	
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header	
Power Source	miniUSB USB or DC Jack	5VDC External Via Expansion Header
PCB	3.4" x 2.1"	6 layers
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB	
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS	
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated	
Ethernet	10/100, RJ45	
SD/MMC Connector	microSD , 3.3V	
User Input	Reset Button Boot Button Power Button	
Video Out	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support	
Audio	Via HDMI Interface, Stereo	
Expansion Connectors	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Weight	1.4 oz (39.68 grams)	

TABLE 1: BEAGLEBONE BLACK FEATURES[1]

`

2.1.2 INTEL GALILEO

The Intel Galileo [2] is an Arduino certified development platform based around Intel Quark X1000 CPU, which a single core single thread CPU is running at 400MHz. It is augmented with 256MB RAM and 8MB flash. Intel Galileo is designed to be directly compatible with Arduino shields. Though in itself the Galileo is a single board computer, it is more commonly compared with Arduino platforms.

The support and community of Intel Galileo is growing gradually, however, as pointed out earlier, this board is treated more like an Arduino development platform (a microcontroller based platform) rather than a single board computer.

TABLE 2: SPECIFICATIONS OF INTEL GALILEO GEN 2

Board Dimension	On Board Features of Galileo
Processor	Intel Quark X1000-SC
Description	X86 based, low power for IoT
Speed	400MHz
Width	32-bit
Real Time Clock	Yes, needs a 3.3v coin cell
Cache	16KB L1 Cahe
RAM	512KB, 256 SRAM and SDRAM
Flash Memory	8MB NOR Flash
EEPROM	11KB
GPU	No
Video Support	No
Compatibility	Arduino Shields

`

2.1.3 RASPBERRY PI 3

Raspberry Pi [3] series is perhaps the most well-known series of single board computers. Launched back in 2012 by UK based Raspberry Pi foundation, this is a computer that took the world by a storm. The latest in the series is the Raspberry Pi 3, boasting a 1.2GHz quad-core ARMv8 CPU coupled with 1GB RAM and a Video Core IV 3D graphics core. It also has on-board 802.11n WLAN, Bluetooth 4.1 and BLE support. The board provides 4 USB ports, HDMI port, a stereo 3.5mm audio jack, CSI and DSI interfaces. All of this is packaged into a small credit card sized board priced at about $30. The Raspberry Pi is backed by a huge and talented open source community support which rather simplifies the learning curve for the SBC and hence enables users to utilize it better. Since its launch, it has percolated to the masses and it is greatly praised for its ease of use and operation.

FIGURE 4: RASPBERRY PI 3

TABLE 3: A COMPARISON BETWEEN VARIOUS MODELS OF RASPBERRY PI

	Raspberry Pi 3 Model B	Raspberry Pi Zero	Raspberry Pi 2 Model B	Raspberry Pi Model B+
Introduction Date	2/29/2016	11/25/2015	2/2/2015	7/14/2014
SoC	BCM2837	BCM2835	BCM2836	BCM2835
CPU	Quad Cortex A53 @ 1.2GHz	ARM11 @ 1GHz	Quad Cortex A7 @ 900MHz	ARM11 @ 700MHz
Instruction set	ARMv8-A	ARMv6	ARMv7-A	ARMv6
GPU	400MHz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV
RAM	1GB SDRAM	512 MB SDRAM	1GB SDRAM	512MB SDRAM
Storage	micro-SD	micro-SD	micro-SD	micro-SD
Ethernet	10/100	none	10/100	10/100
Wireless	802.11n / Bluetooth 4.0	none	none	none
Video Output	HDMI / Composite	HDMI / Composite	HDMI / Composite	HDMI / Composite
Audio Output	HDMI / Headphone	HDMI	HDMI / Headphone	HDMI / Headphone
GPIO	40	40	40	40
Price	$35	$5	$35	$35

`

2.2 OPEN SOURCE COMMUNITY

As you might have noticed, while introducing various SBCs, we drew special attention to its community support. Why did we do that? What is the importance of the open source community?

You are an experimenter performing one experiment according to your own understanding and experience. There will be several similar experimenters around the world performing the same experiment, each having their own methods, understanding and experience. The open source community helps bring together such isolated experiences around the world, facilitating information exchange and ultimately improving the overall experiences of everyone involved. This leads to a much reliable result.

An SBC is not a simple microcontroller or a microprocessor, it is a full-fledged system running an operating system (usually a version of Linux). So exploring it completely is not easy, especially for a person who is just starting out. An open source community comes in handy, to debug, help and improve the application as well as the knowledge of the user. A person can face a variety of problems and issues, related to hardware, operating system or the application software, each of which requires a different kind of expertise. A good community support, in such a case, becomes vital for the success of the project. One can definitely argue that all the issues can be resolved without community support, but the sheer amount of hard work and time required for that makes it unfeasible in most scenarios.

Another important facet of open source community is the freedom it provides. The open source projects are free to use, modify and distribute, which allows people to use snippets of existing projects to improve their own. This helps in rapid development of large and well-designed applications with comparatively lesser effort.

2.3 PROTOTYPING

Now that we are somewhat familiar with Single Board Computers and what they offer, let us explore the process of prototyping using these devices. Any experiment you set out to perform or any problem you set out to solve, the first thing that you need is a prototype. You need to be able to know whether what you think is indeed in the right direction or not. Many a times, what we study in the books, and what actually happens in the real world are two very different things. Theoretical analysis, to simplify mathematics, ignores a lot of factors which affect the results in the real world. Thus, what one may expect as the result of an experiment may not exactly be the same when it is done practically. In such a scenario, a working prototype of the desired experiment becomes necessary to verify the concept before actually replicating it on a large scale. Effective prototyping involves taking care of certain important aspects.

`

2.3.1 RAPIDITY OF THE SOLUTION

The purpose of a prototype is to establish that the proposed idea works according to the design and has the ability to solve the problem, basically providing a proof of concept. Thus, it won't do if the development of prototype itself takes a lot of time. The longer it takes to achieve a proof of concept, the longer it takes for a final solution to be developed. Thus, one should be able to develop a prototype rapidly.

The use of single board computers in such a case is quite helpful. Any solution one proposes can have a hardware aspect and a software aspect. The software aspect is eased out in case of SBC primarily because the SBC would have prepared libraries for most kinds of operations, thus the user would only need to arrange the routines specific to his applications and would need to himself write only special kind of routines, accelerating the development of the software for the prototype. As for the hardware aspect, any solution would require certain kinds of input devices, output devices and communication channels. In most SBCs, these kind of channels have already been provided and interfaced with. Thus, the user need not get involved in the intricacies of interfaces such as HDMI, USB, displays, etc. You just need to focus on interface the experiment specific hardware, thus accelerating the hardware development of the prototype.

2.3.2 SCALABILITY OF THE SOLUTION

A prototype by itself does not solve a problem. It just hints at the correctness of the method to solve the problem. Suppose, for an experiment, you find out that the prototype can be developed within a week, but it requires materials that are not easily available or the development process itself is too intricate. Would you consider the prototype to be a good solution? No! Simply because it cannot be replicated easily and hence is not scalable. Scalability of a solution is important. The experiment you perform or the problem you solve is not unique to you. A lot of people across the world would be facing the exact same problem or performing the same experiment, all under varied conditions. In such a case, your solution would be irrelevant if it cannot be adopted by the others as well. Hence, it is important to ensure that the solution you come up with is not only quick to implement, but is also easy to replicate and scale up.

Introducing single board computers in the scenario helps in making the prototype scalable by default. How? The prototype you develop would be using standard libraries and custom software would be minimum. In addition, the custom hardware added to the platform would be according to the bare minimum, which the user would have to take care of to make sure it is scalable. Most of the system would be standard and hence scalable, reducing the load of making the entire thing scalable from the user.

`

2.3.3 OPTIMIZATION

Once you have a working prototype ready, proving that the solution you propose indeed works, can be rapidly developed and is scalable, your focus should now be to improving your solution, optimizing its performance. While single board computers are quite helpful in rapidly developing good prototypes, rarely does the experiment demand full performance from it. So, sometimes the usage of a SBC in certain applications becomes an overkill. At the same time, in certain applications, the processing requirements as so high that relying on the SBC for data acquisition as well as processing makes the solution somewhat inefficient. In such cases, it becomes necessary to break up the prototypes into smaller blocks and bring in additional hardware/controllers to handle those blocks efficiently, thereby making the overall solution much more efficient.

2.4 EASE OF PROTOTYPING USING RASPBERRY PI 3

Now that we have an idea how to approach a problem, Let us now focus on our choice of Raspberry Pi 3 as the SBC for **विज्ञान हथेली पे.** As mentioned earlier, Raspberry Pi 3 has one of the largest and most advanced open source community in the world. As a result, the software and hardware aspects have been explored in-depth and well documented. Due to the active community support, most of the hardware interfacing could be done making sure the software was largely standardized. In addition, active support from experts was available at all times, be it writing the software or kernel level drivers. Using active kernel level drivers make sure that most of the hardware interaction remained as optimized as possible by eliminating the intervention of operating system. As would be evident in later chapters, our applications required GUI and audio applications for which, compared to other SBCs, Raspberry Pi 3 has more extensive support. The presence of WLAN and BLE on-board is an added advantage in terms of improving the prototype in terms of connectivity.

2.5 LIMITATIONS OF RASPBERRY PI 3

Despite the popularity and abilities highlighted above, Raspberry Pi 3 does suffer from certain limitations. For starters, it does not have an Analog-to-Digital Converter to interface with the physical world. This forces the user to interface an external ADC according to requirements.

The Raspberry Pi 3 it is a multitasking environment, which prevents it from performing really high speed applications in a straightforward fashion. The sheer overhead caused by its operating system limit its ability to interface with really high speed ADCs which might be required in certain applications. For a practical example, the highest supported clock frequency of the Serial Peripheral Interface (SPI) is 128MHz, which if interfaced with an 8-bit ADC means we can only have ADC speeds up to 16MSPS, ignoring other

`

factors such as OS overheads, task scheduling overheads, etc. In addition, as in our case, if the application has a graphical interface, one needs to develop multithreaded applications to use the resources efficiently, thus, increasing the complexity of the code.

In order to overcome some issues related to operating system overheads, one needs to write kernel level modules to interface with the hardware, which would result in a non-standardized prototype and a complex software. An alternative approach, which is used in applications usually involving a lot of post-acquisition data processing, is to interface a high speed auxiliary controller, such as a DSP (if hardware multipliers, high speed sampling, etc. are required) with the SBC. The auxiliary platform would be responsible for data acquisition and transfer the data to the SBC using a high speed communication bus or DMA, whichever is suitable, while the SBC itself would just be responsible for the software interface, processing of the data, generation and display of the results. This approach is quite efficient when implemented correctly. विज्ञान हथेली पे is designed to be compatible with this approach.

`

CHAPTER 3: विज्ञान हथेली पे, AN IDEA TO REALITY

Thank you for your patience till now, for understanding the issues, the solution which we proposed, its justifications and our decision to take Raspberry Pi as the main ingredient of whatever dish we would be preparing in the next few chapters of this report. This chapter is all about the chef, the enabler, विज्ञान हथेली पे, who enabled us to proceed with the novel thought of experimenting with new recipes (read as building the scientific instruments from scratch).

As discussed in Chapter 1 and justified in Chapter 2, all our efforts lead to A Low-Cost Open-Source Scientific Instrumentation Framework using Raspberry Pi., one had to ask that how we created the environment around Raspberry Pi with all the features fitting in the 6 block model of विज्ञान हथेली पे. Hence, we dedicate this chapter to make the reader aware about how and why is विज्ञान हथेली पे as the way it looks, in its final form.

3.1 CONCEPTUALIZATION

It is crystal clear by now that the computer had to be smarter with the assistance of sensors and Analog-to-digital converters. But, do you really think that all the applications about which we talked previously can be accommodated on a single board? Also, were you thinking that the same set of sensor and ADC can meet the requirements of a diverse range of scientific applications? Thought so? Alright, I get that you have been thinking in this way only.

Now, try to see it from a different perspective. Imagine that you only had ears as the sensory organs, would you be able to smell, sense and see? This rests the notion that only one set of sensor and ADC cannot resolve all the issues. Secondly, assume that someone asks you to hit a nail, your first step in the flow of actions will be to find a hammer. See, there. You tried to find the right tool to do the task because you knew that you would end up hurting yourself if you would have used your hand to hit the nail. God gave hands, for us to hold the hammer and the body to perform some very basic tasks. We find the tools to assist us in doing complex tasks which are out of bounds of the human body. Right? In the same way, if we need to address a diverse range of applications, our solution should be like the human body, where I can use it as the central part adding different set of Sensors and ADCs according to the application.

Oh! So different module for each one them? Yes!

3.2 MODULARITY: THE BOOSTERPACK ADAPTATION

Creation of stackable modules which can be mounted on our system with Raspberry Pi being the heart and soul of विज्ञान हथेली पे needs to be standardized so as to maintain a uniformity across all the modules. Secondly, as discussed in chapter 2, our computer Raspberry Pi has its own USPs but for specific applications we need to bring DSPs and equivalent controllers into the scene. So, the modularity we have been talking about

needs to have a compatibility with a wide variety of DSP/microcontroller development kits. Kits because of our aim to keep the solution as user friendly as possible in terms of rapid prototyping. Since, TI offers a range of microcontroller/DSP development boards with tremendous support in terms of software development, maintaining the compatibility of all the modules with these development kits seemed to be an obvious choice. Also, we, the CEDT peeps, have played with these toys a lot made it a final choice for us to keep them a standard.

3.2.1 TI'S BOOSTERPACK STANDARD

TI calls their development boards as 'launchpads'. Going by the same approach as we discussed in the previous section, TI has created multiple boosterpacks [4] to be stacked on to these launchpads, which in turn increase the rapidity of prototyping a project.

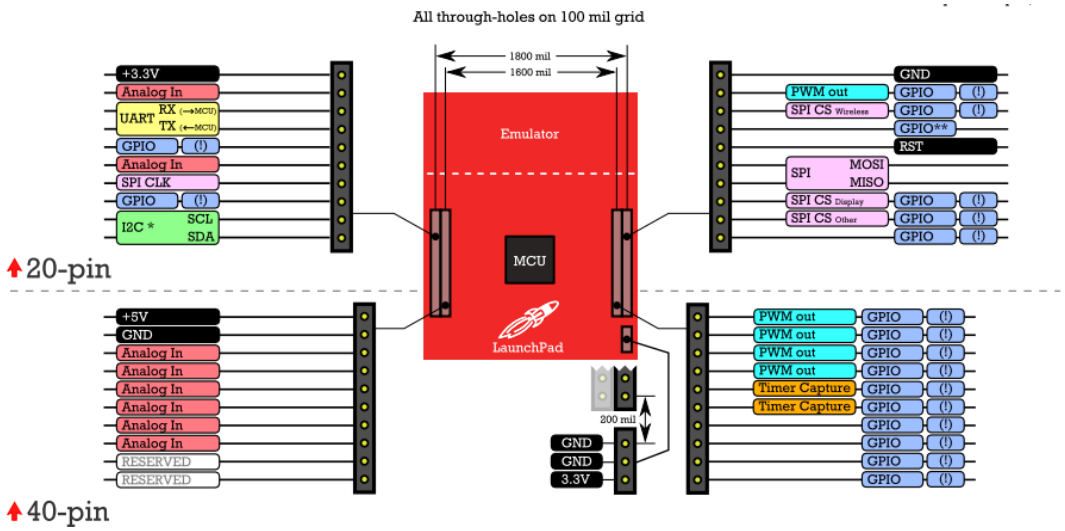


FIGURE 5: BOOSTERPACK PINOUT STANDARD

Fig 3.1 shows that how TI maintains a cross compatibility across all the boosterpacks it makes. We took advantage of this cross compatibility of the boosterpacks and since we have taken into consideration all those possibilities where the Raspberry Pi will be assisted by such development boards, we decided to design all the 'modules' as boosterpacks.

Given the current scenario where we know that how the modules will be developed, it is high time that we should now directly jump on to the 6 block model of विज्ञान हथेली पे . This description will be followed by schematic designing and layout using EAGLE CAD wherein we discuss that how we mapped and connected the pins of Raspberry Pi to various onboard peripherals and the modules, all this while remembering the cross-compatibility constraint.

`

3.3 CONSTITUENT BLOCKS

The 6 block model shows that there will be an input block, an output block, communication block, controller block, host and power.

Since the controller block is Raspberry Pi, we need to limit the number of input output peripherals on विज्ञान हथेली पे framework. This can be easily quantified by analyzing the available GPIOs from a Raspberry Pi. Fig 3.2 shows the pin map of a Raspberry Pi 3 model B.

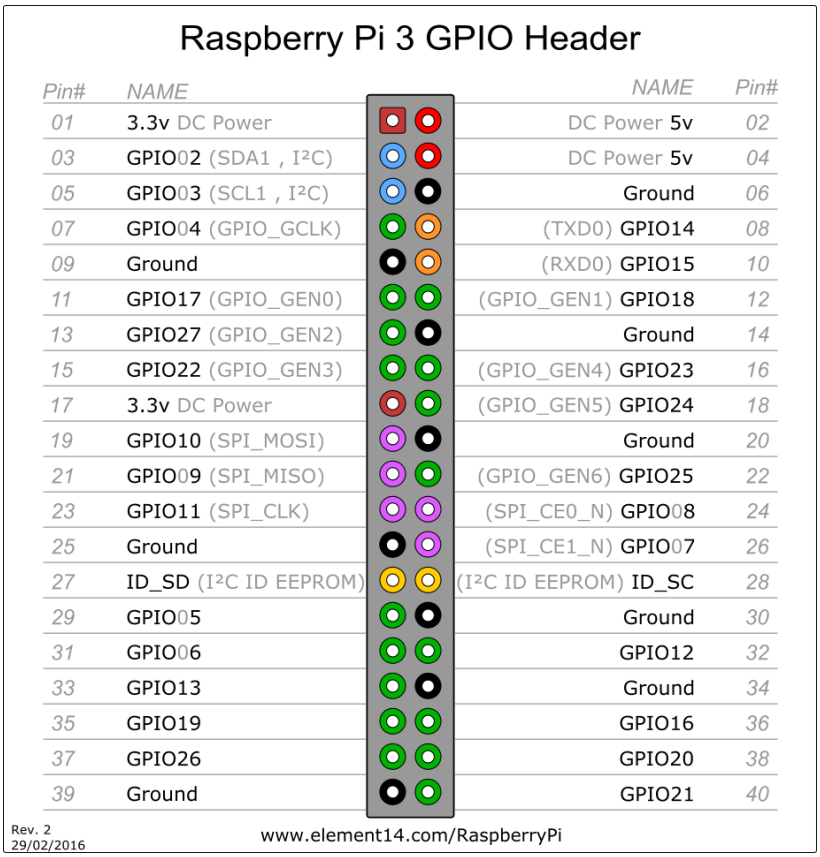


FIGURE 6: RASPBERRY PI 3 MODEL B PINMAP[5]

We have 26 available GPIOs out of which 9 GPIOs are dedicated for communication protocols (SPI, I2C and UART). Although these 9 pins won't be used as GPIOs but you would appreciate after few sections down the line that how these pins were used for dedicated purposes. Hence, we ended up with 17 GPIOs to be assigned to various input output peripherals.

This made us tackle a tradeoff between the numbers of on board inputs to the number of on board outputs.

`

Well, always remembering the motivation behind starting up this project, we assumed a student using the final version of विज्ञान हथेली पे . He has to get started with the cooking but even before that he must have a good taste of the dishes we have made for him already. This means that even before the student start experimenting with the modules, he must have a skill set to be developed to understand that how a Raspberry Pi works. So, few basic input peripherals like switches and some generic output peripherals like a display, LEDs etc. would be more than enough to provide him with the much required jumpstart in the world of single board computers.

So now we define the connection of these peripherals to the Raspberry Pi.

3.3.1 OUTPUT

Raspberry Pi itself has got few tiny onboard LEDs which you can see blinking when it is powered up. But, these aren't sufficient as output peripherals for us. What we were looking for was much more interactive and informative, which exploits the basic hardware features of Raspberry Pi from an embedded engineer point of view.

3.3.1.1 DISPLAYS

First in the list are the displays. These devices are very much necessary when I need my system to provide a valuable visual feedback. This makes the system much more interactive and one can see the processed inputs in a much more decipherable format as compared to using mere LEDs to do so. We needed a display which can be stacked onto the board and becomes an integral part of विज्ञान हथेली पे, thus our search terminated on this beautiful piece which goes by the specifications: 2.2 inches colored Graphic LCD or simply 2.2" GLCD [6].

FIGURE 7: 2.2" ILI9341 BASED GLCD

Since we need to see that how many GPIOs are needed to interface such a LCD with the Raspberry Pi, let us refer to the connections.

`

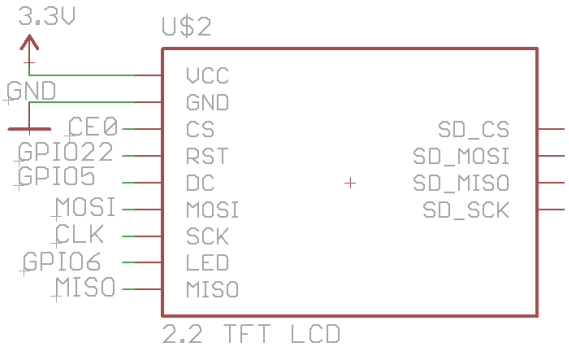


FIGURE 8: CIRCUIT SYMBOL OF A 2.2" GLCD

The connections visible on the right are for interfacing a SD Card and since we don't need a SD card for our application, we left them unconnected.

Focusing on the required connections, it can be clearly understood from the datasheet of this LCD that any controller can talk to it using SPI protocol. The pin functions are as follows:

- SDO: Serial clock output
- LED: 3.3V Power Supply pin for backlight, can combine with VDD pin.
- SCL: Serial clock input
- SDA / SDI: Serial data input
- D/C: Data / Command selection
- RST: Reset, Low level active
- CS: Chip Selection, Low level active
- GND: Ground
- VDD33: 3.3V Power Supply max, regulated 3.3V required

So, the protocol requirements are met by connecting the corresponding protocol pins of Raspberry Pi and this GLCD (remember that the protocol communication pins were no different from the GPIOs). Also, RST, D/C and LED pins need to be controlled via GPIOs. Hence, we are left with 14 GPIOs for the rest of peripherals.

3.3.1.2 AUDIO

For making the system speak while exploring the talking paradigm, our system needs to have Audio Output as one of its feature. Since the Raspberry Pi boasts an Audio out capability, we need not add anything on the hardware designing part as far as the Audio output is concerned.

`

FIGURE 9: AUDIO OUTPUT PORT ON A RASPBERRY PI 3

All the efforts in this segment lie in the software domain. This segment will be dealt in the 5th chapter, where we discuss about the talking paradigm and we explored it using विज्ञान हथेली पे.

The reader must note that Raspberry Pi having Audio output feature never meant that it has a speaker too. We need to connect an external speaker so as to utilize this feature of our main ingredient.

There exists one more output feature on the Raspberry Pi, this time for a driving an external HDMI screen.

3.3.1.3 HDMI

HDMI (High-Definition Multimedia Interface)[7] is a proprietary audio/video interface for transmitting uncompressed video data and compressed or uncompressed digital audio data from a HDMI-compliant source device, such as

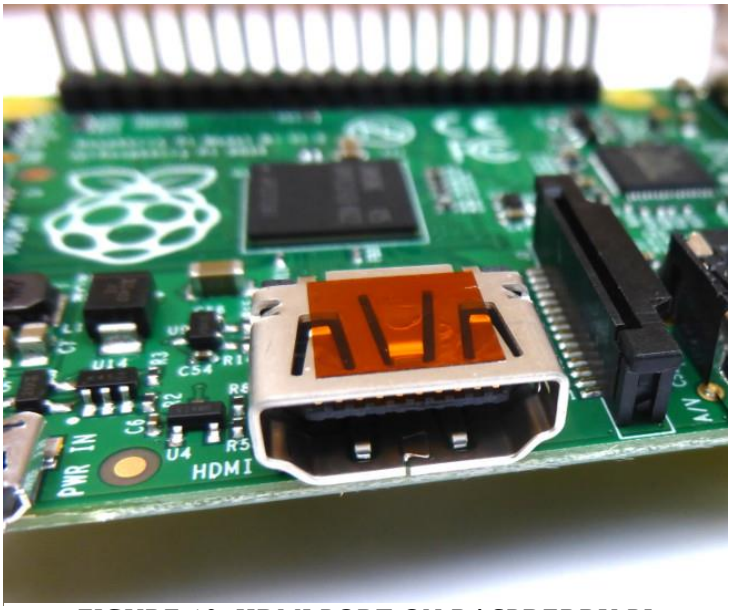


FIGURE 10: HDMI PORT ON RASPBERRY PI

`

a display controller, to a compatible computer monitor, video projector, digital television, or digital audio device. HDMI is a digital replacement for analog video standards.

This would be utilised in specific applications where we need to display the result on bigger screens. HDMI plays a significant role there. As similar to the Audio output port, we need not add any other additional peripheral in the hardware for making it HDMI compatible.

FIGURE 11: A SAMPLE HDMI SCREEN COMPATIBLE WITH RASPBERRY PI

Before concluding the output segment, it is highly appreciable to understand a basic concept, which is dearest to any embedded system designer, the Pulse Width Modulation.

3.3.1.4 PWM LED

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

`

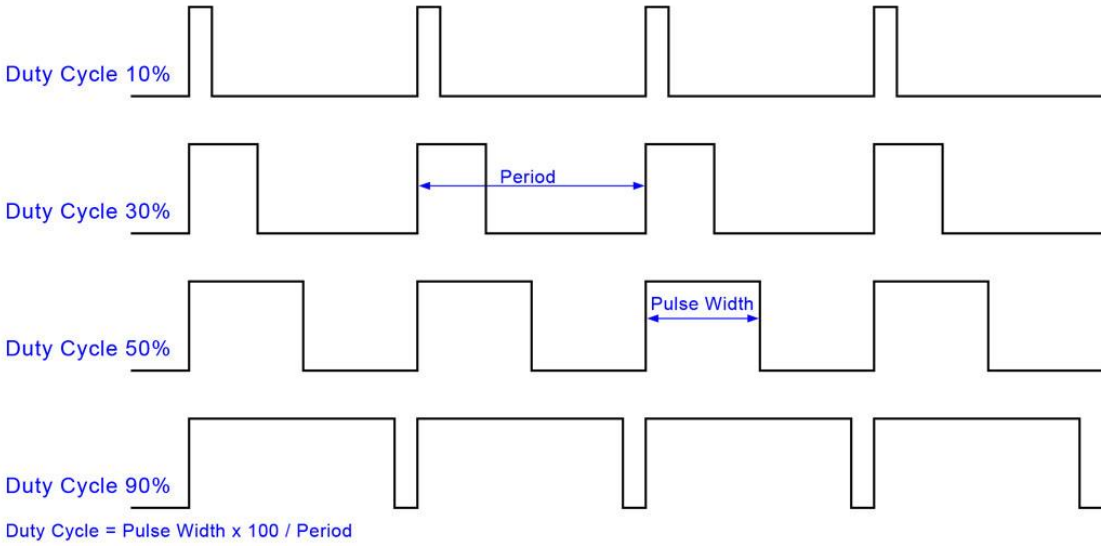


FIGURE 12: WAVEFORMS SHOWING PWM WITH DIFFERENT DUTY CYCLE [8]

The technique of Pulse Width Modulation helps us in getting the analog outputs with the use of a Low pass filter put in front of the digital output pins producing PWM waves.

Although this technique can be implemened using software algorithms but still the development boards and even our Raspberry Pi has counter based Hardware PWM modules which just needs to be initialised with few parameters and they would not disturb the processor while running in the background.

One such Hardware PWM channel is etended on GPIO18 of Raspberry Pi. We extended the GPIO18 on board and drove a LED through a transistor (as the current sinking and sourcing capacities of GPIOs are fixed). This need not have a low pass filter as a second stage because our eyes will behave like one. The changing duty cycle will be perceived as the change in intensity of the LED.

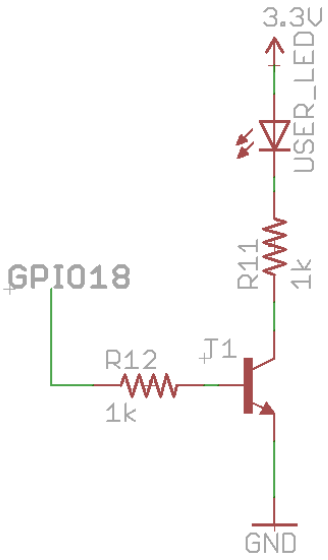


FIGURE 13: PWM CONTROLLED LED

After talking a lot about output block and the peripherals in it to such an extent, let us repopulate the number of GPIOs left with us, free to be used for another blocks.

2.2” GLCD took 3 GPIOs whereas PWM controlled LED took 1, hence the count is now 17-4=13 GPIOs.

With 13 GPIOs to go, let us now understand the next blocks!

`

3.3.2 INPUT

In an attempt to make the system even more interactive, an input through switches is always appreciated. We circled upon two different type of switches to make the inputs more fun. Also, we included a Real Time Clock for keeping a track of time while performing various experiments.

3.3.2.1 5-WAY JOYSTICK

Joysticks have always been fascinating as the kind of control they used to give us while playing our favorite video games in the childhood. They make the input system hassle free due to the degree of freedom offered by them. But, all the joystick we used to have were analog in nature, which means that once you move the joystick in any direction, voltages will be changed corresponding to X axis and Y axis. All cool till here, but

FIGURE 14: ANALOG JOYSTICK[9]

Sadly, Raspberry Pi doesn't have an integrated ADC which would have allowed us to use an analog joystick. Till now we have had our minds made to use a joystick so we started up searching for their digital counterparts and our search ended on a 5-way joystick which is basically a 5 way switch and each of its terminals needs to be connected to a GPIO.

The connection diagram tells us the exact connections to be made while using it in our design.

FIGURE 15: 5-WAY JOYSTICK[10]

`

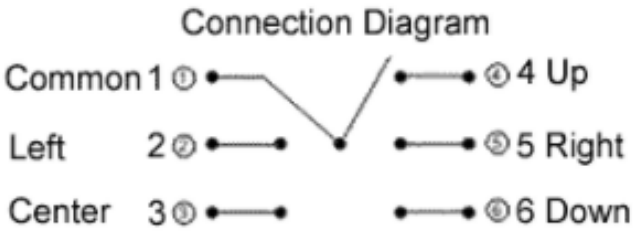


FIGURE 16: CONNECTION DIAGRAM OF A

5-WAY JOYSTICK

Now as we have put up a joystick in the circuitn for smooth manourvering in the GUI going to be developed by you or for any of the projects you imagine to be doing with विज्ञान हथेली पे , there is still a need of an independent user switch which is the basic most input of the system.

3.3.2.2 USER SWITCH

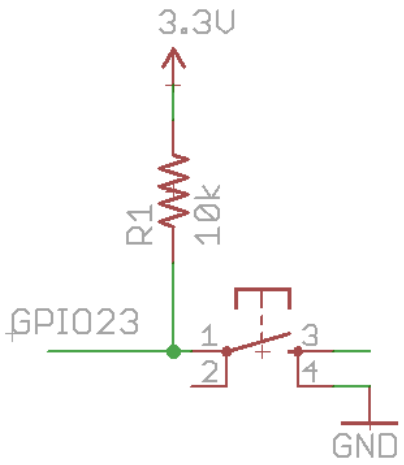


FIGURE 17: USER SWITCH

Fig 3.13 shows the circuit connection of the user switch where the default state of GPIO23 is high as it has been pulled up through a 10kilo ohm resistor. As soon as the switch is pressed, the logic state of GPIO23 changes to 0, hence detecting a switch press.

Putting these two input devices, we have used 6 more GPIOs, leaving us with 7 GPIOs.

`

3.2.2.3 RTC

The RTC which we used is PCF853 [11]. The PCF8563 is a CMOS1 Real-Time Clock (RTC) and calendar optimized for low power consumption. A programmable clock output, interrupt output, and voltage-low detector are also provided. All addresses and data are transferred serially via a two-line bidirectional I 2C-bus. Maximum bus speed is 400 Kbit/s. The register address is incremented automatically after each written or read data byte.

Symbol	Pin		Description
	SO8, TSSOP8	HVSON10	
OSCI	1	1	oscillator input
OSCO	2	2	oscillator output
$\overline{INT}$	3	4	interrupt output (open-drain; active LOW)
V_{SS}	4	5[1]	ground
SDA	5	6	serial data input and output
SCL	6	7	serial clock input
CLKOUT	7	8	clock output, open-drain
V_{DD}	8	9	supply voltage
n.c.	-	3, 10	not connected; do not connect and do not use as feed through

FIGURE 18: PIN DECRIPTION OF RTC PCF8563

Table 3.1 shows the pin description of RTC PCF8563 and as it uses I2C protocol to converse with another peripheral, Raspberry Pi's I2C pins are connected to corresponding I2C pins of PCF8563.

Interfacing PCF8563 doesn't require any other GPIO as per se, so we are still left with 7 unused GPIOs.

3.3.3 POWER

The power to the entire विज्ञान हथेली पे is provided by the Raspberry Pi. Since the Raspberry Pi takes up 5V, 2A as input, we distributed the same bus throughout the design. Since the onboard peripherals don't have specific power needs different than Raspberry Pi, this idea of common bus worked fine.

3.3.4 EXTENDED COMMUNICATION BUSES

Remember that we mentioned about the 'special' GPIOs which are dedicated for communication purposes? In this section we tell you about how we extended these communication protocol buses for the user to connect such peripherals which can talk with Raspberry Pi by using a particular communication protocol.

`

3.3.4.1 UART BUS

The most common of them all is the universal asynchronous receiver/transmitter (UART) [12]. It is the block of circuitry responsible for implementing serial communication. Essentially, the UART acts as an intermediary between parallel and serial interfaces. On one end of the UART is a bus of eight-or-so data lines (plus some control pins), on the other is the two serial wires - RX and TX.

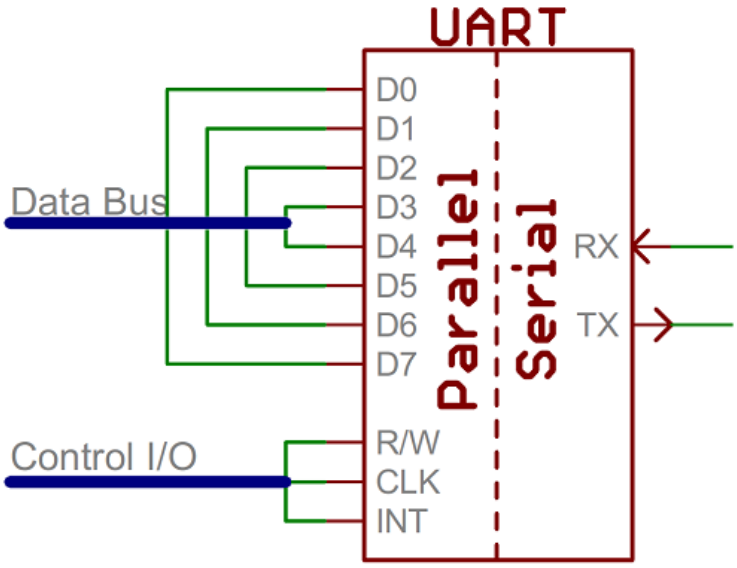


FIGURE 19: UART INTERFACING

As the R and T in the acronym dictate, UARTs are responsible for both sending and receiving serial data. On the transmit side, a UART must create the data packet - appending sync and parity bits - and send that packet out the TX line with precise timing (according to the set baud rate). On the receive end, the UART has to sample the RX line at rates according to the expected baud rate, pick out the sync bits, and spit out the data.

3.3.4.2 1-WIRE BUS

1-Wire [13] is the only voltage-based digital system that works with two contacts, data and ground, for half-duplex bidirectional communication. In contrast to other serial communication systems such as I²C or SPI, 1-Wire devices are designed for use in a contact environment. Either disconnecting from the 1-Wire bus or a loss of contact puts the 1-Wire slaves into a defined reset state. When the voltage returns, the slaves wake up and signal their presence. With only one contact to protect, the built-in ESD protection of 1-Wire devices is extremely high. With two contacts, 1-Wire devices are the most economical way to add electronic functionality to non-electronic objects for identification, authentication, and delivery of calibration data or manufacturing information.

`

Each 1-Wire slave device has a unique, unalterable, factory-programmed, **64-bit ID** (identification number), which serves as device address on the 1-Wire bus. The 8-bit **family code**, a subset of the 64-bit ID, identifies the device type and functionality. Typically, 1-Wire slave devices operate over the voltage range of 2.8V (min) to 5.25V (max). Most 1-Wire devices have no pin for power supply; they take their energy from the 1-Wire bus (**parasitic supply**).

Raspberry Pi's GPIO04 has a specific kernel level support as a 1-wire protocol pin. Hence we extended this bus so that if an application requires us to connect a sensor following 1-wire protocol, it can be interfaced easily without any hassle.

3.3.4.3 I2C BUS

This is just two wires, called SCL and SDA. SCL is the clock line. It is used to synchronize all data transfers over the I2C [14] bus. SDA is the data line. The SCL & SDA lines are connected to all devices on the I2C bus. There needs to be a third wire which is just the ground or 0 volts. There may also be a 5volt wire is power is being distributed to the devices. Both SCL and SDA lines are "open drain" drivers. What this means is that the chip can drive its output low, but it cannot drive it high. For the line to be able to go high you must provide pull-up resistors to the 5v supply. There should be a resistor from the SCL line to the 5v line and another from the SDA line to the 5v line. You only need one set of pull-up resistors for the whole I2C bus, not for each device, as illustrated below:

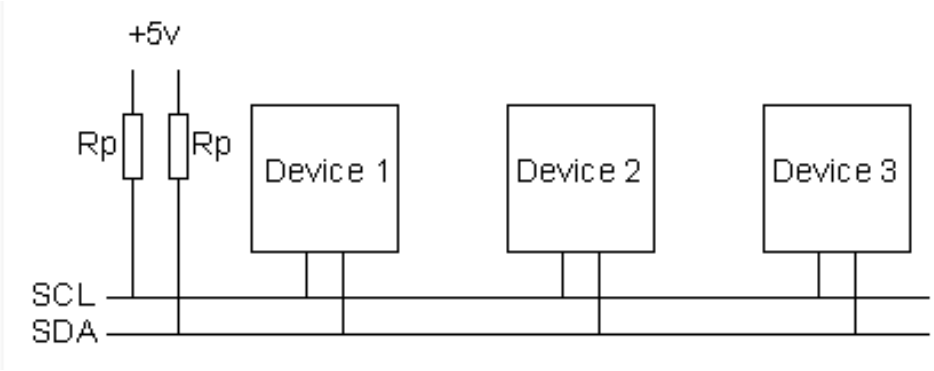


FIGURE 20: DEVICE CONNECTIONS OVER THE I2C BUS

Now someone can ask that we extended mostly all the buses but why not the SPI bus because विज्ञान हथेली पे houses a 2.2" GLCD which is used almost every time we work over nay application due to the importance of the visual feedback.

The last thing which should not go unnoticed is the booster pack connections for the modularity we want to achieve. This has been duly taken care of while capturing the schematic.

Now its turn to start making the schematic and laying out the board to be fabricated by any in-house PCB fab facility or through a custom PCB vendor.

`

3.4 SCHEMATIC DESIGNING AND LAYOUT

If you are new to **EAGLE**, try learning through video tutorials [15],

Every circuit designer first works out the design requirements like voltages and currents involved in the circuit, physical robustness and shielding against the ambient noise, which we have done till now.

After connecting the peripherals as discussed in the previous sections, we finalized the following schematic, forming out step one towards making विज्ञान हथेली पे a reality.

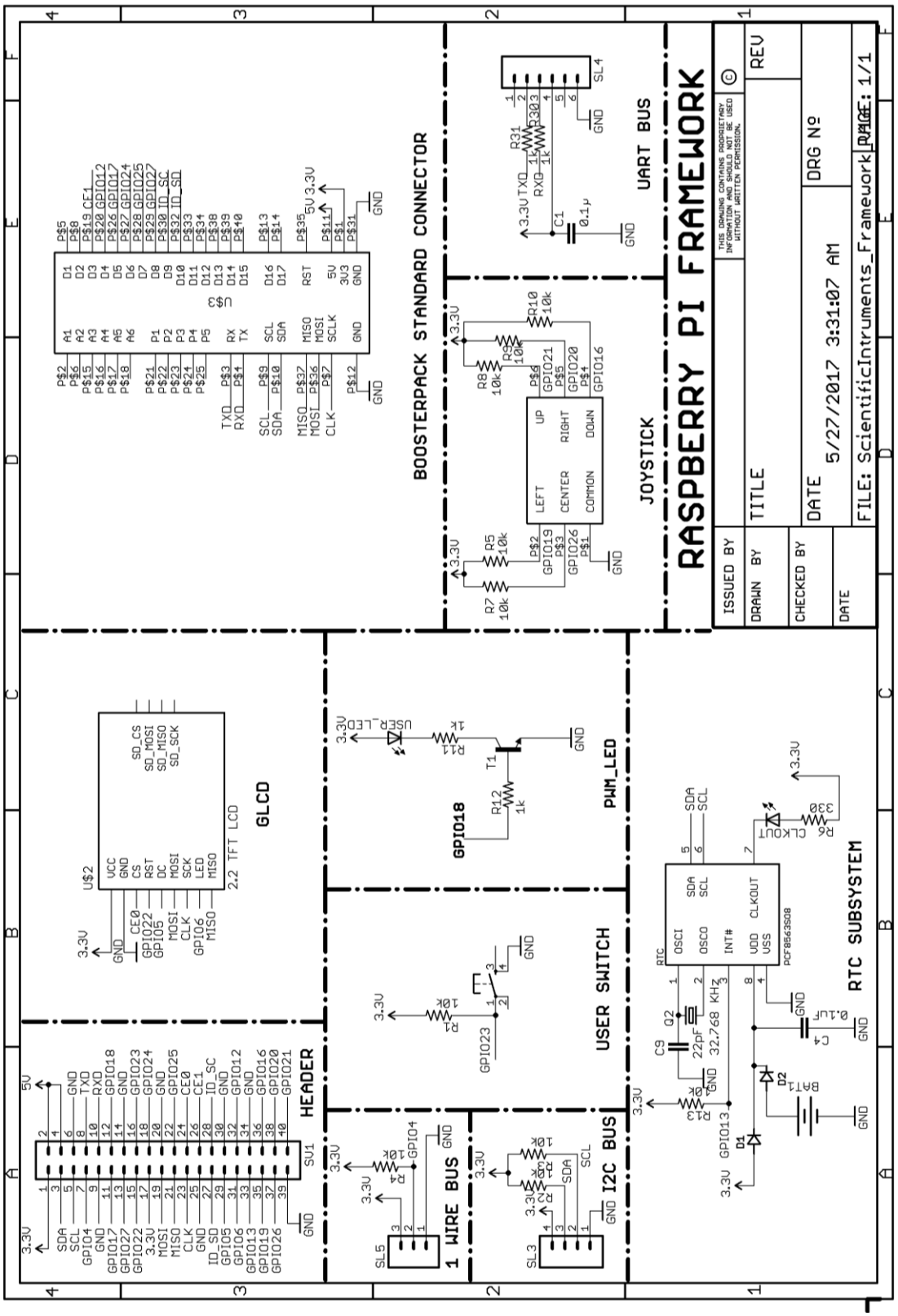


FIGURE 21: CIRCUIT SCHEMATIC FOR विज्ञान हथेली पे

`

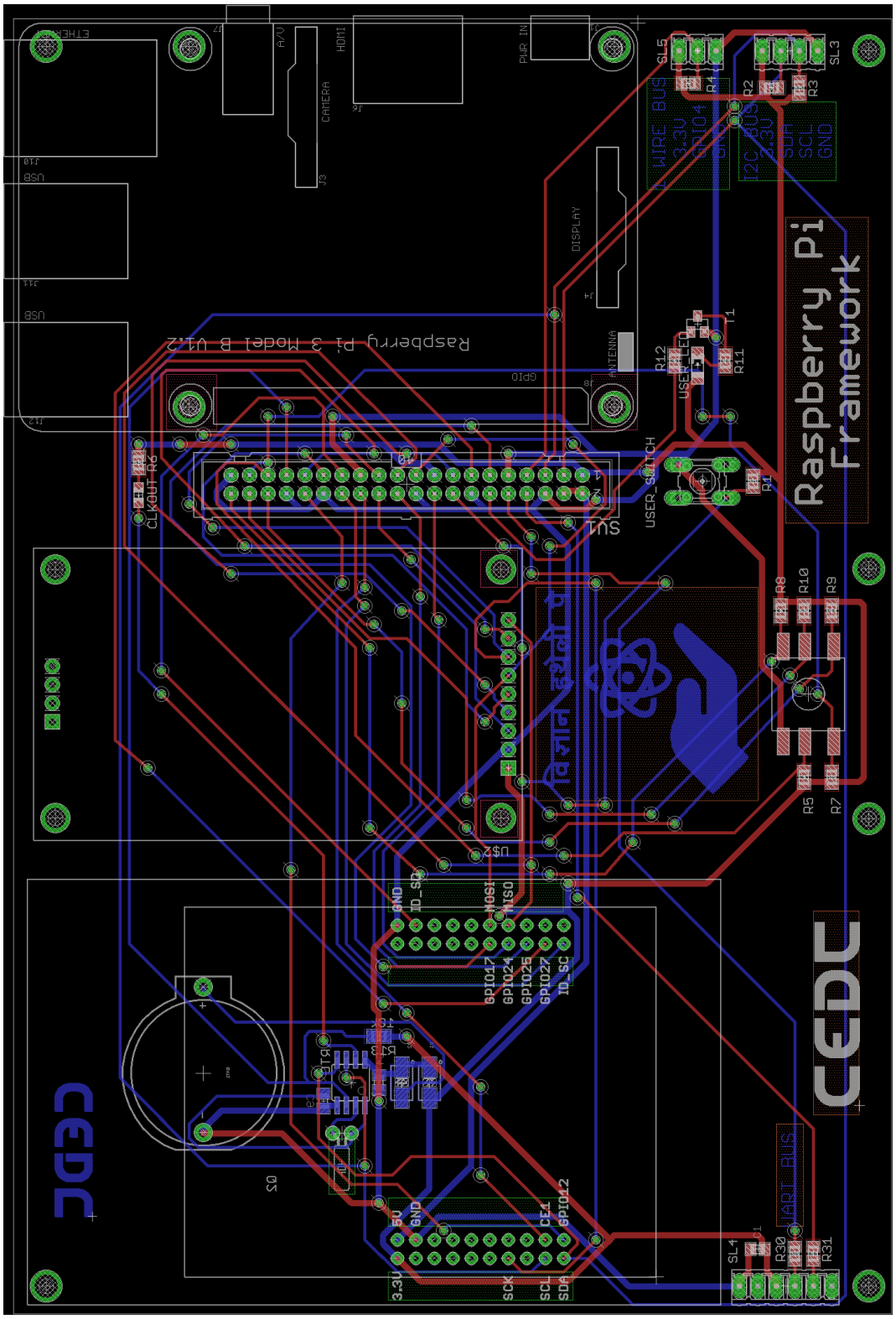


FIGURE 22: BOARD LAYOUT FOR विज्ञान हथेली पे

`

3.5 CREATING GERBER DATA

Once the schematic and board layout has been finalized, it's a green signal for us to start out for fabrication. In-house fabrication facilities can be used for single layer board fabrication but for a much professional touch and 2 layered boards, we preferred to get this done by a custom PCB fabricator.

The custom fabricator will ask for the Gerber Data which can be easily created using EAGLE. The steps are as follows:

1. Switch to the CAM processor while you were in the board layout window.

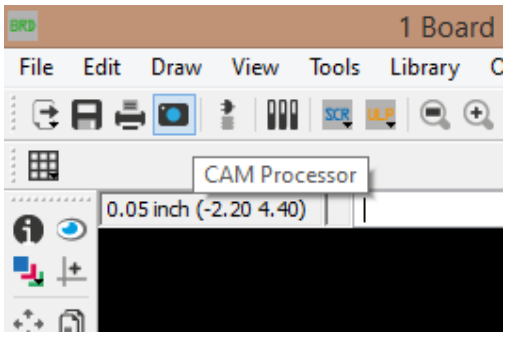


FIGURE 23: CAM PROCESSOR OPTION

2. CAM Processor dialog box appears.

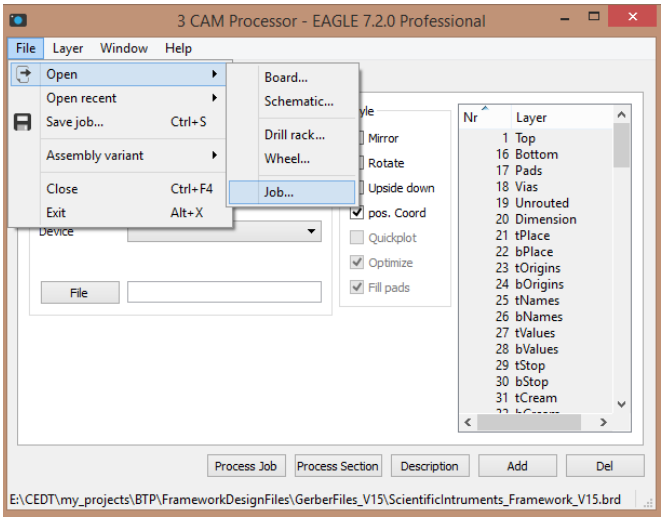


FIGURE 24: CAM PROCESSOR DIALOG BOX

`

3. Chose ‘Open’ and select ‘Job’ from the drop down menu. Choose ‘excellon.cam’ and select ‘Open’.

4. Now select the device as Excellon_24 and click on ‘Process Job’. This will generate the drill data of the board file.

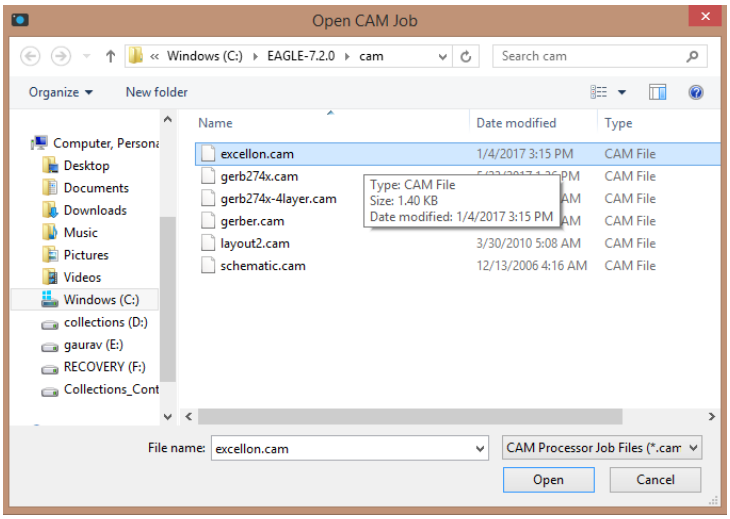


FIGURE 25: CHOOSING THE JOB

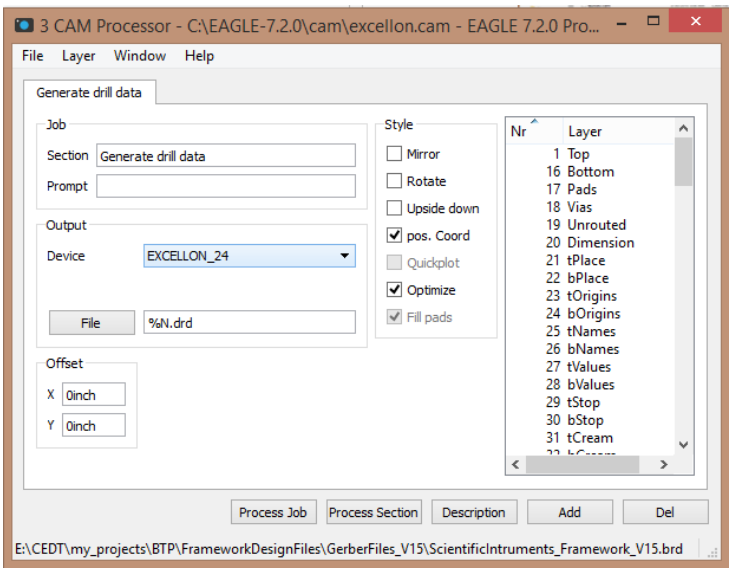


FIGURE 26: CHOOSING THE DEVICE

5. Now repeat step 3 and choose ‘gerb274x.cam’ from the available jobs.

6. Deselct the layers that you don’t want to be included in the final gerber data and then click on Process Job.

`

7. The Gerber Data has been cretaed and now you can verify this gerber data by using any 3D gerber viewer such as mayhewlabs.com

We are attaching some sample images captured from the 3D viewer which allows you to see your circuit board even before it is sent for fabrication.

This was all about the hardware aspects of विज्ञान हथेली पे and now we move into a totally new world where we used this platform in association with the boosterpacks made for specific applications. We made a Software Defined Radio, a Photometer, 2 talking gadgets and a milliohmeter using the approach we have been talking about till now. And if we can, even you can! Jai Hind!

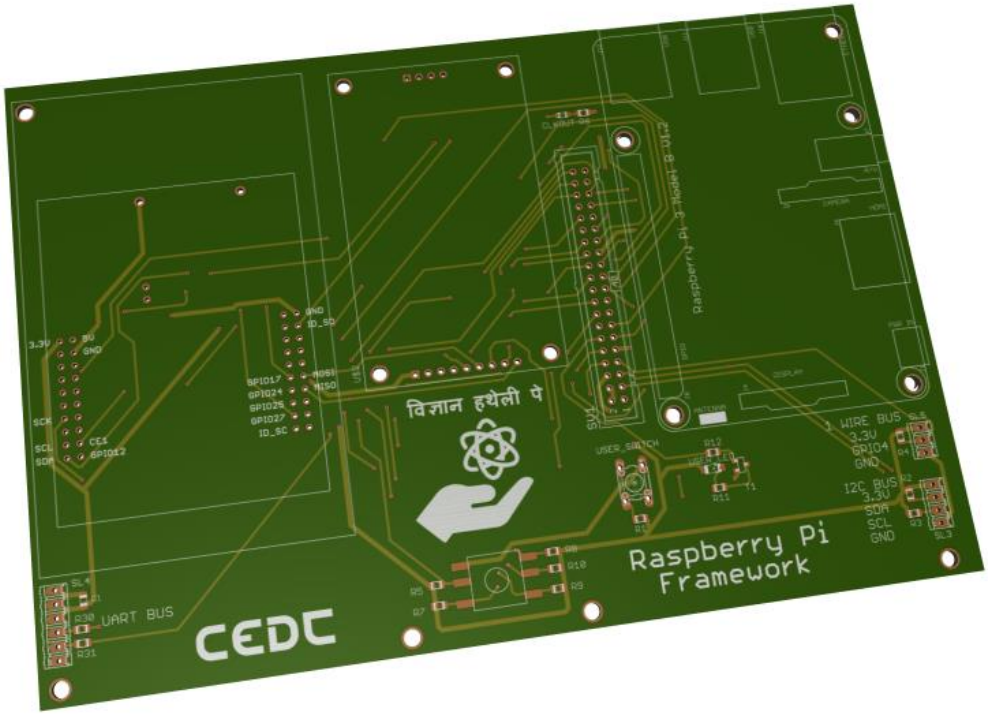


FIGURE 27: COMPONENT SIDE

`

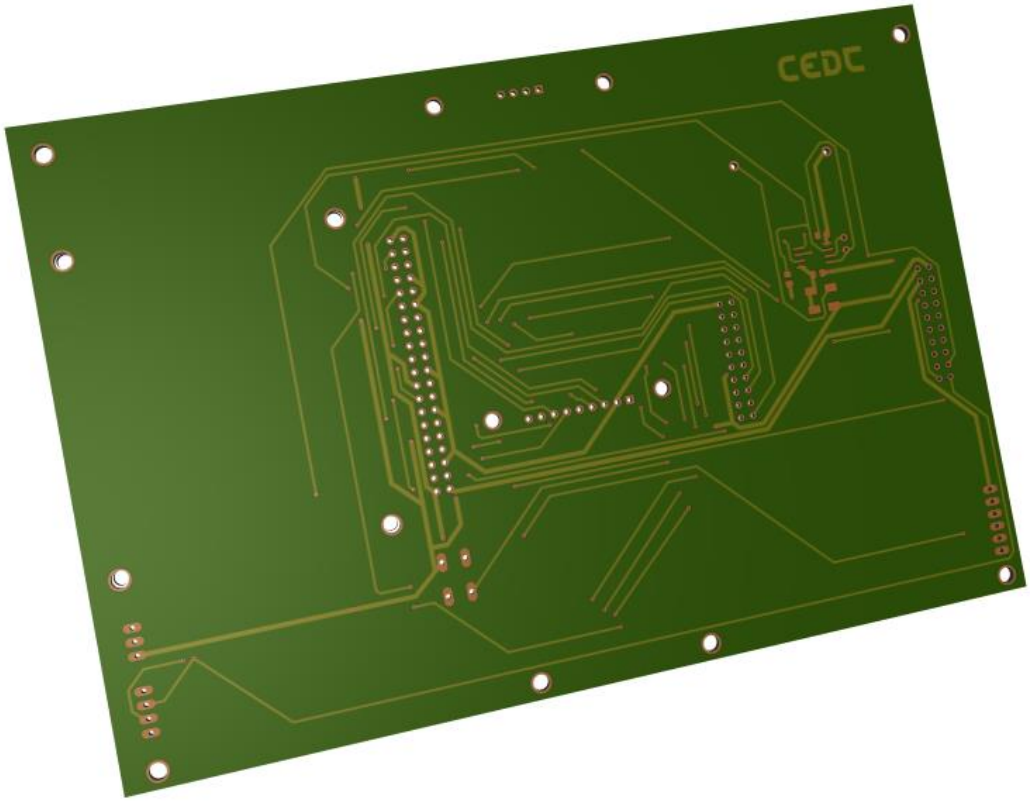


FIGURE 29: SOLDER SIDE

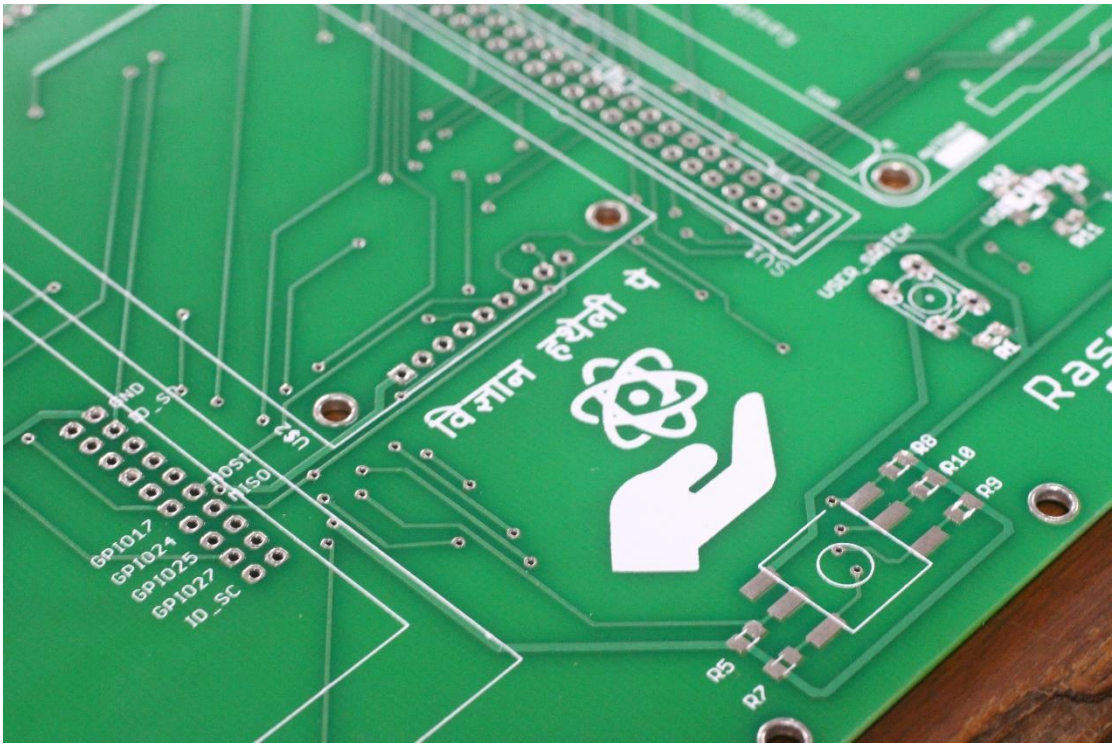


FIGURE 28: THE REAL PCB AFTER FABRICATION

`

FIGURE 30: विज्ञान हथेली पे WITH THE SDR BOOSTERPACK

`

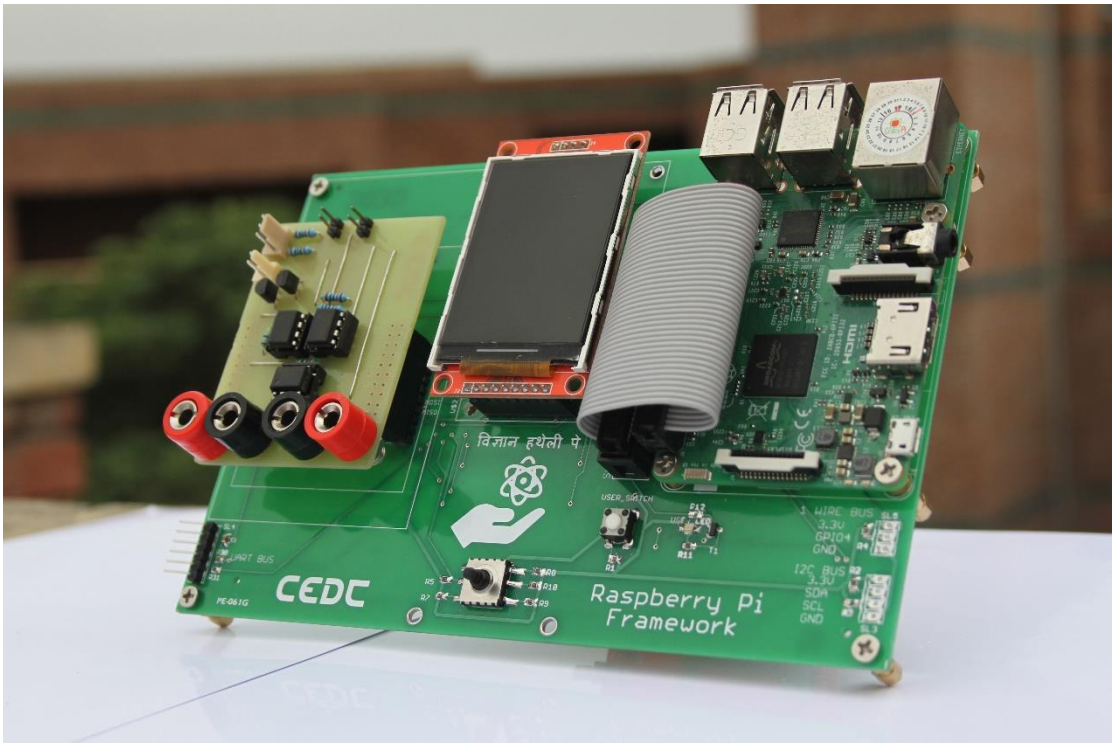


FIGURE 31: विज्ञान हथेली पे WITH THE WEIGHING SCALE AND MILLI-OHMMETER BOOSTERPACK

`

CHAPTER 4: HF SDR

Time to see some real thing going on using विज्ञान हथेली पे!

Radio, even as a term and what it means in the real sense, was introduced to us by our mentor, Prof. Dhananjay V. Gadre who is himself an Amateur Radio [16] Operator, with the call-sign VU2NOX.

Anyone can lag behind in understanding that why did we take this up as a project? Even we were when he introduced us to the hobby of Amateur Radio. What potential does Amateur Radio has as a hobby? Well, our mentor credits Amateur Radio for pushing him into understanding electronics to the deeper extents.

Radios are potent enough to make a kid go crazy enough to be busy with them all day long, enthusiastic enough to find the answers and creative enough to experiment. The rom of experimentation given by Radios stands unmatched and unparalleled. More than anything, they sow the basic most seed of "getting your hands dirty".

It's the result of his motivation and guidance at every step that one of the students involved in विज्ञान हथेली पे, Gaurav Tyagi is now a General Grade Amateur Radio operator too, with the call-sign VU2YGT. Anshuman Mishra has applied for the license and so have the kids at CEDT.

Our need to adapt the boosterpack standards have been clearly described in the previous section, making it very obvious that even the HF SDR has to be in the form of a boosterpack. This chapter focusses on a special detector circuit which made it possible for us to decode the energy around us, feeling the Radio waves.

4.1 WHAT IS SOFTWARE DEFINED RADIO?

The dependency of Radios on Software? How is that even possible? It has been our whole life that we have seen AM/FM Radios at our homes, they didn't have any computer in them. Then how are we going to talk about Radios with विज्ञान हथेली पे?

Digital Signal Processing. Yes! You read it right. With the advent of Digital Signal Processing, the conventional analog hardware has seen a drastic phase change in terms of hardware implementation and the way any analog signal was treated before anyone tried doing it with DSP. Radio waves are analog, their demodulation circuit is analog, the filters used there are analog and the tuning is done on the basis of what frequency we want to listen to and what we want to reject. Simple!

But, if you haven't seen any radio split wide open when it would have slipped from your hands, or you could have been so enthusiastic to see what is there inside, you won't be able to appreciate the amount of hardware components it takes to build a good quality high order filter. We are attaching a circuit schematic for your reference on how big and complex an analog Radio waves decoding circuit can be. See Fig 4.1. And not to forget that we used to have different set of radios for listening AM and FM stations. So

`

if you wish to demodulate different schemes, you have to have different set of hardware for each of them.

_Also, not every radio that uses software is an SDR [17]. There is a distinct difference between a radio that internally uses software for some of its functions and a radio that can be completely redefined in the field through modification of software. The latter is a software-defined radio.

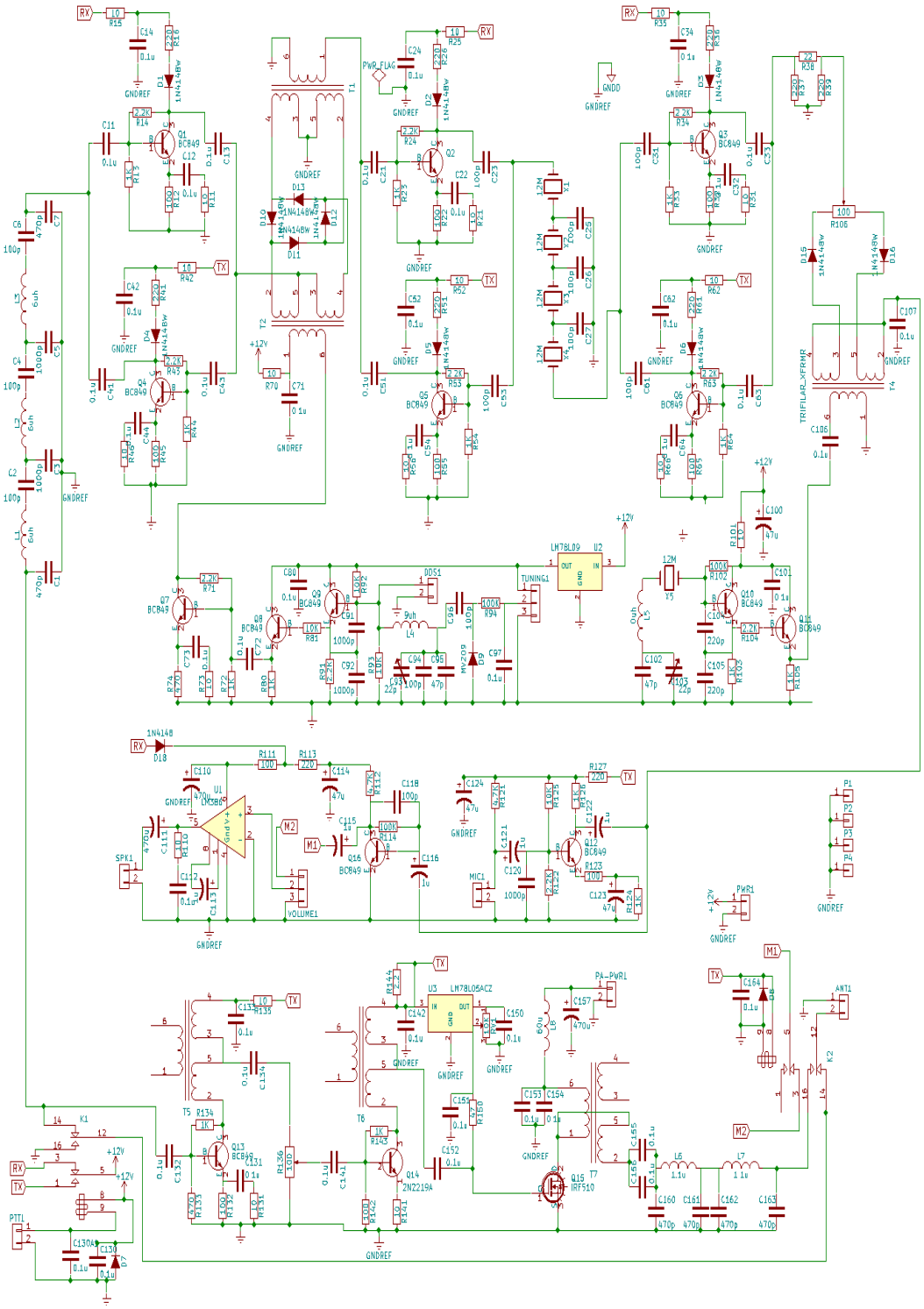


FIGURE 32: AN EXAMPLE OF ANALOG DEMODULATOR FOR HF WAVES [18]

`

Being said that, we present to you a complex yet simple way to demodulate Radio waves. All Digital.

Even before describing the circuit which forms the core of HF-SDR, it is of utmost importance that how will Raspberry Pi come into the picture to ease out things. Now, we know that DSP will definitely do the job of reducing the Hardware but one needs to understand that it is Digital Signal Processing, the word 'Digital' you read there must be taken into account before proceeding to what we did. Since the signals which are going to be processed are digital in nature, this implies that the analog Radio waves must be converted to a digital format. This DSP is done by none other than our tiny yet powerful Raspberry Pi. The technique we are going to refer utilizes a PC soundcard for all the audio processing and the PC displays all the features corresponding to the digital algorithms corresponding to various demodulation schemes. Raspberry Pi doesn't have a soundcard hence a soundcard must be connected to it as an added peripheral.

Since the need of converting the analog Radio waves to digital format using a PC soundcard has aroused, the technical specifications of the soundcard in use will play a critical role while designing the HF-SDR boosterpack. The following sections would deal with these points.

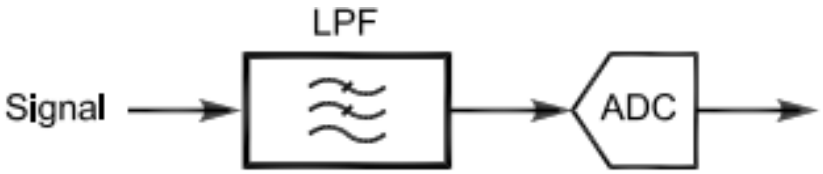


FIGURE 33: SIGNAL FLOW FOR ANY PC BASED SDR

4.1.1 SAMPLING THE RADIOWAVES

We choose to use a standard 16-bit PC sound card that has a maximum sampling rate of 44,100 Hz. By sampling theorem, this means that the maximum-bandwidth signal we can accommodate is 22,050 Hz. With quadrature sampling[19], discussed later, this can actually be extended to 44 kHz. Most sound cards have built-in antialiasing filters that cut off sharply at around 20 kHz. We will need to convert the RF signal to audio frequencies in a way that allows removal of the unwanted mixing products or images caused by the down-conversion process. The simplest way to accomplish this while maintaining wide dynamic range is to use D-C techniques to translate the modulated

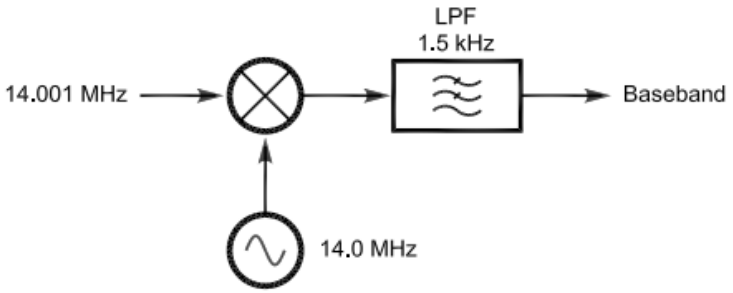


FIGURE 34: RADIOWAVE TO BASEBAND CONVERSION

`

RF signal directly to baseband. We can mix the signal with an oscillator tuned to the RF carrier frequency to translate the bandwidth-limited signal to a 0-Hz IF as shown in Fig 4.3

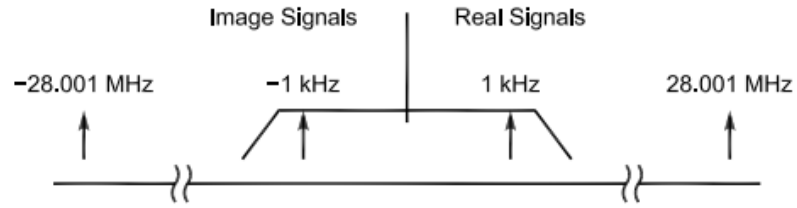


FIGURE 35: FREQUENCY SPECTRUM AFTER MIXING EXAMPLE SIGNALS

But the mixing of signals don't just result in the frequency bands that we want, they also result in image frequencies. This is explained further.

A low-pass filter easily removes the 28.001-MHz sum frequency and its –28.001-MHz image but the –0.001-MHz difference-frequency image will remain in the output. This unwanted image is the opposite sideband centered on the 14.000-MHz carrier frequency. This would not be a problem if there were no signals below 14.000 MHz to interfere. As previously stated all undesired signals between 13.99815 and 14.000 MHz will translate into the passband along with the desired signals above 14.000 MHz. The image results in increased noise in the output.

So how can we remove the image-frequency signals? Itcan be accomplished through quadrature mixing. Phasing or quadrature transmitters and receivers—also called Weaver-method or image-rejection mixers—have existed since the early days of single sideband. In fact, my first SSB transmitter was a used Central Electronics 20A exciter that incorporated a phasing design. Phasing systems lost favor in the early 1960s with the advent of relatively inexpensive, high-performance filters.

4.1.2 QUADRATURE SAMPLING

Now we talk about I and Q sampling or Quadrature Sampling[19].

Remember that in AM envelope detection, both modulation sidebands carry information energy and both are desired at the output. Only amplitude information is required to fully demodulate the original signal. The problem is that most other

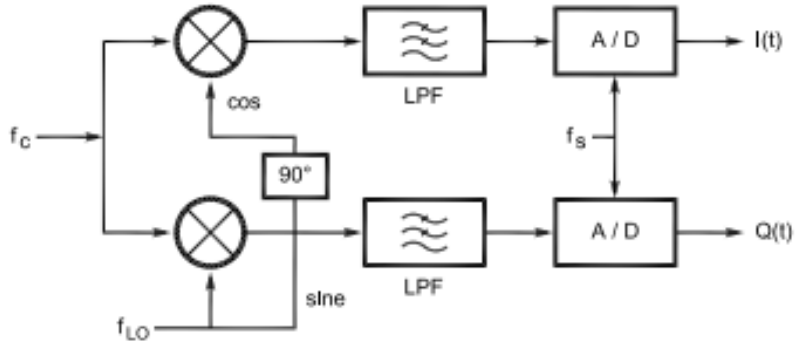


FIGURE 36: A BASIC QUADRATURE SAMPLING MIXER

`

modulation techniques require that the phase of the signal be known. This is where quadrature detection comes in. If we delay a copy of the RF carrier by 90° to form a quadrature (Q) signal, we can then use it in conjunction with the original in-phase signal and the math we learned in middle school to determine the instantaneous phase and amplitude of the original signal.

Quadrature sampling mixer: The RF carrier, fc, is fed to parallel mixers. The local oscillator (Sine) is fed to the lower-channel mixer directly and is delayed by 90°(Cosine) to feed the upper-channel mixer. The low-pass filters provide antialias filtering before analog-to-digital conversion. The upper channel provides the in-phase (I(t)) signal and the lower channel provides the quadrature (Q(t)) signal. In the PC SDR the low-pass filters and A/D converters are integrated on the PC sound card.

4.1.3 THE TAYLOE DETECTOR

Getting a bit of clarity of I and Q signals, let us move on to the technique of getting I and Q signals that we have been talking about. It is one of the most innovative and elegant design by Dan Tayloe, N7VE. Dan, who works for Motorola, has developed and patented (US Patent #6,230,000) what has been called the Tayloe detector.

The beauty of the Tayloe detector[20] is found in both its design elegance and its exceptional performance. It resembles other concepts in design, but appears unique in its high performance with minimal components. In its simplest form you can build a complete quadrature down converter with only three or four ICs Fig 4.6 illustrates a single-balanced version of the Tayloe detector. It can be visualized as a four-position rotary switch revolving at a rate equal to the carrier frequency. The 50-Ω antenna impedance is connected to the rotor and each of the four switch positions is connected to a sampling capacitor. Since the switch rotor is turning at exactly the RF carrier frequency, each capacitor will track the carrier's amplitude for exactly one quarter of the cycle and will hold its value for the remainder of the cycle. The rotating switch will therefore sample the signal at 0°, 90°, 180° and 270°, respectively.

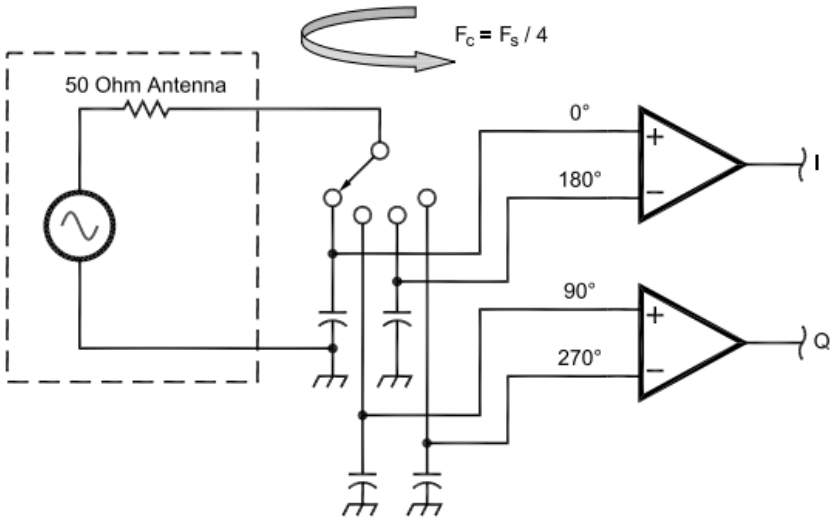


FIGURE 37: A VERSION OF THE TAYLOE DETECTOR

`

As shown in Fig 4.7, the 50-Ω impedance of the antenna and the sampling capacitors form an R-C low-pass filter during the period when each respective switch is turned on. Therefore, each sample represents the integral or average voltage of the signal during its respective one-quarter cycle. When the switch is off, each sampling capacitor will hold its value until the next revolution. If the RF carrier and the rotating frequency were exactly in phase, the output of each capacitor will be a dc level equal to the average value of the sample. If we differentially sum outputs of the 0° and 180° sampling capacitors with an op amp (see Fig 10), the output would be a dc voltage equal to two times the value of the individually sampled values when the switch rotation frequency equals the carrier frequency. Imagine, 6 dB of noise-free gain! The same would be true

for the 90° and 270° capacitors as well. The 0°/180° summation forms the I channel and the 90°/270° summation forms the Q channel of the quadrature down conversion.

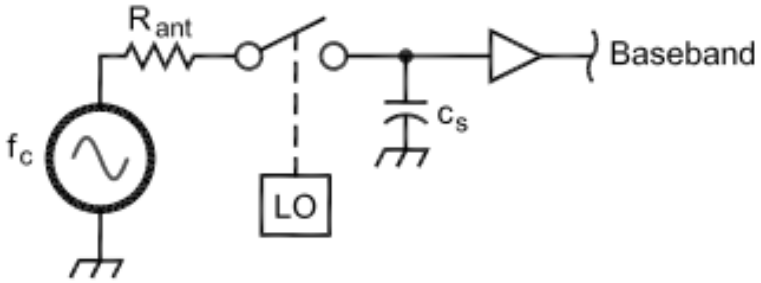


FIGURE 38: A TRACK AND HOLD SAMPLING CIRCUIT

In a track and hold sampling circuit each of the four sampling capacitors in the Tayloe detector form an RC track-and-hold circuit. When the switch is on, the capacitor will charge to the average value of the carrier during its respective one quarter cycle. During the remaining three quarters cycle, it will hold its charge. The local-oscillator frequency is equal to the carrier frequency so that the output will be at baseband.

All these concepts form the basis of the designing of HF-SDR. Adding to these, Fig 4.8 shows that how these quadrature sampling patterns can be generated so as to have the radio wave sampled at every 90 degrees.

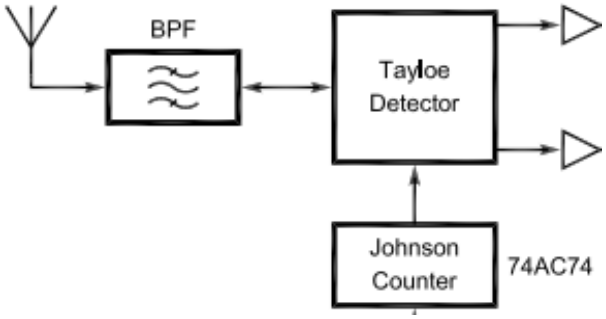


FIGURE 39: JOHNSON COUNTER BASED IMPLEMENTATION

How to feed the frequency to this Johnson counter?

`

4.2 ADAPTATION OF DDS DS1085 AS LOCAL OSCILLATOR

The DS1085 [21] is a dual-output frequency synthesizer requiring no external timing components for operation. It can be used as a standalone oscillator or as a dynamically programmed, processor-controlled peripheral device. An internal master oscillator can be programmed from 66MHz to 133MHz with three resolution options of 10 kHz, 25 kHz, and 50 kHz. A programmable, 3-bit prescaler (divide-by-1, 2, 4, or 8) permits the generation of a reference oscillator output (OUT0) from the master, ranging from 8.2MHz to 133MHz. A second independent prescaler and a 1-to-1025 divider allows the generation of a main oscillator output (OUT1) from 8.1 kHz to 133MHz. The two outputs, although synchronous with the master, can be independently programmed. The combination of programmable master oscillator, prescalers, and dividers allows the generation of thousands of user-specified frequencies. All master oscillator, prescaler, and divider settings are stored in NV (EEPROM) memory, providing a default value on power-up that allows it to be used as a standalone oscillator. A 2-wire serial interface allows in-circuit, on-the-fly programming of the master oscillator, prescalers (P0 and P1), and divider (N). This allows dynamic frequency modification, if required, or, for fixed-frequency applications, the DS1085 can be used with factory- or user-programmed values.

The following section discusses about the interfacing of DS1085.

4.2.1 INTERFACING THE DS1085

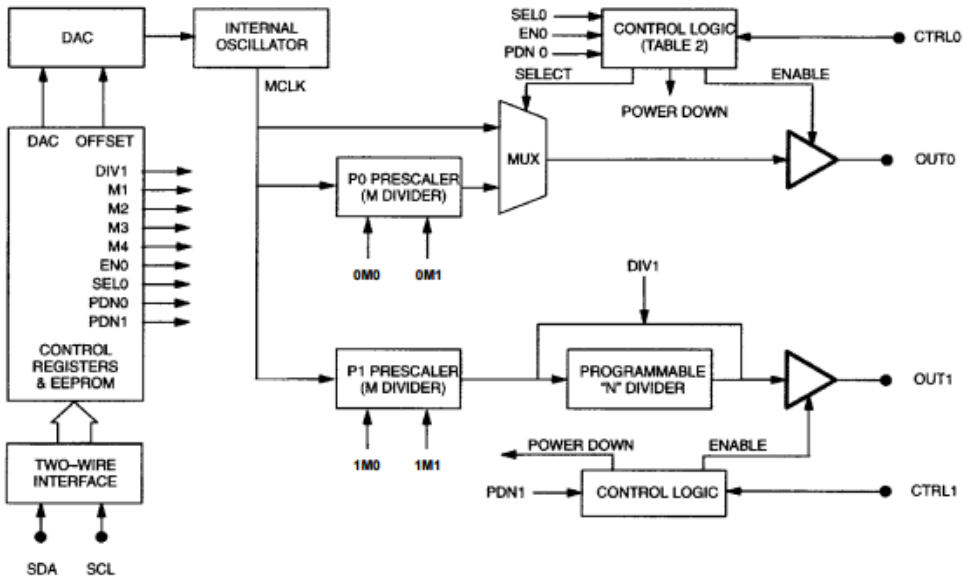


FIGURE 40: DS1085 BLOCK DIAGRAM

1. The DS1085 consists of five major components:
2. Master oscillator control DAC
3. Internal master oscillator (66MHz to 133MHz)

`

4. Prescalers (divide-by-1, 2, 4, or 8)
5. Programmable divider (divide-by-1 to 1025)
6. Control registers

The internal master oscillator provides the reference clock (MCLK), which is fed to the prescalers and programmable dividers. The frequency of the oscillator can be user-programmed over a two-to-one range in increments equal to the step size, by means of a 10-bit control DAC. The master oscillator range is 66MHz to 133MHz, which is larger than the range possible with the 10-bit DAC resolution and available step sizes. Therefore, an additional register (OFFSET) is provided that can be used to select the range of frequency over which the DAC is used.

Refer to the appendix for a sample code to interface DS1085 with a TIVA-C microcontroller.

Designing has been fun, always!

`

4.3 SCHEMATIC DESIGN

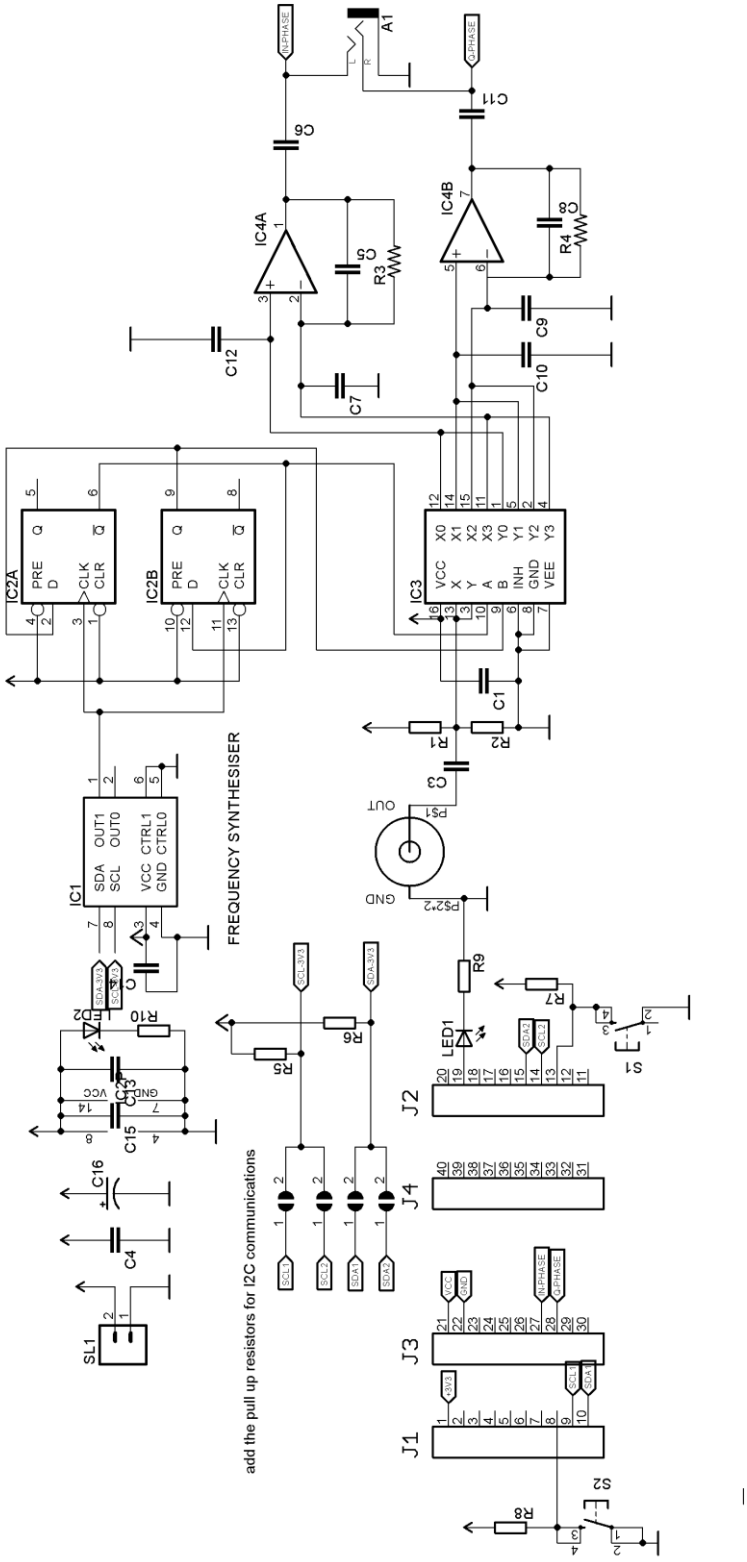


FIGURE 41: HF SDR SCHEMATIC

`

4.4 THE BOARD LAYOUT

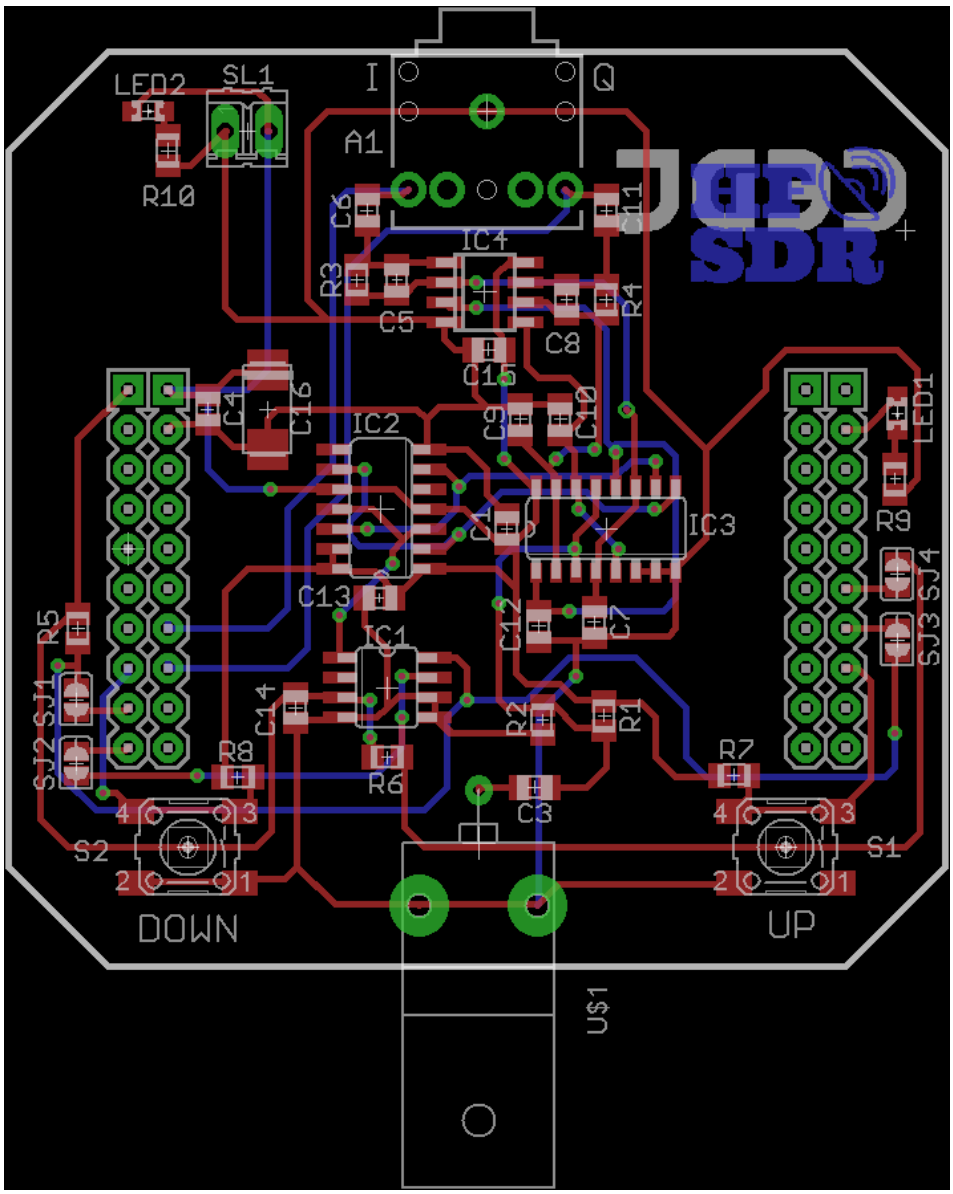


FIGURE 42: BOARD LAYOUT FOR HF-SDR

`

4.6 BILL OF MATERIALS

TABLE 4: BILL OF MATERIALS OF HF-SDR

Part	Value	Package	Note
A1		AUDIO-JACK	3.5mm Audio Jack
A2		4X10-BOOSTERPACK	
C1		C0805	CAPACITOR
C3	100	C0805	CAPACITOR
C4		C0805	CAPACITOR
C5	330	C0805	CAPACITOR
C6	0.1	C0805	CAPACITOR
C7	0.022	C0805	CAPACITOR
C8	330	C0805	CAPACITOR
C9	0.022	C0805	CAPACITOR
C10	0.022	C0805	CAPACITOR
C11		C0805	CAPACITOR
C12	0.022	C0805	CAPACITOR
C13		C0805	CAPACITOR
C14		C0805	CAPACITOR
C15		C0805	CAPACITOR
C16		C/6032-28R	
IC1	DS1085L	SOIC08	
IC2	74HC74D	SO14	Dual D type positive edge triggered FLIP FLOP
IC3	74HC4052	SO16	Dual 4:1 analog mux
IC4	LM358D	SO08	OP AMP also LM158; LM258; LM2904
LED 1	LED0805	CHIPLED_0805	
LED 2	LED0805	CHIPLED_0805	
R1	1k	M0805	RESISTOR
R2	1k	M0805	RESISTOR
R3	5.1k	M0805	RESISTOR
R4	5.1k	M0805	RESISTOR
R5	10k	M0805	RESISTOR
R6	10k	M0805	RESISTOR
R7	10k	M0805	RESISTOR
R8	10k	M0805	RESISTOR
R9	1k	M0805	RESISTOR
R10	1k	M0805	RESISTOR
S1	UP	6MM_SWITCH	OMRON SWITCH
S2	DOWN	6MM_SWITCH	OMRON SWITCH
SJ1	SOLDER_JUMPER	SJFAB	

`

SJ2	SOLDER_JUMPER	SJFAB	
SJ3	SOLDER_JUMPER	SJFAB	
SJ4	SOLDER_JUMPER	SJFAB	
SL1		02P	AMP QUICK CONNECTOR
U$1		BNC_FEMALE_SDR	

`

CHAPTER 5: TALKING FRAMEWORK

Back in chapter 1, we talked about this fascinating feature, called the talking paradigm. What is it all about? How is it implemented? What does it have to do with विज्ञान हथेली पे? These are some of the questions that we will address in this chapter. Talking paradigm refers to imbibing the devices around us with the ability to relay their data using audio i.e. the power of speech. Yes! We are telling you talking devices are indeed possible and implementable. No longer are they work of science fiction.

First we need to ask do we actually need to go through all the trouble. Do we need the devices to speak their data? Well, how about you think about visually impaired people. Won't they be able to use the system efficiently if it talks to them? Imagine another situation, a worker monitoring various parameters of a critical system, making note of them. In such a case, won't it be helpful if the system spoke the data? Yeah!

5.1 IMPARTING POWER OF SPEECH TO THE SYSTEM

Well, enough about why. Let's talk about how. How do we impart the power of speech to the system? Well, it depends on how much do you want the system to talk? Obviously, you can't go ahead and make a talking robot as those shown in the movies, at least not yet. But you can definitely lay down a limited set of vocabulary according to the requirement of the system and work accordingly. There can be two primary approaches that can be used to enable a device to speak.

First approach could be that you use a basic skeleton set of audio files and dynamically "stitch" together the files to make complex words and phrases according to certain semantic rules. This approach would definitely enable the device to speak a large number of words and phrases and is quite appealing. However, there are some major drawbacks to this approach. Semantic rules vary from language to language, hence, a separate code would need to be written to define those semantic rules for each language added. This reduces the scalability of the solution. In addition to that, if the files are stitched together dynamically for playback, the code itself becomes complex.

An alternative approach can be developed by keeping in mind the fact that the purpose of the talking device is, for our purpose, limited to relay of sensor data, which has a limited range and precision. So if we limit our range and precision values to specific values, we could theoretically store the audio files of all the data within the specified range having the specified precision. Indeed, this approach is so versatile that theoretically limitless languages can be stored and played back using the same program! In fact, in case of the second approach, the only limiting factor is the size of available storage space.

`

We preferred to use the second approach to demonstrate the proof of concept of a multilingual talking device.

So far, we finalized the approach we desire to use to playback the audio file, but then the question arises, from where do those files come from? We have an answer ready for you. Anywhere! Yes, you heard it right, anywhere! As long as the audio in it is decipherable, any file would do. Here, we highlight some of the sources from where you could get the files.

5.1.1 GOOGLE TTS

The Google Translate [22] text-to-speech engine is one of the most powerful speech synthesis engines. It has a support of over 20 languages. Google provides an API to generate a WAV file for a given text. Some of the languages sound somewhat mechanical. This is perhaps the best source for audio files on such a large scale. One could write a script and download the files for any desired range, precision and language.

5.1.2 RESPONSIVE VOICE

Responsive Voice [23] is an alternative to Google TTS which offers text-to-speech services to users. It has support for 51 languages. The service provides an API to enable TTS to be used in a given application. One could use it to obtain the audio files in a particular project. An important feature of Responsive Voice is that it allows the users to select the gender of the voice, which is currently unavailable in Google TTS.

5.1.3 SELF RECORDED

The previous two options provide an automated way to obtain the audio files required for the project, but they offer a limited choice in languages and sometimes the voice created by their powerful engines sounds too mechanized. In such a scenario, it is advantageous to record the audio oneself and integrate it into the software. You just pick up the microphone, go to a quiet place and start recording!

5.2 ADAPTATION FOR विज्ञान हथेली पे

We developed a generalized framework that has the capability of making any measurement device talk able. As long as there is a sensor to obtain the measurement, this framework can make it speak. The range we chose was -150.0 to 150.0 and the precision was set to 0.5. In order to demonstrate the multilingual aspect, the framework currently supports two languages: English and Hindi.

For proof of concept, we chose two very common devices to use the framework with: thermometer and weighing scale

`

5.2.1 TALKING THERMOMETER

Temperature is one of the most common parameters that is measured in a device, and also the easiest. Now, you might ask how a thermometer is a scientific instrument. Well, almost all experiments have an underlying temperature condition attached to it. Especially chemistry and biology experiments, where temperature must be tightly controlled for reliable results. This widespread requirement of controlled temperature requirement makes the thermometer one of the most important scientific instruments Thus, it is perhaps the most appropriate device which should have the talking feature.

5.2.1.1 INTERFACING THE TEMPERATURE SENSOR DS18B20

DS18B20 [24] is a digital temperature sensor using the 1-wire protocol. It provides 9-bit to 12-bit Celsius temperature measurements along with an alarm function. Each DS18B20 has a unique 64-bit address which enables multiple such sensors to be interfaced on the same bus.

FIGURE 43: DS18B20 SENSOR

The 1-wire protocol is supported at the kernel level in the Raspbian [25] linux distribution on GPIO 4 of the Raspberry Pi 3. It uses a device tree overlay named "w1-gpio [26]" to communicate data over this bus. Another kernel module named "w1-therm [27]" interfaces with the DS18B20 and obtains its data. The data is written to a file which can then be read from by the temperature monitoring programs.

5.2.1.2 CIRCUIT DESCRIPTION

The circuit for interfacing a DS18B20 with **<u>विज्ञान हथेली पे</u>** is quite simple. The base board has provision for a 3-pin one-wire bus connection. The DS18B20 has 3 pins, VDD (3.3V), Ground and the Data. The data line needs to be pulled up using a 10kΩ resistor.

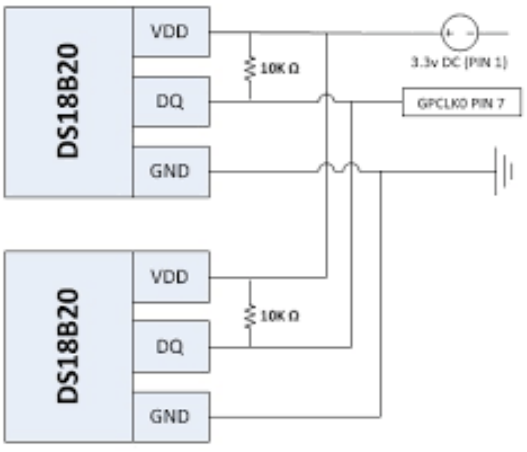


FIGURE 44: CONNECTIONS

The kernel module, once loaded, would continuously probe the 1-wire bus for existence of a sensor. If a sensor is detected, the module creates a directory structure according to the address of the sensor.

Example path of the directory: /sys/bus/w1/devices/28-00000283c6cd/

`

Once the directory structure is created, the kernel repeatedly queries the sensor for the temperature and writes it to the w1_slave file. So, the role of the monitoring program reduces to reading the w1_slave file and extracting the temperature data.

5.2.2 TALKING WEIGHING SCALE

Another intuitive application of talking framework would be with weighing scale. Weighing scale as a scientific instrument is used in gravimetric analysis in chemistry. Many other physics experiments use weight and mass calculations as a subset of the entire procedure. Thus, this becomes another important parameter that can be augmented with the talking framework.

5.2.2.1 STRAIN GAUGES AND LOAD CELL

How does a weighing scale work? How do we take the measurement? These are some of the questions we'd like to answer now. The basic transducer involved in measurement of weights is the strain gauge.

Strain gauge [28] is a transducer that measures strain on an object. Most common type of strain gauge measures the deformity of an object by using a foil stuck to the surface of the object. As the object gets deformed, a proportional deformity is caused in the foil. This deformity reflects as a change in resistance of the foil, measuring which one can estimate the strain on an object.

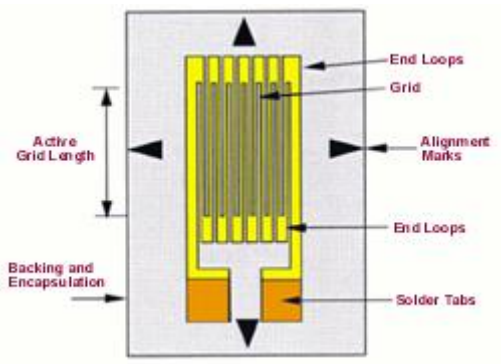


FIGURE 45: STRAIN GAUGES

Four strain gauges connected in a Wheatstone bridge configuration and placed around a metal beam form what is known as a load cell. The forces applied on the load cell deform the strain gauges, thus, disturbing the balance of the bridge. As a result, a small voltage (in order of few millivolts) appears at the balance arms of the bridge, this voltage is proportional to the force applied to the load cell. Thus, the load cell can be used to measure weight.

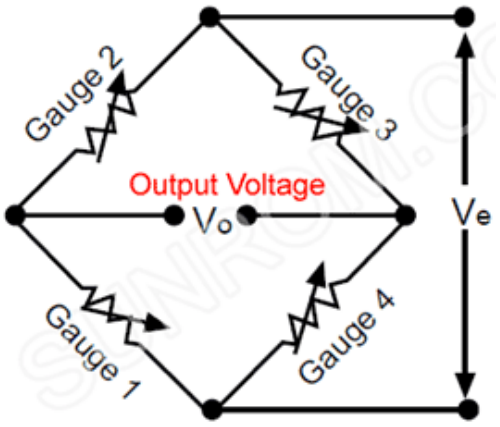


FIGURE 46: LOAD CELL

Typically, the connections to a load cell are colour coded and 5 in number. Red for Excitation Voltage, Black for ground, Green for output voltage (+) and White for

`

output voltage (-) and Yellow for shielding which is grounded to prevent noise interference.

5.2.2.2 CIRCUIT DESCRIPTION

As mentioned earlier, the output of a load cell is typically in the range of millivolts. As a result we would require a high resolution differential ADC in order to measure such small variations. We opted for HX711 [29] load cell sensor, which is a two channel 24-bit 80SPS ADC with internal PGA with gains of 32, 64 and 128. The maximum swing the ADC can measure is 80mV Thus, the minimum voltage the ADC is capable of measuring at its channels is about 4 nV.

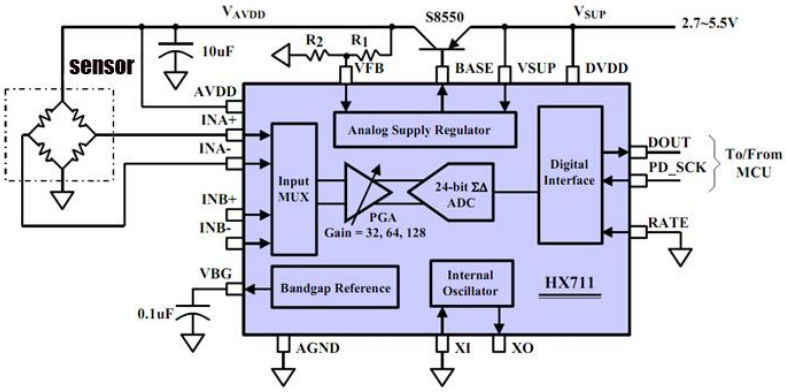


FIGURE 47: INTERFACING HX711 WITH LOAD CELL

The measurement technique requires the user to first calibrate the load cell with the ADC, to determine the proportionality constants, namely Scale and Offset. For this, the ADC measures the average of 10 reading when the platform is empty, to obtain the offset. After measuring the offset, a 1kg weight is placed on the platform and again measurement is performed to obtain the scale.

The HX711 communicates using a custom 2-wire protocol using pins DOUT and SCK. The communication was performed by bit-banging the GPIOs using a custom python library. Since the sampling rate of HX711 is quite low at 80SPS, operating system overheads do not affect much in the rate of sampling if it is done using a high level language construct.

`

5.2.2.3 HX711 CIRCUIT CONNECTIONS

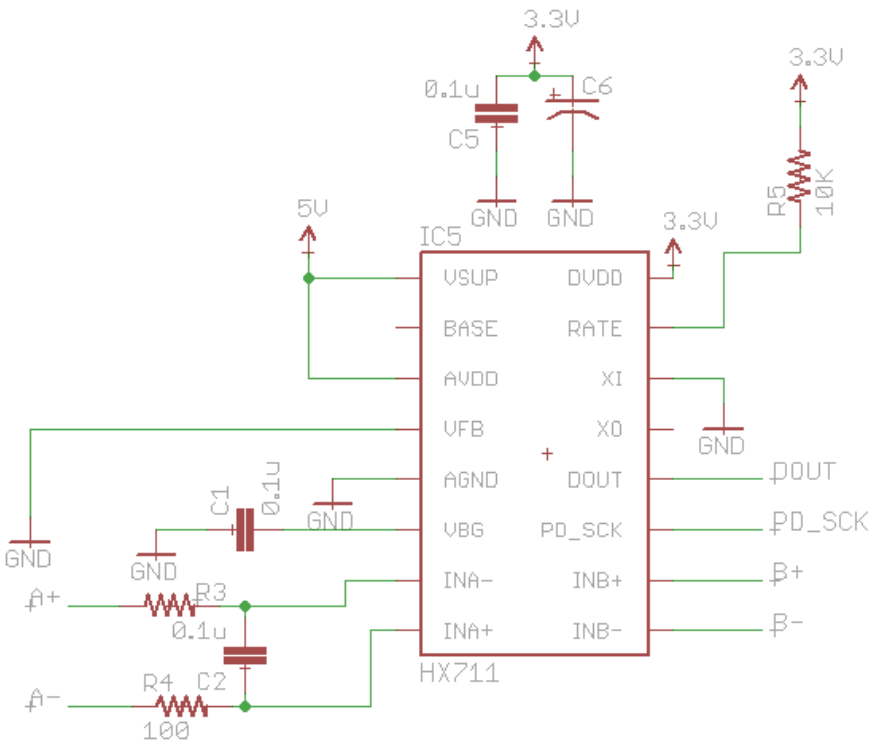


FIGURE 48: HX711 CONNECTIONS

5.3 STORAGE AND ANALYSIS

Besides making the Thermometer and Weighing scale real-time instruments, to just show the current values, we also incorporated the facility of saving the readings in a CSV file, timestamped using the Real-Time Clock (PCF8563) provided. The major advantage such an approach provides is the ability to refer to the readings later and perform appropriate analysis for experiments. This also reduces the human intervention in the experiment and hence the observation errors. For example, let us take a gravimetric analysis. Suppose you need to analyse a reaction where the rate change of sample weight is required. You can setup the reaction and place it on the platform and start saving the data. You need not intervene in the meantime unless the experiment demands so. Later on, when the reaction is complete you can read the CSV file and perform the required analysis to obtain the desired results.

`

CHAPTER 6: PHOTOMETER

All of us have sometimes looked up at the stars and wished to be astronomers in our own right. Looking through the telescopes at the stars is definitely fascinating, but there is much more to the astronomy than just that. An important part of astronomical observations is measuring the energy output of the celestial objects. This data is useful in determining the motion, distance and pathways of the objects in the sky. This measurement of the energy output is done by measuring the brightness of the objects in the night sky, which would be directly proportional to the energy output by the object (direct or indirect). In this chapter, we will revisit this age old measurement technique and modernize it.

6.1 PHOTOMETRY

Simply put, photometry is the science of the measurement of light intensity. The radiations from various celestial objects fall under certain wavelength bands, which then determine the method and analysis to perform on the data obtained. The instrument used for this is called Photometer [30]. We would be focusing on Visible Light Photometry.

FIGURE 49: IUCAA PHOTOMETER

A Photometer consists of a photodiode (PIN) which provides small currents proportional to incident light. These currents are in the order of a few microamperes (μA).

The following table provides an indication of the output current of PIN diode under various light settings.

TABLE 5: SAMPLE CURRENTS IN PHOTOMETER

Environment	Illumination $(\frac{lm}{ft^2})$	Luminous Intensity (lux) $\frac{uW}{mm^2}$	Power	Short Circuit Current
Direct Sunlight	1000	16.1	288.82 uW	173.3 uA
Overcast day	100	1.61	28.8 uW	17.33 uA
Twilight	1	0.016	0.288 uW	0.173 uA
Full moon	0.1	1.61	28.8 nW	17.28 nA
Clear night	0.001	16.1	288.8 pW	0.173 nA

`

This current is then amplified using a Trans impedance amplifier to a measurable voltage. The photometer typically operate on a +9V supply and the output of the photometer varies from +9V to -9V depending on the incident light. The photometer thus has effectively 4 terminals: Two for Power Supply (+9V and GND) and two for the output of the photometer (VOUT and GND).

6.1.1 ASTRONOMICAL OBSERVATIONS USING PHOTOMETER

Enough about the photometer! Let's now see how the cool astronomers use the photometer to obtain the reading [31]. An important point to note that while observing the night sky, the ambient lights also affect the readings, introducing major errors. While this error is largely unavoidable, steps are taken to minimize it. Before starting the experiment, the photometer is kept on a horizontal surface and the tube of the photometer is covered tightly and a reading (Ro) is taken from the output terminals using a mustimeter. This constitutes a dark reading. The dark reading corresponds to the error that may be introduced in the photometer readings due to leakage of light through the encasing of the photometer and other manufacturing offsets. Next, the telescope through which the readings are to be taken is first setup at the desired location and pointed towards a dark spot in the sky. The eyepiece of the telescope is replaced with the photometer and the output is measured after 20 seconds for every 5 seconds and the average is taken (Rb). The difference of the two readings is then taken. This value is known as sky brightness reading.

$$(\text{Sky Brightness}) = Rb - Ro \quad (1)$$

This reading accounts for the errors introduced in the readings due to the ambient sky brightness. This sky brightness is relatively high in urban areas and low in rural areas. Moreover, this sky brightness is not constant, it varies slowly with time. The cause of sky brightness is fourfold:

1. Extraterrestrial radiation, originating from the solar system and outer regions of Space.

2. Radiation emitted by excited atoms and molecules in the atmosphere

3. Radiation from other celestial bodies scattered by the atmosphere.

4. Man-made lights present in the surroundings.

After calculating the sky brightness reading and the dark reading, the telescope is focused on the target celestial body and the photometer is attached and the readings are taken (Rs). Then the telescope is pointed about 10° away from the star and another reading is taken (Rbs). The difference of the two readings is known as star reading.

$$(\text{star reading}) = Rs - Rbs \quad (2)$$

`

This process is repeated over the course of the experiment which can span about 10-12 hours, and the results are tabulated for later analysis.

6.2 LIMITATION OF EXISTING TECHNIQUES

It must be evident to you that the process used currently, as discussed above, is quite cumbersome. Practically, the readings cannot be taken at more than 1 per minute or so. Also, measurements using a multimeter, which use 12-bit ADCs, are bound to introduce errors in the readings. Hence, this technique can be improved a lot. We propose a design that would address all these issues along with satisfying all the requirements.

6.3 PROPOSED DESIGN

Here, we propose a boosterpack for विज्ञान हथेली पे, which interfaces with the Photometer. This design uses HX711 which is a 24-bit 80SPS analog-to-digital converter, with a maximum allowable swing limited to ± 12mV using the following equation:

$$\text{Swing} = \pm 0.5 \left(\frac{AVDD}{GAIN}\right) \quad (2)$$

Where AVDD = 3.3V and GAIN is set to 128. The common mode voltage should be between 1.2V and AVDD-1.3V.

Besides, the Photometer would not require an external +9V source, but using a power supply consisting of a boost stage which converts +5V to +12V and then a linear voltage regulator to obtain +9V from +12V, one can power the photometer from the boosterpack itself.

The ADC, as shown above, accepts an input voltage swing of ±12mV about a common-mode voltage between 1.2V and 2V. However, as described above, the Photometer output has a swing of ±9V. Thus, there arises a requirement for a signal conditioning stage in between. Moreover, if the signal conditioning circuit designed is active, then it would also require a dual-mode supply which would make the circuit complex. In order to simplify the design, the signal conditioning circuit was designed to be completely passive using precision resistances (1% tolerance).

This approach is advantageous in the way that the measurement can be taken and recorded in an accurate and automated fashion using the Raspberry Pi. Moreover, one can also plan out an observation for a particular number of samples, or for a particular duration, starting at some future point of time or at a particular trigger. The analysis of the readings need not be postponed and could be done immediately after all the samples have been taken.

`

6.3.1 CIRCUIT DESCRIPTION

The signal conditioning circuit that we use is completely passive in nature. It is a two stage circuit. The first stage is a potential divider made up of resistors 100Ω and 100kΩ (R11 and R12 respectively). This divides the input voltage, which is the photometer output (J1) by a factor of 1000. So, a ±9V signal is dropped to ±9mV signal. The current flowing through this divider is about 90µA.

This signal, albeit in a range acceptable by HX711, the common mode input voltage is 0V which is not compatible with it. So the second stage added is a passive summer made up of R10 and R9. The resultant output voltage will be:

$$V_+ = 0.5\,(V_1 + V_2) \tag{3}$$

Where V_1 is the voltage across R11 and V_2 is the voltage output of the opamp. The opamp is configured as a voltage buffer for a voltage reference LM336. The voltage is maintained at 2.5V. As a result, the voltage V_+ varies as ±9mV about 1.25V. The resistances R7 and R8 form a potential divider to maintain the voltage V_- at 1.25V. Thus the differential voltage output across V_+ and V_- is proportional to the output voltage of the Photometer.

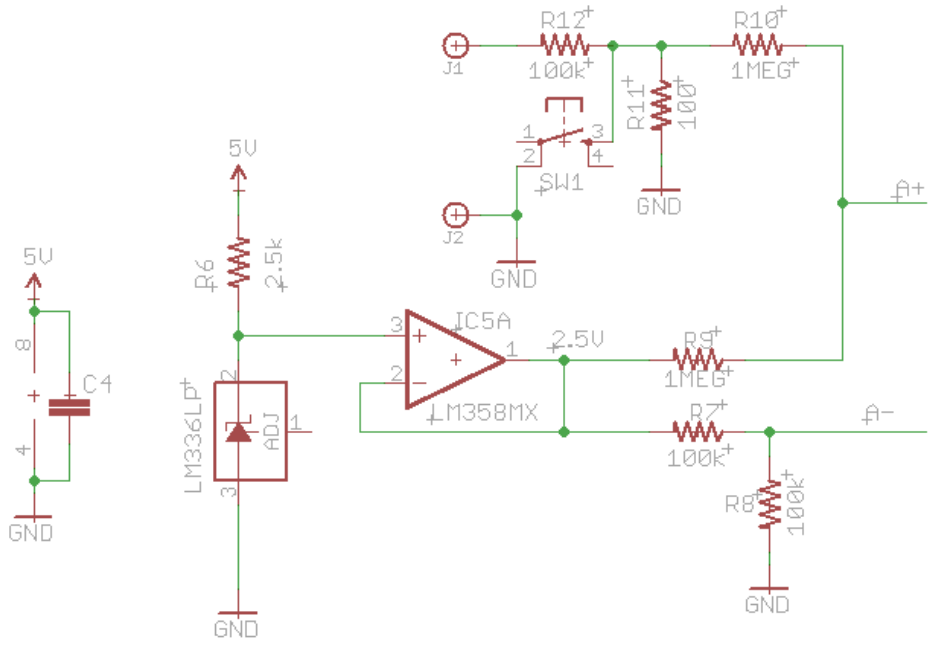


FIGURE 50: SIGNAL CONDITIONING CIRCUIT FOR PHOTOMETER

`

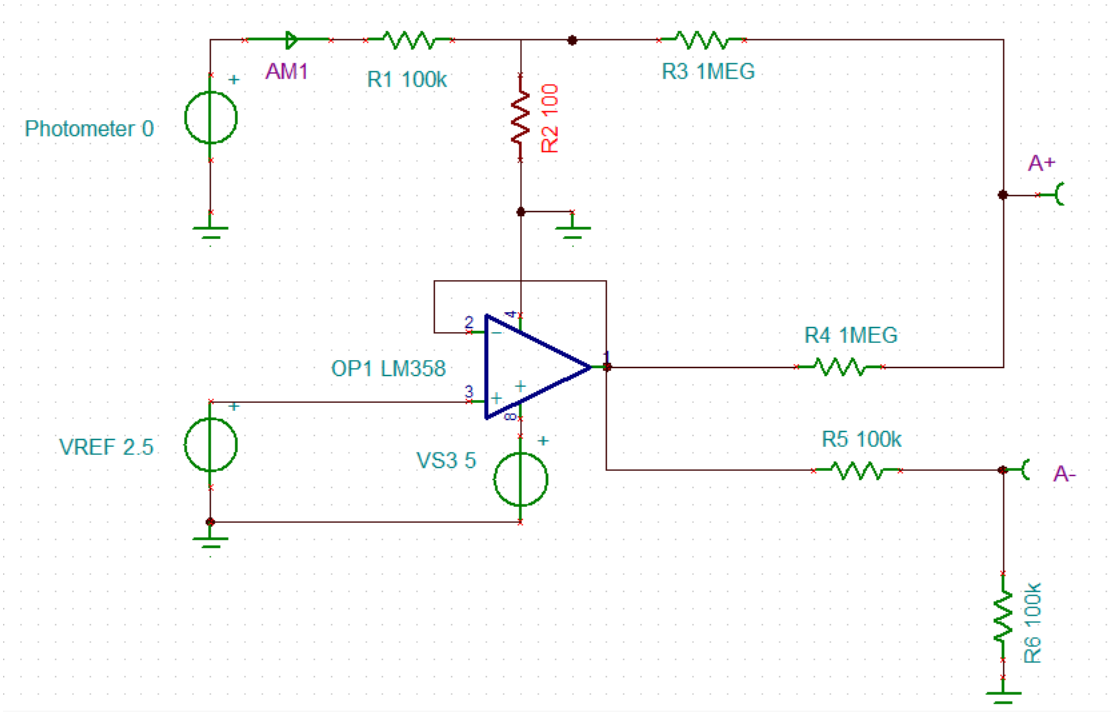


FIGURE 51: SPICE MODEL FOR CONDITIONING CIRCUIT

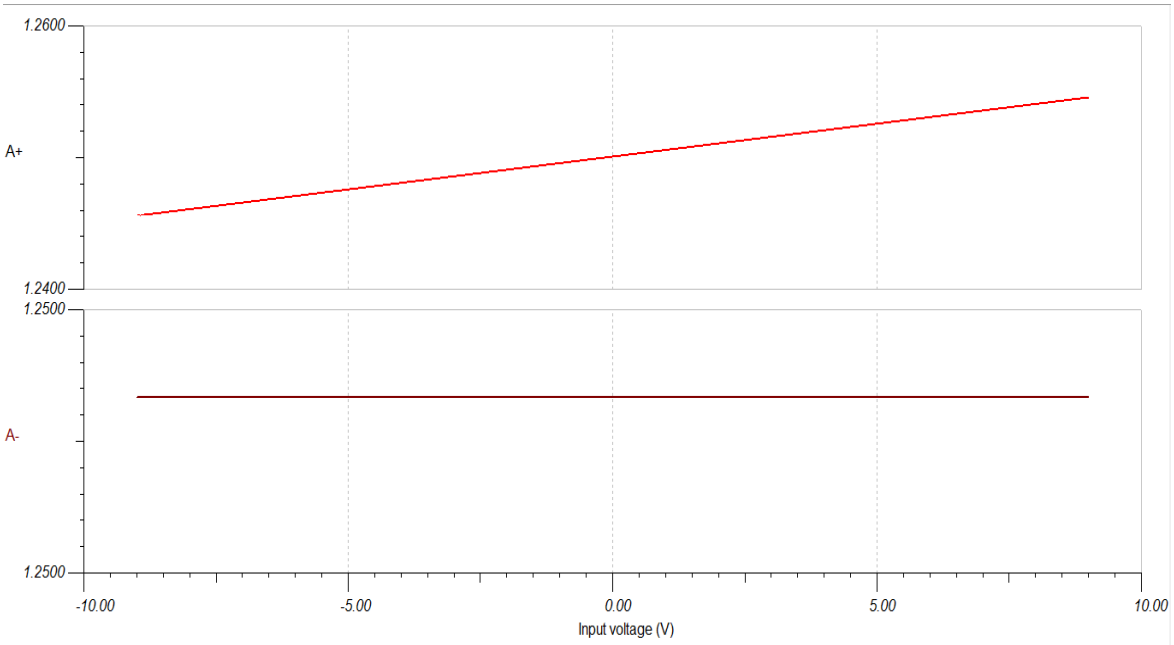


FIGURE 52: DC ANALYSIS SHOWING LINEAR OPERATION OF CONDITIONING CIRCUIT

`

6.4 SCHEMATIC

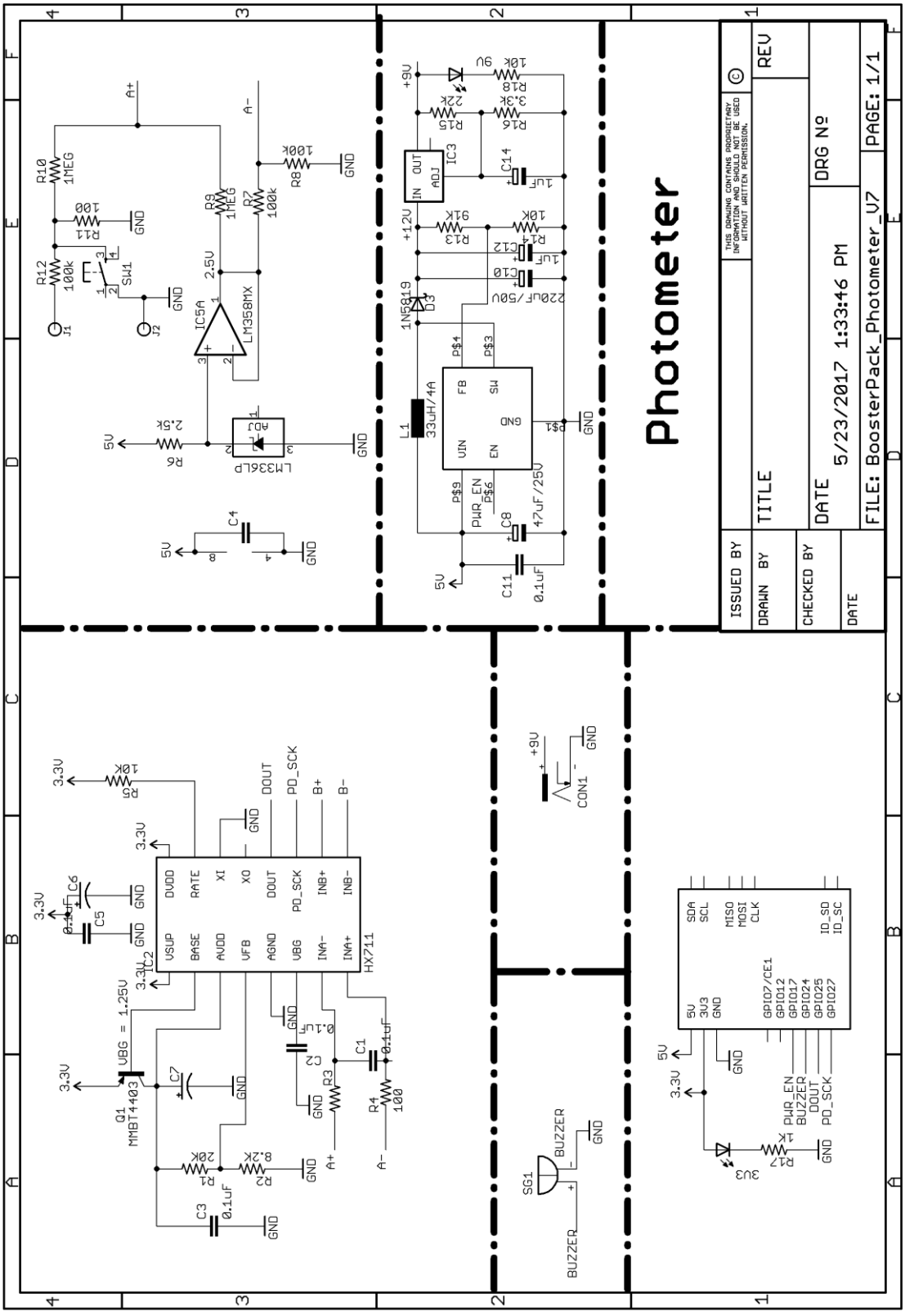


FIGURE 53: PHOTOMETER BOOSTERPACK SCHEMATIC

`

6.5 BOARD LAYOUT

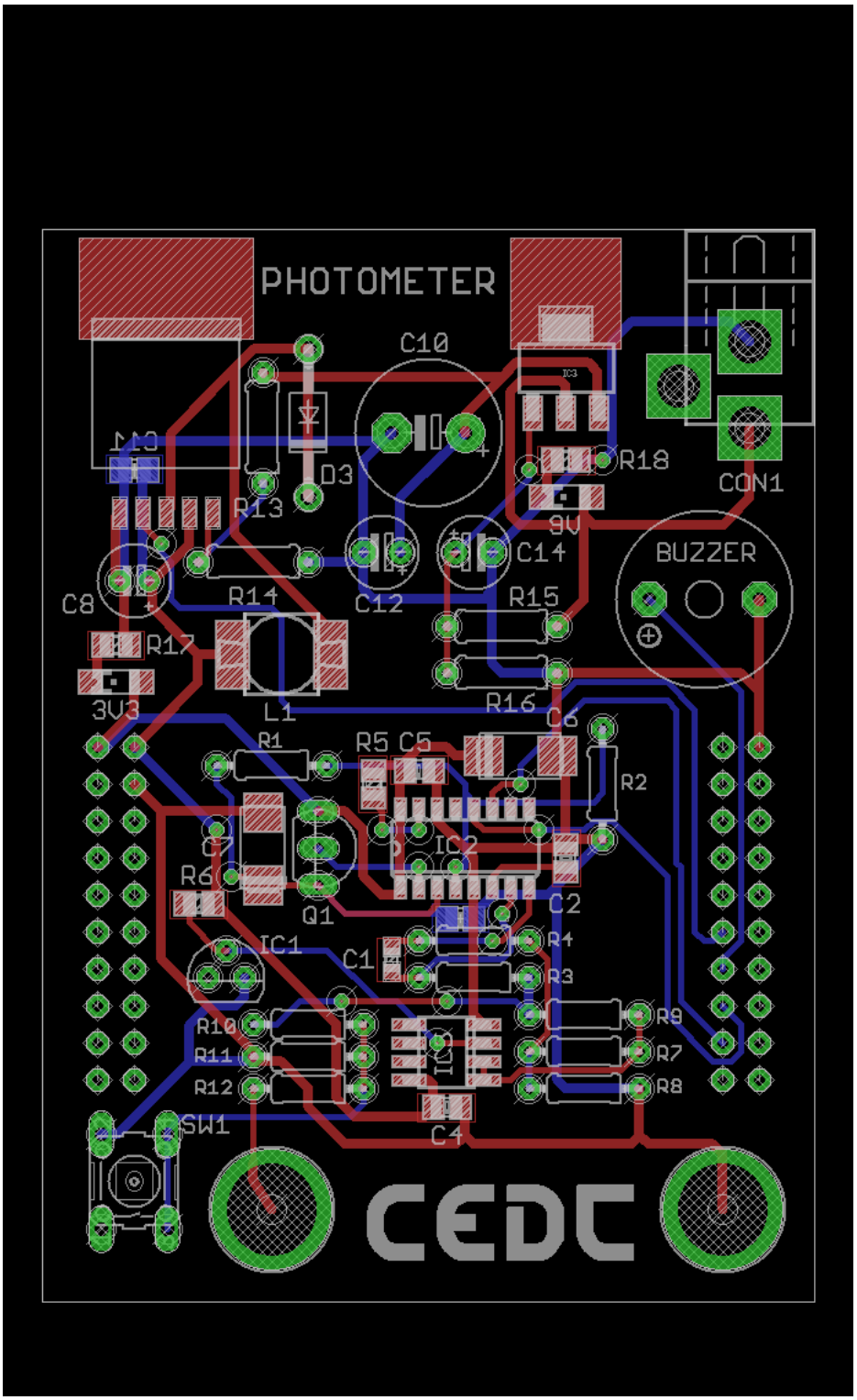


FIGURE 54: PHOTOMETER BOOSTERPACK LAYOUT

`

`

CHAPTER 7: MILLIOHMMETER

The last one in the series of the scientific applications we achieved using विज्ञान हथेली पे is this small, cute device which performs just one function, measuring resistances and that too of the order of milliohms, making the title of this boosterpack self-explanatory.

The flow of this chapter will take you to the first and the foremost doubt that how is this different from the conventional multimeter, then discussing about the difference in techniques of measurement on the circuit level. Then finally we will move onto the ADC part, since the ADC which we are going to use is same as the one discussed in the "Talking weighing scale" in the previous chapter, we won't repeat it here.

7.1 WHY NOT A MULTIMETER FOR MILLI-OHMS?

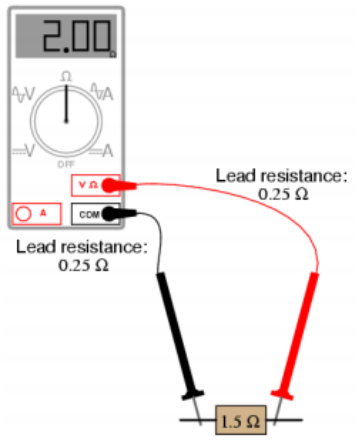


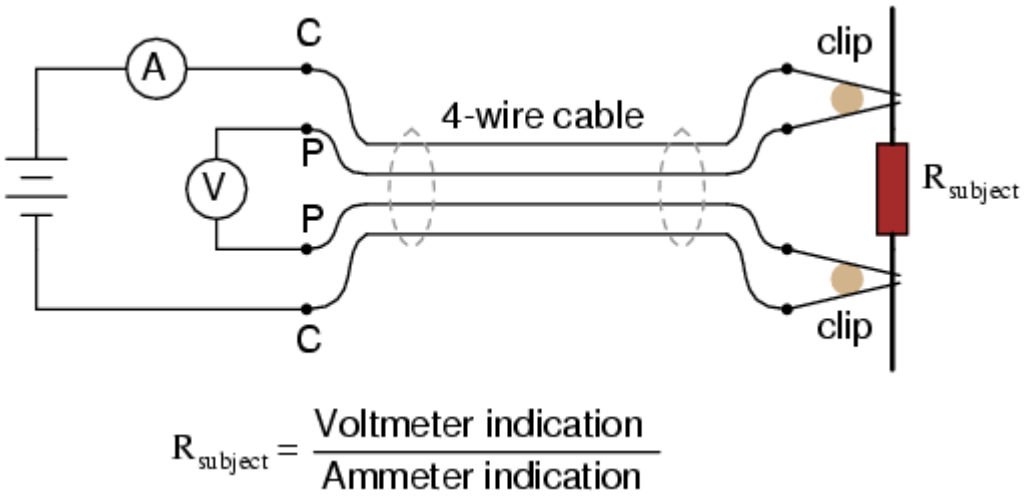
FIGURE 55: CONVENTIONAL–PROBE METHOD[32]

Measurement of low resistances is a tough nut to handle using the Digital Multimeters as two wire method is not recommended for measuring resistances below 100 ohms. This is because, the resistance of the probes or the measuring circuitry is comparable to the resistance of the load itself, thereby giving inaccurate value. The probe resistances will be added up in the resistance which is measured. To overcome this, numerous methods have been devised. One method is to use an accurate but expensive DMM with higher resolution or an LCR Meter which is even more expensive and offers many other amazing features. Here, we propose a much simple solution, cheaper than these solutions which just solves the purpose of measuring low resistances. This Milli-Ohmmeter uses the four-probe principle wherein 4 probes are used in the place of 2 probes of a conventional multimeter.

Let us now discuss the Four-Probe Method [33].

`

7.2 THE FOUR PROBE METHOD



$$R_{subject} = \frac{\text{Voltmeter indication}}{\text{Ammeter indication}}$$

FIGURE 56: WIRING USED IN THE 4-PROBE METHOD

In this method, the current is supplied via a pair of source connections (current probes). These generate a voltage drop across the impedance to be measured according to Ohm's Law, V=IR. A pair of sense connections (voltage probes) are made immediately adjacent to the target impedance, so that they do not include the voltage drop in the force leads or contacts. Since almost no current flows to the measuring instrument, the voltage drop in the sense leads is negligible. And therefore, the resistance is measured accurately.

But, how did we create a current source?

Please turn the page to understand!

`

7.3 THE CURRENT SOURCE

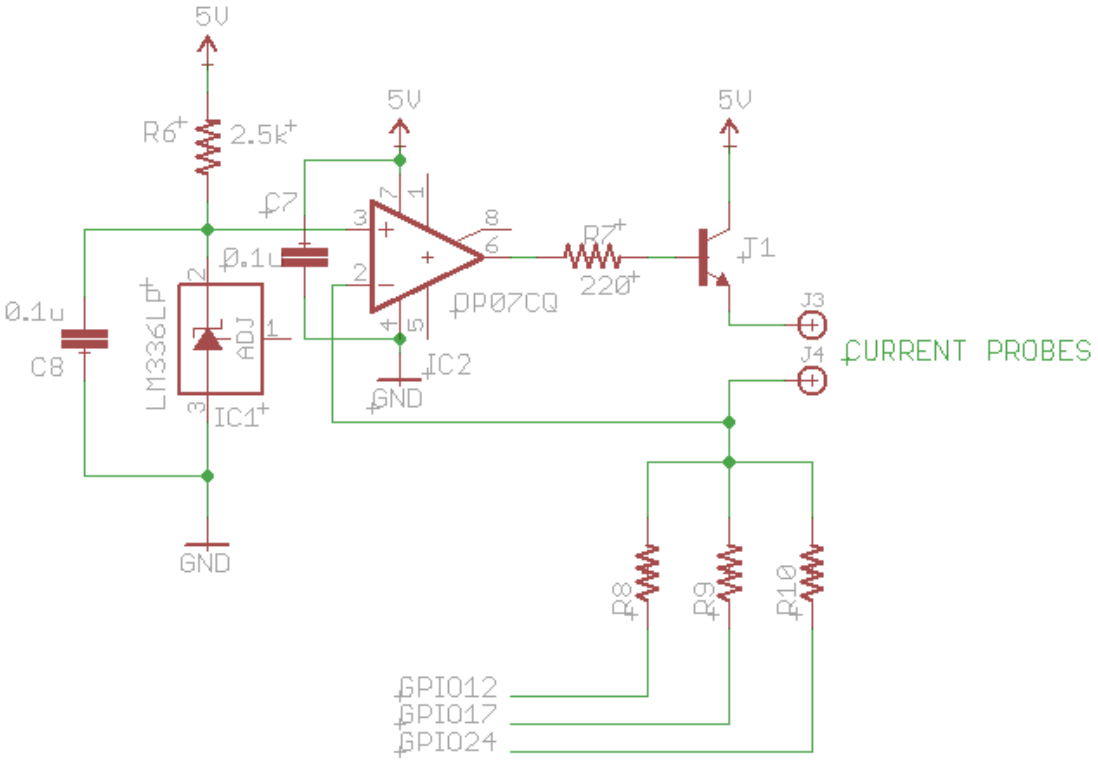


FIGURE 57: CURRENT SOURCE CIRCUIT

The current source supplying the current and the voltage measurement circuit using an analog to digital converter form the core of the resistance measurement circuitry. Ideally, a current source should deliver constant current irrespective of the load. The current source circuit which we have implemented, involves fixing the current through the emitter of a transistor. The circuit essentially utilizes an op-amp driven by a constant 2.5 V voltage reference LM336 applied at its non-inverting input. The sense resistor (R8, R9, and R10) which are connected to the emitter terminal of the transistor has a fixed voltage drop across it, by virtue of being connected to the inverting input of the op-amp. This decides the current through this resistor and hence through the load resistance as well. As a result, the current flowing through the load is constant as well, being numerically equal to the reference voltage divided by the sense resistor. For the three different ranges, we have used three different sense resistances. The three different current sources correspond to three different resistance ranges. The nodes between which the resistance to be measured have been connected, are brought out as terminals of the current probes. The voltage probes connected to the load resistance are connected to the Channel B input terminals of the HX711 ADC after being applied to the voltage buffers to stop any current flow in the voltage measuring circuit.

`

7.3.1 INTERFACING HX711

Refer to the appendix for the sample code for interfacing HX711 with Raspberry Pi.

7.4 SCHEMATIC

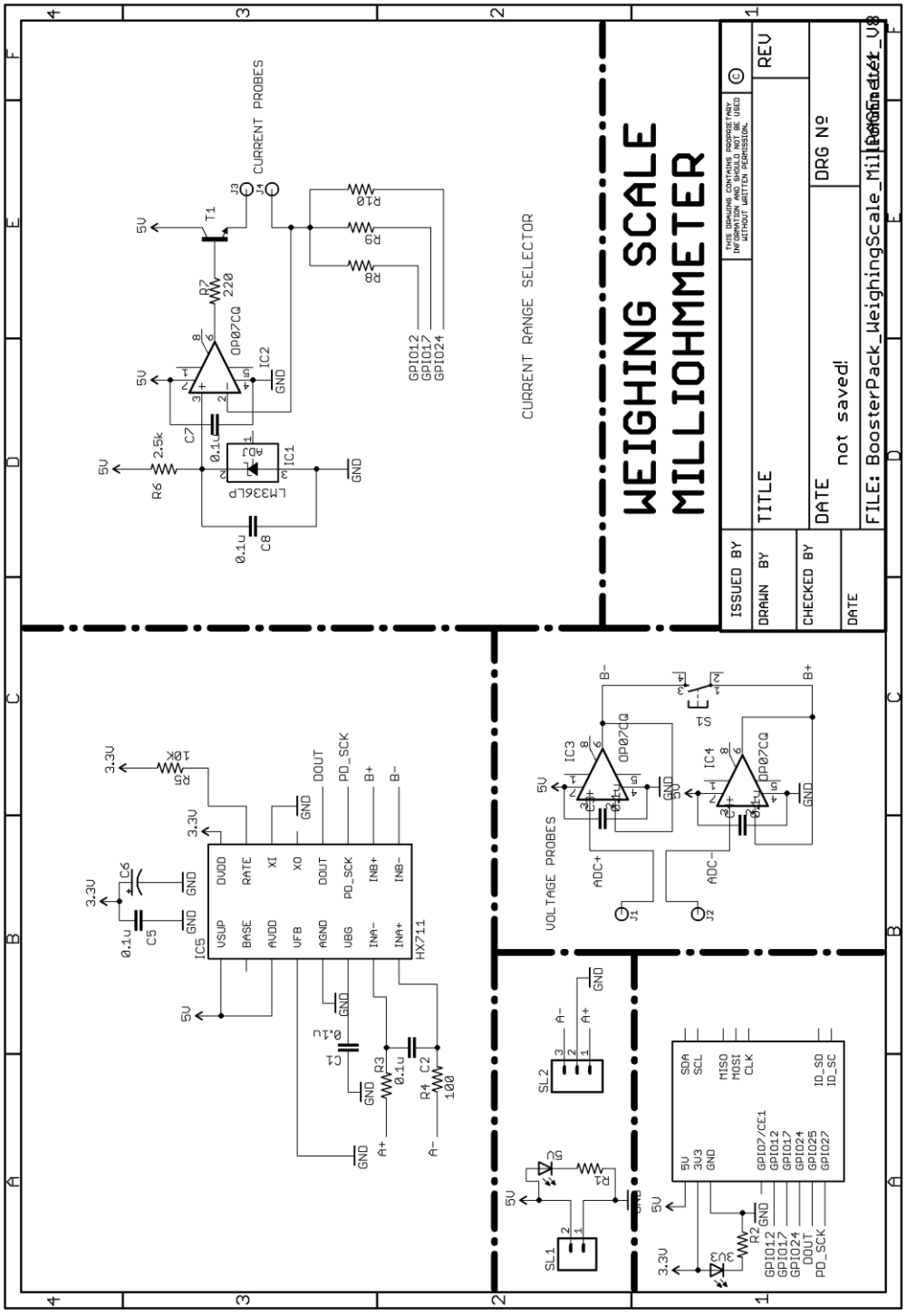


FIGURE 58: CIRCUIT SCHEMATIC FOR MILLI-OHMMETER

`

7.5 BOARD LAYOUT

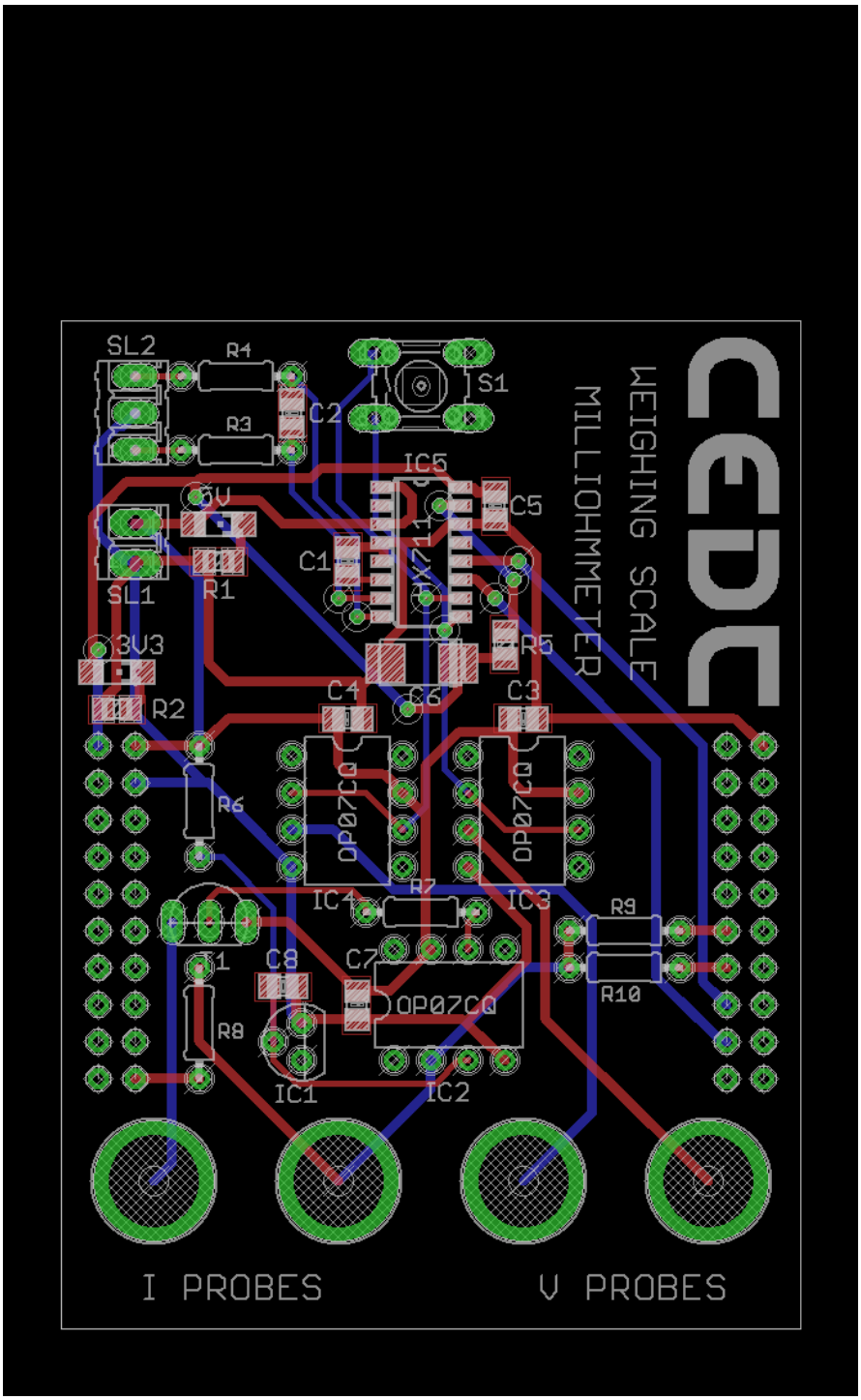


FIGURE 59: BOARD LAYOUT FOR MILLI-OHMMETER

`

We want the reader to know that since the ADC used in the Milli-ohmmeter and weighing scale is same (HX711) without any significant adaptations in wither of the applications.

7.6 BILL OF MATERIALS

TABLE 6: BILL OF MATERIALS FOR MILLI-OHMMETER AND WEIGHING SCALE BOOSTERPACK

Part	Value	Package	Description
3V3	LED1206	CHIPLED_1206	
5V	LED1206	CHIPLED_1206	
C1	0.1u	C0805	CAPACITOR
C2	0.1u	C0805	CAPACITOR
C3	0.1u	C0805	CAPACITOR
C4	0.1u	C0805	CAPACITOR
C5	0.1u	C0805	CAPACITOR
C6		C/6032-28R	
C7	0.1u	C0805	CAPACITOR
C8	0.1u	C0805	CAPACITOR
IC1	LM336LP	TO92	2.5-V INTEGRATED REFERENCE CIRCUITS
IC2	OP07CQ	DIL08	OP AMP
IC3	OP07CQ	DIL08	OP AMP
IC4	OP07CQ	DIL08	OP AMP
IC5	HX711	SO16	
J1		BANANA_CONN	Through-hole banana jack
J2		BANANA_CONN	Through-hole banana jack
J3		BANANA_CONN	Through-hole banana jack
J4		BANANA_CONN	Through-hole banana jack
R1		M0805	RESISTOR
R2		M0805	RESISTOR
R3	100	0204/7	RESISTOR
R4	100	0204/7	RESISTOR
R5	10K	R0805	RESISTOR
R6	2.5k	0204/7	RESISTOR
R7	220	0204/7	RESISTOR
R8		0204/7	RESISTOR
R9		0204/7	RESISTOR
R10		0204/7	RESISTOR
S1	SWITCH_10-XX	B3F-10XX	
SL1		02P	AMP QUICK CONNECTOR
SL2		03P	AMP QUICK CONNECTOR
T1		TO92-EBC	NPN Transistor
U$1		TIVA_LAUNCHPAD_FOOTPRINT	

`

7.7 PROTOTYPE PHASES

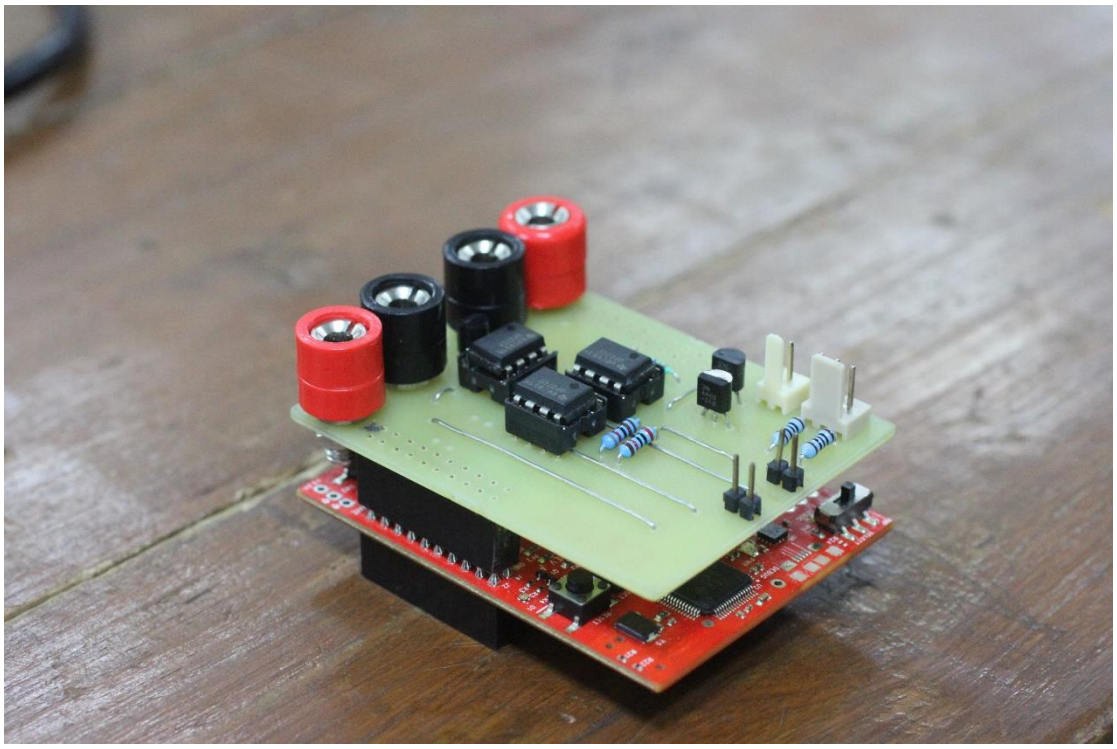


FIGURE 60: WEIGHING SCALE AND MILLI-OHMMETER BOOSTERPACK DEVELOPED IN CEDT

The initial prototype was developed in the lab, once the testing was done, the Gerber data was sent to the custom PCB fabricator.

`

`

CHAPTER 8: CONCLUSIONS

We are sincerely thankful for your patience to travel along with us in our journey to transform an idea into reality. **विज्ञान हथेली पे** helped us in converging five different scientific applications ranging from HF-SDR to Milli-ohmmeter, to astronomically indulging Photometer and lastly the exploration of talking paradigm implementing a talking weighing scale and talking thermometer.

You must have got a strong feeling that this journey would not stop here definitely and will go onto something even more creative and engrossing scientific applications. We will mention the possible applications in a short while.

8.1 JOURNEY TILL NOW

We are more excited about sharing with you the achievements we have had while developing **विज्ञान हथेली पे** . We are enlisting these here so as to make you even more excited about the number of lives we have touched and we plan to touch.

1. Our presentation on "**LOW COST OPEN SOURCE FRAMEWORK FOR MULTILINGUAL TALKING DEVICES**" have been selected to be presented at **INNOVATION & DO-IT-YOURSELF ELECTRONICS** (IDIYE 2017[34]) Conference organized by IEEE-CAS (Circuits and Systems), Bangalore Section.

2. A workshop was organized for Computer Science Teachers of Delhi Schools by **Google under their "CS for Schools"** initiative. We developed pedagogy material for Raspberry Pi helping the participants to learn the **basics of Physical Computing** and then implement their newly gained knowledge to code their way to a **talking thermometer.**

3. We are on our way to publish a Project Paper revolving around the core of this project.

These are the things which have kept us motivated by the impact this project can create right form the schools, to the engineering college and covering the industry, motivating the participants to take up the challenge of developing custom hardware for a diverse range of scientific applications.

`

8.2 TILL WHERE DO WE PLAN TO TAKE THIS?

Now we list the scientific applications which we would be aiming to converge upon so as to publish this work in the most professional form, book on scientific instrumentation around Raspberry Pi.

1. Lab Tools Instrumentation[35]
2. Red Pitaya[36]
3. Wind Vane/ Anemometer[37]
4. SatNOGS[38]
5. Benchtop Power supply[39]
6. Potentiostat[40]
7. Microscope setup[41]
8. Gas Arc Welding Monitor[42]
9. Microbalance[43]
10. Smartphone spectrometer[44]
11. Seismograph[45]
12. Low cost arduino instrumentation[46]
13. Smart Lamp[47]
14. Environment Monitoring System[48]
15. pH meter[49]
16. PQ Meter[50]
17. Microscope[51]
18. Photometer and Colorimeter[52]
19. Thermocycler[53]
20. Conductimeter[54]
21. 3 axis magnetic field sensor[55]
22. Attitude Sensor[56]
23. Visible Light Spectrometer[57]
24. Raman Spectrometer[58]
25. Fluorimeter[59]
26. Turbidimeter[60]
27. Colorimeter[61]
28. Curve Tracer[62]
29. Magnetometer[63]
30. Luxmeter[64]

Mentioning all these possible applications using **विज्ञान हथेली पे** we bid a good bye on a high note that whatever we learnt throughout the journey is going to make us stand firm for the rest of it. All that we talked about is going to taste good only when you try to cook new dishes every day using Raspberry Pi.

Happy Cooking! :D

`

REFERENCES

[1] https://beagleboard.org/black
[2] https://software.intel.com/en-us/iot/hardware/galileo
[3] https://www.raspberrypi.org/
[4] http://www.ti.com/lsds/ti/tools-software/launchpads/boosterpacks/about/about.page
[5] https://www.element14.com/community/docs/DOC-73950/l/raspberry-pi-3-model-b-gpio-40-pin-block-pinout
[6] http://www.sunrom.com/p/color-lcd-tft-2-2
[7] https://en.wikipedia.org/wiki/HDMI
[8] https://en.wikipedia.org/wiki/Pulse-width_modulation
[9] https://rydepier.wordpress.com/tag/ky-023/
[10] http://www.sunrom.com/p/5-way-tactile-switch
[11] http://www.nxp.com/documents/data_sheet/PCF8563.pdf
[12] https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter
[13] https://en.wikipedia.org/wiki/1-Wire
[14] https://en.wikipedia.org/wiki/I²C
[15] https://learn.sparkfun.com/tutorials/using-eagle-schematic
[16] www.arrl.org/what-is-ham-radio
[17] https://en.wikipedia.org/wiki/Software-defined_radio
[18] https://www.hfsigs.com
[19] https://www.dsprelated.com/showarticle/192.php
[20] www.norcalqrp.org/files/Tayloe_mixer_x3a.pdf
[21] https://datasheets.maximintegrated.com/en/ds/DS1085.pdf
[22] https://developer.chrome.com/apps/tts
[23] https://responsivevoice.org/
[24] https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf
[25] https://www.raspbian.org/
[26] https://www.kernel.org/doc/Documentation/w1/masters/w1-gpio
[27] idledeveloper.com/tag/w1-therm/
[28] https://en.wikipedia.org/wiki/Strain_gauge
[29] www.sunrom.com/p/loadcell-sensor-24-bit-adc-hx711
[30] https://en.wikipedia.org/wiki/Photometry_(optics)
[31] ojs.iucaa.ernet.in/index.php/khagol/article/viewFile/159/142
[32] https://en.wikipedia.org/wiki/Multimeter
[33] four-point-probes.com/four-point-probe-theory/
[34] http://ieeebangalore.org/index.php/2017/04/11/innovation-do-it-yourself-electronicsidiye-2017/
[35] http://www.lab-tools.com/instrumentation/
[36] http://redpitaya.com/
[37] http://www.instructables.com/id/Complete-DIY-Raspberry-Pi-Weather-Station-with-Sof/
[38] https://satnogs.org/documentation/hardware/

`

[39] https://www.envox.hr/eez/bench-power-supply/psu-introduction.html
[40] http://web.chem.ucsb.edu/~kwp/cheapstat/
[41] https://hackaday.io/project/5059-flypi-cheap-microscopeexperimental-setup
[42] http://www.appropedia.org/Low-cost_Open-Source_Voltage_and_Current_Monitor_for_Gas_Metal_Arc_Weld_3-D_Printing
[43] http://openqcm.com/
[44] http://www.upb.edu/en/contenido/smartphone-spectrometer
[45] https://www.kickstarter.com/projects/angelrodriguez/raspberry-shake-your-personal-seismograph
[46] http://file.scirp.org/pdf/MI20120200002_83441536.pdf
[47] https://www.ncbi.nlm.nih.gov/pubmed/26959035
[48] https://www.ncbi.nlm.nih.gov/pubmed/26053749
[49] http://pubs.acs.org/doi/abs/10.1021/ed5008102
[50] www.instructables.com/id/ARDUINO-ENERGY-METER/
[51] http://www3.eng.cam.ac.uk/news/stories/2013/Open-source_through_the_lens_of_a_microscope/
[52] http://www.instructables.com/id/An-Inexpensive-Photometer-and-Colorimeter/wet
[53] http://www.iisc.ernet.in/currsci/aug25/articles18.htm
[54] http://www.instructables.com/id/Conductivity-Meter/
[55] http://nora.nerc.ac.uk/513347/1/A%26G-2016-Beggan_final.pdf ; http://www.geomag.bgs.ac.uk/education/raspberry_pi_magnetometer.html
[56] http://www.worldscientific.com/doi/abs/10.1142/S2251171714400066
[57] http://www.isp.uu.se/digitalAssets/238/c_238615-l_1-k_new-methods-for-developping-scientific-instruments.pdf
[58] http://www.isp.uu.se/digitalAssets/238/c_238615-l_1-k_new-methods-for-developping-scientific-instruments.pdf
[59] http://www.isp.uu.se/digitalAssets/238/c_238615-l_1-k_new-methods-for-developping-scientific-instruments.pdf
[60] http://www.isp.uu.se/digitalAssets/238/c_238615-l_1-k_new-methods-for-developping-scientific-instruments.pdf
[61] http://www.appropedia.org/Open-source_colorimeter
[62] http://www.open-sensing.org/windvaneblog/
[63] http://www.geomag.bgs.ac.uk/education/raspberry_pi_magnetometer.html
[64] http://www.mdpi.com/1424-8220/15/6/13012/htm

`

APPENDIX

Here we provide to you the codes we developed for **विज्ञान हथेली पे**

A. INTERFACING DS1085 FOR SDR

The following code interfaces a TIVA TM4C123G Launchpad with the DS1085 based SDR Boosterpack

```
/*
 * Objective : Program for Software Defined Radio (BoosterPack version)
 *
 * This code interfaces a DS1085LZ-25 to clock a Tayloe detector which samples the I and Q of the
 * Signal received through an antenna. These samples are then sent to a software to be processed
 *        (Software :  Quisk SDR (Linux) )
 *
 * Author:       Gaurav Tyagi, Anshuman Mishra
 */

/******************** INCLUDES **********************/
#include <stdint.h>
#include <stdbool.h>

#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h"
#include "inc/hw_types.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/i2c.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "inc/hw_ints.h"
#include "driverlib/systick.h"
#include "inc/hw_i2c.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.c"

/************* CONSTANTS AND DEFINES ****************/

#define SLAVE_ADDRESS 0x58              //The default address of the DS1085LZ-25

int currentRange = 0;                   //Tracks the frequency range that can be currently recieved by the hardware
unsigned char setting  = 0;             //Setting of the N divider
```

`

```
/************ FUNCTION DECLARATIONS ****************/

unsigned char setN(int,int);    //Calculates the required value of the N divider of the DS1085LZ-25
int getSet(int,int);       //Gets the setting range according to the requirements
void InitConsole();              //Configures UART for debugging purposes
void nextRange();                //Cycles through the various ranges
void setDiv(unsigned char);   //Sets the DIV register of the DS1085LZ-25
void seeStatus();         //Gets the status of all the registers in the DS1085LZ-25


/***************** MAIN FUNCTION ********************/

int main(void) {
        unsigned char OS[2];
        unsigned char OSVal = 0;
        SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);                  //Configure system clock to 16MHz
        InitConsole();  //Initializes UART0 Module and configures it.
//      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);        //Enable Port B
        SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);          //Enable I2C1 Module
        GPIOPinConfigure(GPIO_PA6_I2C1SCL);          //Configure PA6 as SCL of I2C1
        GPIOPinConfigure(GPIO_PA7_I2C1SDA);          //Configure PA7 as SDA of I2C1
        GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_6 );
        //Enable GPIO Output on PA6
        GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_7 );
        //Enable GPIO Output on PA7
        GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_7);            //Enable I2C Module output to PA7
        GPIOPinTypeI2CSCL(GPIO_PORTA_BASE,GPIO_PIN_6);
        //Enable I2C Module output to PA6
        I2CMasterInitExpClk(I2C1_BASE, SysCtlClockGet(), false);       //Enable Clock to I2C1 Module
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Set the device Slave Address
        UARTprintf("Initialized!!\n");                  //Notify user that Initialization has been Completed via UART
        seeStatus();    //Displays the status of DS1085LZ-25 on UART
        UARTprintf("Attempting read of RANGE register\n");
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Set the I2C1 module in TX mode to DS1085LZ-25
        I2CMasterDataPut(I2C1_BASE, 0x37);
                //Puts the RANGE register address on bus
```

`

```
        UARTprintf("Sent Range Address: %02x\n",0x37);
        //Keeps User updateds
        I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);
        //Starts sending data
        while(I2CMasterBusy(I2C1_BASE));                //Wait till the operation is
completed
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);
        //Start I2C RX from DS1085LZ-25
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_START);                //Start Recieving
in burst mode
        while(I2CMasterBusy(I2C1_BASE));                //Wait till the operation is
completed
        OS[0] = I2CMasterDataGet(I2C1_BASE);
        //Get first byte of RANGE (MSByte)
        UARTprintf("MSB Recieved : %02x\n",OS[0]);
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);
        //Recieve data from DS1085LZ-25
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_FINISH);       //Get last byte from the
burst mode data
        while(I2CMasterBusy(I2C1_BASE));                //Wait till the operation is
completed
        OS[1] = I2CMasterDataGet(I2C1_BASE);
        //Get 2nd byte of RANGE (LSByte)
        UARTprintf("LSB Recieved : %02x\n",OS[1]);
        OSVal  = OS[0]>>3;  //Get the 5 MSBs of MSByte - to extract the actual data
of frequency range
        OSVal +=5;                      //Set OSVal = OS+5   - To get the desired
frequency range (54.4MHz - 80MHz)
        UARTprintf("OSVal Calculated : %02x\n",OSVal);
        UARTprintf("Finished Accessing RANGE Register \n");
        UARTprintf("Attempting set of OFFSET register");
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Transmit a byte to DS1085LZ-25
        I2CMasterDataPut(I2C1_BASE, 0x0e);
                //Put OFFSET Register address on the bus
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_SEND_START);   //Start Sending data in burst
mode
        while(I2CMasterBusy(I2C1_BASE));                //Wait till the operation is
completed
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Transmit a byte to DS1085LZ-25
        I2CMasterDataPut(I2C1_BASE,OSVal);
                //Put OS+5 on the bus
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_SEND_FINISH);   //Start TXing
        while(I2CMasterBusy(I2C1_BASE));                //Wait till the operation is
completed
```

`

```
        UARTprintf("Finished Accessing OFFSET register");
        seeStatus();

        UARTprintf("Attempting setting of DAC register\n");
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Transmit a byte to DS1085LZ-25
        I2CMasterDataPut(I2C1_BASE, 0x08);
                //Put DAC register address on the bus
        UARTprintf("DAC address sent\n");
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_SEND_START);    //Start sending data in burst mode
        while(I2CMasterBusy(I2C1_BASE));                //Wait till the operation is
completed
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Transmit a byte to DS1085LZ-25
        I2CMasterDataPut(I2C1_BASE, 0xFF);
                //Set DAC to max value
        UARTprintf("Set MSB to 0xFF\n");
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_SEND_CONT);     //Send data in burst mode
        while(I2CMasterBusy(I2C1_BASE));                //Wait till the operation is
completed
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        I2CMasterDataPut(I2C1_BASE, 0xFF);
        UARTprintf("Set LSB to 0xFF\n");
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_SEND_FINISH);
        while(I2CMasterBusy(I2C1_BASE));
        UARTprintf("DAC Setting Finished\n");
        seeStatus();

        // Here the DS1085L-25 has been set to 80Mhz MCO Operation
        // Now Setting up the interrupts

        GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_4);
        //Set PA4 as GPIO Input
        GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_5);
        //Set PA5 as GPIO Input
        GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_2);
        //Set PB2 as GPIO Output
        GPIOIntRegister(GPIO_PORTA_BASE, nextRange);
        //Register nextRange as the ISR for Port A
        GPIOIntClear(GPIO_PORTA_BASE, GPIO_PIN_4|GPIO_PIN_5);
        //Clear existing Interrupts if any
        GPIOPadConfigSet(GPIO_PORTA_BASE, GPIO_PIN_4|GPIO_PIN_5,
GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_STD_WPU);                //Configure
the pins PA4 and PA5 as Weak Pull ups
        GPIOIntTypeSet(GPIO_PORTA_BASE, GPIO_PIN_4|GPIO_PIN_5,
GPIO_FALLING_EDGE);             //Set interrupt to trigger at Falling Edge
```

`

```
        GPIOIntEnable(GPIO_PORTA_BASE, GPIO_PIN_4|GPIO_PIN_5);
        //Enable the interrupts on PA4 and PA5
        IntEnable(INT_GPIOA);       //Enable Interrupts on GPIO Port A
        IntMasterEnable();      //Enable global Interrupts
        //Configuration Done. Now putting the processor on to a Wait state
        while(1);
        return 0;
}

/************ FUNCTION DEFINITIONS ****************/


unsigned char setN(int Coarse_Set, int Fine_Set)
{
   unsigned char setRange = Coarse_Set*5 + Fine_Set+9;              //Calculate the
range setting according to the given values of Coarse Set and Fine Set
//  cout<<"Coarse : "<<Coarse_Set<<" Fine : "<<Fine_Set<<" setN : "<<setRange;
   int byte = setRange << 6;              //Shift the Range value to align according to the
requirements of DS1085LZ-25
   UARTprintf("\nCoarse : %d , Fine : %d, Setting : %d  -> %02x \n
",Coarse_Set,Fine_Set,setRange,byte);
   return setRange;      //return the calculated value
}

int getSet(int readVal,int maxVal)
{
        int step = maxVal/5;   //Divide the maxVal by 5 to obtain the step size
        int Set = readVal/step + 1;                //Calculate the setting value
        return Set+1;
}

void InitConsole(void)
{
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);                  //Enable
Port A
        SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);                  //Enable
UART0 Module
        GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
        //Configure PA0 and PA1 as UART pins
        UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC); //Configure
UART0 Clock source as the Precision Internal Oscillator
        UARTStdioConfig(0, 115200, 16000000);              //Configure UART0 to
run at 16MHz and a baudrate of 115200
}

void nextRange(void)
{

        GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2,0);     //Set PB2 at LOW
(LED Off)
```

`

```
        if(GPIOIntStatus(GPIO_PORTA_BASE,true)==GPIO_PIN_4)             //If
interrupt is generated by PA4
        {
                currentRange++;                 //Increment the current range
                if(currentRange>25)             //If current range goes beyond 25
                {
                        currentRange =25;       //Set it at 25
                        GPIOPinWrite(GPIO_PORTB_BASE,
GPIO_PIN_2,GPIO_PIN_2); //Set PB2 at HIGH (LED On)
                }
        }
        else if(GPIOIntStatus(GPIO_PORTA_BASE,true)==GPIO_PIN_5)        //If
interrupt is generated by PA5
        {
                        currentRange--;                 //Decrement the current range
                        if(currentRange<0)      //If current range goes less than 0
                        {
                                currentRange =0;        //Set current range to 0
                                GPIOPinWrite(GPIO_PORTB_BASE,
GPIO_PIN_2,GPIO_PIN_2); //Set PB2 at HIGH (LED On)
                        }
        }
        GPIOIntClear(GPIO_PORTA_BASE, GPIO_PIN_4|GPIO_PIN_5);
        //Clear the Interrupts on PA5 and PA4

        setting  = setN(currentRange/5, currentRange%5);   //Get the setting value to
be sent to DS1085LZ-25
        setDiv(currentRange+9);                 //Set the DIV value to the set value
        UARTprintf("Settings: %d : %02x \n",currentRange,setting);
        seeStatus();
}

void setDiv(unsigned char a)
{
        unsigned char x = a>>2;                 //Obtains the MSB from the given setting
        unsigned char y = (a&0x03)<<6;                  //Obtains the LSB from the given
setting
        UARTprintf("\nAttempting DAC modification to %d = 0x%02x\n",a,a);
        UARTprintf("\n After modification x = %02x y = %02x\n",x,y);
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Transmit a byte to DS1085LZ-25
        I2CMasterDataPut(I2C1_BASE, 0x01);                  //Put DIV register address
on bus
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_SEND_START);   //Start sending the data in burst
mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Transmit a byte to DS1085LZ-25
```

`

```
        I2CMasterDataPut(I2C1_BASE, x);         //Put the MSB on the bus
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_SEND_CONT);    //Send the data
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Transmit a byte to DS1085LZ-25
        I2CMasterDataPut(I2C1_BASE, y);         //Put LSB on the bus
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_SEND_FINISH);          //Send the last byte of
data with stop bit
        while(I2CMasterBusy(I2C1_BASE));                //Wait till the operation is
completed

}

void seeStatus()
{
        UARTprintf("\n\n----------------------------------------\n");
        UARTprintf("STATUS of DS1085LZ-25\n----------------------------------------
\n\n");

        //*****************************RANGE*****************************
*******
        unsigned char OS[2];
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Transmit a byte to DS1085LZ-25
        I2CMasterDataPut(I2C1_BASE, 0x37);
                //Put RANGE register address on bus
        I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);
        //Start sending the data in single mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);
        //Receive data from DS1085LZ-25
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_START);        //Start receiving data in
burst mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        OS[0] = I2CMasterDataGet(I2C1_BASE);
        //Get first byte of RANGE (MSByte)
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);
        //Receive data from DS1085LZ-25
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_FINISH);       //Recieve the last byte of
burst mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
```

`

```
        OS[1] = I2CMasterDataGet(I2C1_BASE);
        //Get 2nd byte of RANGE (LSByte)
        UARTprintf("RANGE : %02x%02x\n OS : %02x\n",OS[0],OS[1],OS[0]>>3);
        //*************************************************************
*****
        //***************************OFFSET*****************************
********
        unsigned char OSVal = 0;
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Transmit a byte to DS1085LZ-25
        I2CMasterDataPut(I2C1_BASE, 0x0e);
                //Put OFFSET register address on bus
        I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);
        //Start sending the data in single mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);
        //Receive data from DS1085LZ-25
        I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
                //Start receiving data in single mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        OSVal = I2CMasterDataGet(I2C1_BASE);  //Get the OS Value
        UARTprintf("OFFSET : %02x\n OFFSET Value : %02x\n",OSVal,
OSVal&0x1f);
        //***************************DAC*******************************
*****
        unsigned char DAC[2];
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Transmit a byte to DS1085LZ-25
        I2CMasterDataPut(I2C1_BASE, 0x08);
                //Put DAC register address on bus
        I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);
        //Start sending the data in single mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);
        //Receive data from DS1085LZ-25
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_START);        //Start receiving data in
burst mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        DAC[0] = I2CMasterDataGet(I2C1_BASE);
        //Get first byte of DAC (MSByte)
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);
        //Receive data from DS1085LZ-25
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_FINISH);        //Receive the last byte in
burst mode
```

`

```
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        DAC[1] = I2CMasterDataGet(I2C1_BASE);
        //Get 2nd byte of DAC  (LSByte)
        UARTprintf("DAC : %02x%02x\n",DAC[0],DAC[1]);
        //*****************************N-
DIVIDER*********************************
        unsigned char DIV[2];
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Transmit a byte to DS1085LZ-25
        I2CMasterDataPut(I2C1_BASE, 0x01);
                //Put DIV register address on bus
        I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);
        //Start sending the data in single mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);
        //Receive data from DS1085LZ-25
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_START);                //Start receiving
data in burst mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        DIV[0] = I2CMasterDataGet(I2C1_BASE);
        //Get first byte of DIV (MSByte)
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);
        //Receive data from DS1085LZ-25
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_FINISH);       //Receive the last byte in
burst mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        DIV[1] = I2CMasterDataGet(I2C1_BASE);
        //Get 2nd byte of DIV (LSByte)
        UARTprintf("DIV : %02x%02x\n",DIV[0],DIV[1]);
        //*****************************MUX*****************************
*****
        unsigned char MUX[2];
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false);
        //Transmit a byte to DS1085LZ-25
        I2CMasterDataPut(I2C1_BASE, 0x02);
                //Put MUX register address on bus
        I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);
        //Start sending the data in single mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);
        //Receive data from DS1085LZ-25
```

`

```
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_START);        //Start receiving data in
burst mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        MUX[0] = I2CMasterDataGet(I2C1_BASE);
                //Get first byte of MUX (MSByte)
        I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true);
        //Receive data from DS1085LZ-25
        I2CMasterControl(I2C1_BASE,
I2C_MASTER_CMD_BURST_RECEIVE_FINISH);       //Receive the last byte in
burst mode
        while(I2CMasterBusy(I2C1_BASE));        //Wait till the operation is
completed
        MUX[1] = I2CMasterDataGet(I2C1_BASE);
                //Get 2nd byte of MUX (LSByte)
        UARTprintf("MUX : %02x%02x\n",MUX[0],MUX[1]);
        UARTprintf("\n\n----------------------------------------\n\n");
}
```

B. TALKING FRAMEWORK

This python class can be used to make any device a talking gadget

```
#!/usr/bin/python

from time import sleep
import os
import sys

class TalkingDevice():
   def __init__(self,measure='Temperature'):
      #Initialize the audio player to enable quicker start next time
           self.player = 'aplay -D sysdefault '
      try:
                      pass
#          os.system("amixer cset numid=3 1")
#              os.system("amixer set  75%")
      except:
         print "Oops! Can't access mplayer, attempting to install..."
#          os.system("apt-get install mplayer")

      self.languageList = []
      self.languageIndex = 0
      self.language = -1
      try:
         self.languageList = os.listdir('./lang/support')
         for index in range(len(self.languageList)):
            self.languageList[index] = self.languageList[index][:-4]
```

`

```
            self.language = self.languageList[0]
            self.languageIndex = 0
        except:
            print "No supported languages found!"

        self.unitSupport = {'Temperature':'degree_celsius','Weight':'kilogram'}

        try:
            self.unit = self.unitSupport[measure]
        except:
            print "Cannot support" + measure + " currently."

    def changeLanguage(self):
        self.languageIndex = (self.languageIndex + 1)%len(self.languageList)
        self.language = self.languageList[self.languageIndex]
        command = self.player + ' ./lang/support/'+self.languageList[self.languageIndex]
+ '.wav  '
        os.system(command)

    def speak(self, value):
        print "Received: " + `value`
        if value < 0 :
            command = self.player + ' ./lang/' + self.language + '/minus.wav '
        if ((int(value)+0.5) < (value-0.25)):
            filename = './lang/' + self.language+ '/' + `int(value + 1)` + '.0.wav  '
        elif (int(value)>(value-0.25)):
            filename = './lang/'+self.language+'/'+`int(value)`+'.0.wav  '
        else:
            filename = './lang/'+ self.language + '/'+`int(value)`+'.5.wav  '
        command = self.player + filename
              os.system(command)
              sleep(0.01)
              command =  self.player + ' ./lang/' + self.language + '/' + self.unit +'.wav '
        os.system(command)
        sleep(0.1)

if __name__ == '__main__':
    test = TalkingDevice(sys.argv[1])
    test.speak(1.0)
    sleep(2)
    test.changeLanguage()
    sleep(2)
    test.speak(1.0)
```

`

C. THERMOMETER

The code for interfacing DS18B20 in Talking Thermometer

```
#!/usr/bin/python

import sys
from time import sleep
from PyQt4 import QtCore, QtGui
from ThermometerUI import Ui_Thermometer
from Sensors import DS18B20,PCF8563
from TalkingDevice import TalkingDevice
import csv

class Thermometer(QtGui.QWidget,Ui_Thermometer):
        def __init__(self,parent=None):
                super(Thermometer,self).__init__(parent)
                self.setupUi(self)
                self.saveBtn.setFocus(True)
                self.setupSignals()
                self.show()

        def setupSignals(self):
                self.talker = TalkingDevice("Temperature")
                self.sensor = TempSensor(self)
                self.sensor.start()
                self.csvout = False
                self.timeSensor = PCF8563(1,0x51)

        self.sensor.connect(self.sensor,QtCore.SIGNAL("updateTemp(float)"),self.u
pdateTemp)
                self.langBtn.clicked.connect(self.talker.changeLanguage)
                self.speakBtn.clicked.connect(self.speakUp)
                self.quitBtn.clicked.connect(self.close)
                self.saveBtn.clicked.connect(self.saveCSV)


        def keyPressEvent(self,e):
                if e.key() == QtCore.Qt.Key_Enter:
                        if self.langBtn.hasFocus() == True:
                                print "langBtn"
                                self.talker.changeLanguage()
                        elif self.speakBtn.hasFocus() == True:
                                print "speakBtn"
                                self.speakUp()
                        elif self.quitBtn.hasFocus() == True:
                                print "quitBtn"
                                self.close()
                        elif self.saveBtn.hasFocus() == True:
                                print "saveBtn"
```

`

```
                                        self.saveCSV()

        def saveCSV(self):
                if self.csvout == False:
                        self.csvout = True
                        self.saveBtn.setText('Stop')
                elif self.csvout == True:
                        self.outfile.close()
                        self.csvout = False
                        self.saveBtn.setText('Start')

                if self.csvout == True:
                        try:
                                self.filename = self.timeSensor.read_str() +
".csv"
                                self.outfile = open("Data/"+self.filename,'a')
                                self.writer = csv.writer(self.outfile)
                        except:
                                self.csvout = False
                                self.outfile.close()
                                print "Unable to open CSV File"

        def updateTemp(self,temp):
                self.lcdNumber.display(temp)
                if self.csvout == True:
                        reading = []
                        reading.append(self.timeSensor.read_str())
                        reading.append(temp)
                        self.writer.writerow(reading)

        def speakUp(self):
                self.talker.speak(self.sensor.getTemp())


class TempSensor(QtCore.QThread):
        def __init__(self,parent=None):
                QtCore.QThread.__init__(self,parent)
                self.setTerminationEnabled(True)
                self.temp=0
                self.sensor = DS18B20()

        def run(self):
                while True:
                        try:
                                self.temp = round(self.sensor.read(),2)
                                print "Sense: " + `self.temp`

        self.emit(QtCore.SIGNAL("updateTemp(float)"),self.temp)
                                sleep(0.2)
                        except (KeyboardInterrupt,SystemExit):
```

`

```
                                        GPIO.cleanup()
                                        sys.exit()

        def getTemp(self):
                return self.temp

def main():
        app = QtGui.QApplication(sys.argv)
        ex = Thermometer()
        sys.exit(app.exec_())

if __name__== '__main__':
        main()
```

D. WEIGHING SCALE

Code for interfacing HX711

```
#!/usr/bin/python

import sys
from time import sleep
from PyQt4 import QtCore, QtGui
from WeighingScaleUI import Ui_WeighingScale
from Sensors import PCF8563,HX711
from TalkingDevice import TalkingDevice
import csv

class WeighingScale(QtGui.QWidget,Ui_WeighingScale):
        def __init__(self,parent=None):
                super(WeighingScale,self).__init__(parent)
                self.setupUi(self)
                self.saveBtn.setFocus(True)
                self.setupSignals()
                self.show()

        def setupSignals(self):
                self.talker = TalkingDevice("Weight")
                self.sensor = WeightSensor(self)
                self.sensor.start()
                self.csvout = False
                self.timeSensor = PCF8563(1,0x51)

        self.sensor.connect(self.sensor,QtCore.SIGNAL("updateWt(float)"),self.upd
ateWt)
                self.langBtn.clicked.connect(self.talker.changeLanguage)
                self.speakBtn.clicked.connect(self.speakUp)
                self.quitBtn.clicked.connect(self.close)
                self.saveBtn.clicked.connect(self.saveCSV)
```

`

```
                self.calibBtn.clicked.connect(self.calibrate)

        def keyPressEvent(self,e):
                if e.key() == QtCore.Qt.Key_Enter or e.key() ==
QtCore.Qt.Key_Return:
                        if self.langBtn.hasFocus() == True:
                                print "langBtn"
                                self.talker.changeLanguage()
                        elif self.speakBtn.hasFocus() == True:
                                print "speakBtn"
                                self.speakUp()
                        elif self.quitBtn.hasFocus() == True:
                                print "quitBtn"
                                self.close()
                        elif self.saveBtn.hasFocus() == True:
                                print "saveBtn"
                                self.saveCSV()
                        elif self.calibBtn.hasFocus() == True:
                                self.calibrate()

        def saveCSV(self):
                if self.csvout == False:
                        self.csvout = True
                        self.saveBtn.setText('Stop')
                elif self.csvout == True:
                        self.outfile.close()
                        self.csvout = False
                        self.saveBtn.setText('Save')

                if self.csvout == True:
                        try:
                                self.filename = "Weight - " +
self.timeSensor.read_str() + ".csv"
                                self.outfile = open("Data/"+self.filename,'a')
                                self.writer = csv.writer(self.outfile)
                        except:
                                self.csvout = False
                                self.outfile.close()
                                print "Unable to open CSV File"

        def updateWt(self,weight):
                self.lcdNumber.display(weight)
                if self.csvout == True:
                        reading = []
                        reading.append(self.timeSensor.read_str())
                        reading.append(round(weight,3))
                        self.writer.writerow(reading)

        def speakUp(self):
                self.talker.speak(self.sensor.getWeight())
```

`

```
        def calibrate(self):
                respoffset = QtGui.QMessageBox.information(self,'Offset
Calculation',"Remove everything from
platform.",QtGui.QMessageBox.Ok|QtGui.QMessageBox.Cancel,QtGui.QMessageB
ox.Ok)
                if respoffset==QtGui.QMessageBox.Ok:
                        self.sensor.calib(0)
                else:
                        pass
                respscale = QtGui.QMessageBox.information(self,'Scale
Calculation',"Place a weight of 1 kg on the
platform",QtGui.QMessageBox.Ok|QtGui.QMessageBox.Cancel,QtGui.QMessageBo
x.Ok)
         if respscale==QtGui.QMessageBox.Ok:
              self.sensor.calib(1)
         else:
              pass
                resp = QtGui.QMessageBox.information(self,'Calibration
Done',"Calibration done. Offset: " + `self.sensor.offset` + " Scale:
"+`self.sensor.scale`)


class WeightSensor(QtCore.QThread):
        def __init__(self,parent=None):
                QtCore.QThread.__init__(self,parent)
                self.setTerminationEnabled(True)
                self.weight=0
                self.sensor = HX711(25,27)
                self.offset =0
                self.scale = 1.172
                self.sensor.set_reading_format("LSB","MSB")
                self.sensor.set_reference_unit(92)
                self.sensor.reset()
                self.sensor.tare()

        def run(self):
                while True:
                        try:
                                self.weight =
abs(round((self.sensor.get_weight(10) - self.offset)/self.scale,3))
                                print "Sense: " + `self.weight`

        self.emit(QtCore.SIGNAL("updateWt(float)"),self.weight)
                                sleep(0.2)
                        except (KeyboardInterrupt,SystemExit):
                                GPIO.cleanup()
                                sys.exit()

        def getWeight(self):
```

`

```
                return self.weight

        def calib(self,option):
                try:
                        if option == 0:
                                values = [0,0,0,0,0,0,0,0,0,0]
                                sm = 0
                                for i in range(10):
                                        values[i] = self.sensor.get_weight(10)
                                        print values[i]
                                        self.sensor.power_down()
                                        self.sensor.power_up()
                                        sm  = sm + values[i]
                                        sleep(0.1)
                                self.offset = abs(round(sm/10,2))
                                print "Offset done " + `self.offset`
                        elif option == 1:
                                values = [0,0,0,0,0,0,0,0,0,0]
                                sm = 0.0
                                for i in range(10):
                                        values[i] = self.sensor.get_weight(10)
                                        print values[i]
                                        self.sensor.power_down()
                                        self.sensor.power_up()
                                        sm = sm + values[i]
                                        sleep(0.1)
                                self.scale =abs(sm/10)
                                print "Scale done " + `self.scale`
                except KeyboardInterrupt:
                        pass

def main():
        app = QtGui.QApplication(sys.argv)
        ex = WeighingScale()
        sys.exit(app.exec_())

if __name__== '__main__':
        main()
```

`

E. PHOTOMETER

```
import sys
import csv
from PyQt4 import QtGui,QtCore
from PhotometerUI import Ui_Photometer
from time import sleep
#from Sensors import PCF8563, HX711

# currentState = 0 :  Starting Condition (Radio Buttons)
# currentState = 1 :  Time Selection for laterStart
# currentState = 2 :  Ending Condition (Radio Buttons)
# currentState = 3 :  Time Selection for timeEnd
# currentState = 4 :  Sample Selection for sampleEnd
# currentState = 5 :  Sample Rate Selection
# currentState = 6 :  Start of Sampling
# currentState = 7 :  End Sampling

# startSetting = 0 : Start Immediately
# startSetting = 1 : Start at a particular time
# startSetting = 2 : Start when the button is pressed

# endSetting = 0 : Stop when button is pressed
# endSetting = 1 : Stop at a particular time
# endSetting = 2 : Stop after taking N number of samples

operationReady = 0

class Photometer(QtGui.QMainWindow,Ui_Photometer):
    global operationReady
    def __init__(self,parent=None):
        super(Photometer,self).__init__(parent)
        self.setupUi(self)
        self.nowStartSelect.setEnabled(True)
        self.laterStartSelect.setEnabled(True)
        self.buttonStartSelect.setEnabled(True)
        self.nowStartSelect.setChecked(True)
        self.nowStartSelect.setFocus(True)
        self.setupSignals()
        self.show()

    def setupSignals(self):
        self.rtcInt = 13
        self.hxDat = 25
        self.hxSck = 27
        self.led = 18
        self.button = 23
        self.buzzer = 24
        self.power = 17
        self.sensor = None
```

`

```
        self.currentState = 0
        self.startSetting = 0
        self.endSetting = 0
        self.startTime = None
        self.endTime = None
        self.sampleRate = 0
        self.endSamples = 0
        self.value = 0
        self.RTC = PCF8563(1,0x51)
        self.ADC = HX711(self.hxDat,self.hxSck)
        self.ADC.set_reading_format("LSB","MSB")
        self.ADC.set_reference_unit(92)
        self.ADC.reset()
        self.ADC.tare()
        GPIO.setup(self.button,GPIO.IN)
        GPIO.setup(self.power,GPIO.OUT)
        GPIO.output(self.power,GPIO.LOW)
        GPIO.setup(self.buzzer,GPIO.OUT)
        GPIO.output(self.buzzer,GPIO.LOW)
       GPIO.setup(self.led,GPIO.OUT)
       GPIO.output(self.led,GPIO.LOW)
        self.goButton.clicked.connect(self.goButtonClicked)

    def keyPressEvent(self,e):
        if e.key() == QtCore.Qt.Key_Enter or e.key() == QtCore.Qt.Key_Return:
            if self.currentState == 0:
                if self.nowStartSelect.isChecked() == True:
                    self.currentState = 2
                    self.startSetting = 0
                    self.nowStartSelect.setEnabled(False)
                    self.laterStartSelect.setEnabled(False)
                    self.buttonStartSelect.setEnabled(False)
                    self.startTimeEdit.setEnabled(False)
                    self.timeEndSelect.setEnabled(True)
                    self.buttonEndSelect.setEnabled(True)
                    self.sampleEndSelect.setEnabled(True)
                    self.buttonEndSelect.setFocus(True)
                elif self.laterStartSelect.isChecked() == True:
                    self.currentState = 1
                    self.startSetting = 1
                    self.nowStartSelect.setEnabled(False)
                    self.laterStartSelect.setEnabled(False)
                    self.buttonStartSelect.setEnabled(False)
                    self.startTimeEdit.setDate(QtCore.QDate.currentDate())
                    self.startTimeEdit.setEnabled(True)
                    self.startTimeEdit.setFocus()
                elif self.buttonStartSelect.isChecked() == True:
                    self.currentState = 2
                    self.startSetting = 2
                    self.nowStartSelect.setEnabled(False)
```

`

```
            self.laterStartSelect.setEnabled(False)
            self.buttonStartSelect.setEnabled(False)
            self.startTimeEdit.setEnabled(False)
            self.timeEndSelect.setEnabled(True)
            self.buttonEndSelect.setEnabled(True)
            self.sampleEndSelect.setEnabled(True)
            self.buttonEndSelect.setFocus(True)

        elif self.currentState == 1:
            self.currentState = 2
            self.startTime = self.startTimeEdit.dateTime()
            self.nowStartSelect.setEnabled(False)
            self.laterStartSelect.setEnabled(False)
            self.buttonStartSelect.setEnabled(False)
            self.startTimeEdit.setEnabled(False)
            self.timeEndSelect.setEnabled(True)
            self.buttonEndSelect.setEnabled(True)
            self.sampleEndSelect.setEnabled(True)
            self.buttonEndSelect.setFocus(True)

        elif self.currentState == 2:
            if self.buttonEndSelect.isChecked() == True:
                self.currentState = 5
                self.endSetting = 0
                self.buttonEndSelect.setEnabled(False)
                self.timeEndSelect.setEnabled(False)
                self.sampleEndSelect.setEnabled(False)
                self.endTimeEdit.setEnabled(False)
                self.sampleSelect.setEnabled(False)
                self.rateSelect.setEnabled(True)
                self.rateSelect.setFocus(True)
            elif self.timeEndSelect.isChecked()==True:
                self.currentState = 3
                self.endSetting = 1
                self.endTimeEdit.setDate(QtCore.QDate.currentDate())
                self.endTimeEdit.setEnabled(True)
                self.endTimeEdit.setFocus()
            elif self.sampleEndSelect.isChecked()==True:
                self.currentState = 4
                self.endSetting = 2
                self.endTimeEdit.setEnabled(False)
                self.sampleSelect.setEnabled(True)
                self.sampleSelect.setFocus()

        elif self.currentState == 3:
            self.currentState = 5
            self.endTime = self.endTimeEdit.dateTime()
            self.buttonEndSelect.setEnabled(False)
            self.timeEndSelect.setEnabled(False)
            self.sampleEndSelect.setEnabled(False)
```

`

```
            self.endTimeEdit.setEnabled(False)
            self.sampleSelect.setEnabled(False)
            self.rateSelect.setEnabled(True)
            self.rateSelect.setFocus(True)

        elif self.currentState == 4:
            self.currentState = 5
            self.endSamples = self.sampleSelect.value()
            self.timeEndSelect.setEnabled(False)
            self.sampleEndSelect.setEnabled(False)
            self.endTimeEdit.setEnabled(False)
            self.sampleSelect.setEnabled(False)
            self.rateSelect.setEnabled(True)
            self.rateSelect.setFocus(True)

        elif self.currentState == 5:
            self.currentState = 6
            self.sampleRate = self.rateSelect.value()
            self.rateSelect.setEnabled(False)
            self.goButton.setEnabled(True)
            self.goButton.setFocus(True)

        elif self.currentState == 6:
            self.sampleCount = 0
            self.nowStartSelect.setEnabled(False)
            self.laterStartSelect.setEnabled(False)
            self.buttonStartSelect.setEnabled(False)
            self.buttonEndSelect.setEnabled(False)
            self.timeEndSelect.setEnabled(False)
            self.sampleEndSelect.setEnabled(False)
            self.rateSelect.setEnabled(False)
            self.goButtonClicked()

        elif self.currentState == 7:
            if self.endSetting == 0:
                print "Stop Sampling using Button"
                operationReady = 0

    elif e.key() == QtCore.Qt.Key_Left:
        if self.currentState == 0 :
            if self.laterStartSelect.isChecked() == True:
                                                self.nowStartSelect.setChecked(True)
            elif self.buttonStartSelect.isChecked() == True:
                                                self.laterStartSelect.setChecked(True)

        elif self.currentState == 2:
            if self.timeEndSelect.isChecked() == True:
                self.buttonEndSelect.setChecked(True)
            elif self.sampleEndSelect.isChecked() == True:
                self.timeEndSelect.setChecked(True)
```

`

```
        elif self.currentState == 1:
            if self.startTimeEdit.currentSection() ==
QtGui.QDateTimeEdit.MinuteSection:

self.startTimeEdit.setCurrentSection(QtGui.QDateTimeEdit.HourSection)
            elif self.startTimeEdit.currentSection() ==
QtGui.QDateTimeEdit.HourSection:

self.startTimeEdit.setCurrentSection(QtGui.QDateTimeEdit.MinuteSection)

        elif self.currentState == 3:
            if self.endTimeEdit.currentSection() ==
QtGui.QDateTimeEdit.MinuteSection:
                self.endTimeEdit.setCurrentSection(QtGui.QDateTimeEdit.HourSection)
            elif self.endTimeEdit.currentSection() ==
QtGui.QDateTimeEdit.HourSection:

self.endTimeEdit.setCurrentSection(QtGui.QDateTimeEdit.MinuteSection)

    elif e.key() == QtCore.Qt.Key_Right:
        if self.currentState == 0:
            if self.nowStartSelect.isChecked() == True:
                self.laterStartSelect.setChecked(True)
            elif self.laterStartSelect.isChecked() == True:
                self.buttonStartSelect.setChecked(True)

        elif self.currentState == 2:
            if self.buttonEndSelect.isChecked() == True:
                self.timeEndSelect.setChecked(True)
            elif self.timeEndSelect.isChecked() == True:
                self.sampleEndSelect.setChecked(True)

        elif self.currentState == 1:
            if self.startTimeEdit.currentSection() ==
QtGui.QDateTimeEdit.HourSection:

self.startTimeEdit.setCurrentSection(QtGui.QDateTimeEdit.MinuteSection)
            elif self.startTimeEdit.currentSection() ==
QtGui.QDateTimeEdit.MinuteSection:

self.startTimeEdit.setCurrentSection(QtGui.QDateTimeEdit.HourSection)

        elif self.currentState == 3:
            if self.endTimeEdit.currentSection() ==
QtGui.QDateTimeEdit.HourSection:

self.endTimeEdit.setCurrentSection(QtGui.QDateTimeEdit.MinuteSection)
            elif self.endTimeEdit.currentSection() ==
QtGui.QDateTimeEdit.MinuteSection:
```

`

```
                self.endTimeEdit.setCurrentSection(QtGui.QDateTimeEdit.HourSection)
        elif e.key() == QtCore.Qt.Key_Up:
            if self.startTimeEdit.isEnabled()==True:
                time = self.startTimeEdit.time()
                if self.startTimeEdit.currentSection() == 0x0010:
                    if time.hour()+1 == 24:
                                                    time.setHMS(0,time.minute(),0)
                    else:
                                                time.setHMS(time.hour() +
1,time.minute(),0)
                    self.startTimeEdit.setTime(time)
                elif self.startTimeEdit.currentSection() == 0x0008:
                                                    if time.minute()+1 == 60:

time.setHMS(time.hour(),0,0)
                                                    else:

                time.setHMS(time.hour(),time.minute()+1,0)
                                                    self.startTimeEdit.setTime(time)

            elif self.endTimeEdit.isEnabled()==True:
                                            time = self.endTimeEdit.time()
                                            if self.endTimeEdit.currentSection() ==
0x0010:
                                                    if time.hour()+1 == 24:

time.setHMS(0,time.minute(),0)
                                                    else:
                                                            time.setHMS(time.hour() +
1,time.minute(),0)
                                                    self.endTimeEdit.setTime(time)
                                            elif self.endTimeEdit.currentSection() ==
0x0008:
                                                    if time.minute()+1 == 60:

time.setHMS(time.hour(),0,0)
                                                    else:

                time.setHMS(time.hour(),time.minute()+1,0)
                                                    self.endTimeEdit.setTime(time)

            elif self.sampleSelect.isEnabled() == True:
                                            v = self.sampleSelect.value()
                                            v = v+10
                                            if v == 1010:
                                                    v = 10
                                            self.sampleSelect.setValue(v)

            elif self.rateSelect.hasFocus() == True:
                                            v = self.rateSelect.value()
```

`

```
                                        v = v+1
                                        if v == 11:
                                                v = 1
                                        self.rateSelect.setValue(v)

        elif e.key() == QtCore.Qt.Key_Down:
                                if self.startTimeEdit.isEnabled()==True:
                                        time = self.startTimeEdit.time()
                                        if self.startTimeEdit.currentSection() ==
0x0010:
                                                if time.hour()-1 == -1:

time.setHMS(23,time.minute(),0)
                                                else:
                                                        time.setHMS(time.hour() -
1,time.minute(),0)
                                                self.startTimeEdit.setTime(time)
                                        elif self.startTimeEdit.currentSection() ==
0x0008:
                                                if time.minute()-1 == -1:

time.setHMS(time.hour(),59,0)
                                                else:

                time.setHMS(time.hour(),time.minute()-1,0)
                                                self.startTimeEdit.setTime(time)

                                elif self.endTimeEdit.isEnabled()==True:
                                        time = self.endTimeEdit.time()
                                        if self.endTimeEdit.currentSection() ==
0x0010:
                                                if time.hour()-1 == -1:

time.setHMS(23,time.minute(),0)
                                                else:
                                                        time.setHMS(time.hour() -
1,time.minute(),0)
                                                self.endTimeEdit.setTime(time)
                                        elif self.endTimeEdit.currentSection() ==
0x0008:
                                                if time.minute()-1 == -1:

time.setHMS(time.hour(),59,0)
                                                else:

                time.setHMS(time.hour(),time.minute()-1,0)
                                                self.endTimeEdit.setTime(time)

                                elif self.sampleSelect.isEnabled() == True:
                                        v = self.sampleSelect.value()
```

`

```
                                        v = v-10
                                        if v == 0:
                                                v = 1000
                                        self.sampleSelect.setValue(v)

                                elif self.rateSelect.hasFocus() == True:
                                        v = self.rateSelect.value()
                                        v = v-1
                                        if v == 0:
                                                v = 10
                                        self.rateSelect.setValue(v)


    def goButtonClicked(self):
        global operationReady
        if self.currentState == 6:
            if operationReady == 0:
                self.goButton.setEnabled(False)
                if self.startSetting == 0:
                    self.currentState = 7
                    operationReady = 1
                    print  "Starting Sampling"
                    self.startSampling()
                elif self.startSetting == 1:
                    #self.setAlarm(self.startTime)
                    self.waitAlarm()
                    self.currentState = 7
                    operationReady = 1
                    print  "Starting Sampling after RTC"
                    self.startSampling()
                elif self.startSetting == 2:
                    operationReady = 1
            elif operationReady == 1:
                self.currentState = 7
                print  "Starting Sampling after Wait"
                self.startSampling()

    def waitAlarm(self):
        GPIO.setup(self.rtcInt,GPIO.IN)
        while GPIO.input(self.rtcInt) == True:
            pass
        return

    def setAlarm(self,t):
        hrs = int(t.toString("h"))
        mins = int(t.toString("m"))
                        self.RTC.disable_alarm_interrupt()
                        self.RTC.clear_alarm()
                        self.RTC.enable_alarm_interrupt()
                        self.RTC.set_daily_alarm(hrs,mins)
```

`

```
    return

def startSampling(self):
    global operationReady
    filename = self.RTC.read_str() + ".csv"
    delaytime = (1.0/self.sampleRate) - 0.08
    outfile = open("./Data/" + filename,'a')
    writer = csv.writer(outfile)
    print "File Open : "+ filename
    data = []
    num_samples = 0
    if self.endSetting == 1:
      self.setAlarm(self.endTime)
      self.alarmThread = rtcAlarm(self.rtcInt)
      self.alarmThread.start()
    elif self.endSetting == 0:
                                self.buttonThread = buttonEndThread(self.button)
                                self.buttonThread.start()

    GPIO.output(self.power,GPIO.HIGH)
                        GPIO.output(self.buzzer,GPIO.LOW)
                        GPIO.output(self.led,GPIO.LOW)

    while operationReady == 1:
       data.append(self.RTC.read_str())                    #Obtain Timestamp
       reading = ((self.ADC.get_weight(8)>>6))
       data.append(reading)
       print data
       writer.writerow(data)
       num_samples = num_samples + 1
       if self.endSetting == 2:
          if num_samples == self.endSamples:
             operationReady = 0
       data = []
                                sleep(delaytime)
    outfile.close()
    GPIO.output(self.power,GPIO.LOW)
                        GPIO.output(self.led,GPIO.HIGH)
                        GPIO.output(self.buzzer,GPIO.HIGH)
                        sleep(0.5)
                        GPIO.output(self.buzzer,GPIO.LOW)
                        sleep(0.5)
                        GPIO.output(self.buzzer,GPIO.HIGH)
                        sleep(0.5)
                        GPIO.output(self.buzzer,GPIO.LOW)
                        sleep(0.5)
                        GPIO.output(self.buzzer,GPIO.HIGH)
                        sleep(0.5)
                        GPIO.output(self.buzzer,GPIO.LOW)
                        sleep(0.5)
```

`

```
                    if self.endSetting == 1:
                            self.alarmThread.terminate()
                            self.RTC.clear_alarm()
                            self.RTC.disable_alarm_interrupt()
                    elif self.endSetting == 0:
                            self.buttonThread.terminate()

        self.nowStartSelect.setEnabled(True)
                    self.laterStartSelect.setEnabled(True)
        self.buttonStartSelect.setEnabled(True)
                    self.nowStartSelect.setChecked(True)
                    self.nowStartSelect.setFocus(True)
        self.currentState = 0
        operationReady = 0

class buttonEndThread(QtCore.QThread):
        def __init__(self,pin,parent=None):
                    QtCore.QThread.__init__(self,parent)
                    self.setTerminationEnabled(True)
                    self.pin = pin
                    GPIO.setup(self.pin,GPIO.IN)
        def run(self):
                    global operationReady
                    sleep(2)
                    while True:
                            try:
                                    if GPIO.input(self.pin)==GPIO.LOW:
                                            sleep(0.08)
                                            if GPIO.input(self.pin)==GPIO.LOW:
                                                    print "buttonEnd"
                                                    operationReady = 0
                                    sleep(0.1)
                            except KeyboardInterrupt:
                                    print "Exiting"

class rtcAlarm(QtCore.QThread):
        def __init__(self,pin,parent=None):
                    QtCore.QThread.__init__(self,parent)
                    self.setTerminationEnabled(True)
                    self.pin = pin
                    GPIO.setup(self.pin,GPIO.IN,pull_up_down=GPIO.PUD_UP)
        def run(self):
                    global operationReady
                    while True:
                            try:
                                    if GPIO.input(self.pin)==GPIO.LOW:
                                            sleep(0.08)
                                            if GPIO.input(self.pin)==GPIO.LOW:
                                                    print "alarm"
                                                    operationReady = 0
```

`

```
                    sleep(0.1)
                except KeyboardInterrupt:
                    print "Exiting"


def main():
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        global operationReady
        operationReady = 0
    try:
        app = QtGui.QApplication(sys.argv)
        ex = Photometer()
        app.exec_()
    finally:
        GPIO.cleanup()

if __name__=='__main__':
    main()
```

F. MILLIOHMMETER

```
#!/usr/bin/python

import sys
from time import sleep
from PyQt4 import QtCore, QtGui
from MilliohmmeterUI import Ui_Milliohmmeter
#from Sensors import PCF8563,HX711
import csv

boldFont = None
normalFont = None
class Milliohmmeter(QtGui.QWidget,Ui_Milliohmmeter):
    global boldFont
    global normalFont
    def __init__(self,parent=None):
            super(Milliohmmeter,self).__init__(parent)
            self.setupUi(self)
            self.saveBtn.setFocus(True)
            self.setupSignals()
            self.show()

    def setupSignals(self):
            self.sensor = ResistanceSensor(self)
        self.sensor.start()
        self.csvout = False
```

`

```
                self.timeSensor = PCF8563(1,0x51)

self.sensor.connect(self.sensor,QtCore.SIGNAL("updateRes(float,int)"),self.updateRes)
        self.quitBtn.clicked.connect(self.close)
        self.saveBtn.clicked.connect(self.saveCSV)

            def keyPressEvent(self,e):
                if e.key() == QtCore.Qt.Key_Enter or e.key() == QtCore.Qt.Key_Return:
                    if self.quitBtn.hasFocus() == True:
                        print "quitBtn"
                        self.close()
                    elif self.saveBtn.hasFocus() == True:
                        print "saveBtn"
                        self.saveCSV()

    def saveCSV(self):
                if self.csvout == False:
                    self.csvout = True
                    self.saveBtn.setText('Stop')
                elif self.csvout == True:
                    self.outfile.close()
                    self.csvout = False
                    self.saveBtn.setText('Save')

                if self.csvout == True:
                    try:
                        self.filename = "Milliohm - " + self.timeSensor.read_str() + ".csv"
                        self.outfile = open("Data/"+self.filename,'a')
                        self.writer = csv.writer(self.outfile)
                    except:
                        self.csvout = False
                        self.outfile.close()
                        print "Unable to open CSV File"

    def updateRes(self,resistance,rangeSelect):
        if rangeSelect == 0:
            self.lblLow.setFont(boldFont)
            self.lblMed.setFont(normalFont)
            self.lblHigh.setFont(normalFont)
        elif rangeSelect == 1:
            self.lblLow.setFont(normalFont)
            self.lblMed.setFont(boldFont)
            self.lblHigh.setFont(normalFont)
        elif rangeSelect == 2:
            self.lblLow.setFont(normalFont)
            self.lblMed.setFont(normalFont)
            self.lblHigh.setFont(boldFont)
```

`

```
                        self.lcdNumber.display(resistance)
                        if self.csvout == True:
                                reading = []
                                reading.append(self.timeSensor.read_str())
                                reading.append(round(resistance,3))
                                self.writer.writerow(reading)

class ResistanceSensor(QtCore.QThread):
        def __init__(self,parent=None):
                        QtCore.QThread.__init__(self,parent)
                        self.setTerminationEnabled(True)
                        self.resistance=0
                        self.sensor = HX711(25,27)
                        self.offset =0
                        self.scale = 1.172
                        self.sensor.set_reading_format("LSB","MSB")
                        self.sensor.set_reference_unit(92)
                        self.sensor.reset()
                        self.sensor.tare()
      self.rangeMin = 0
      self.rangeMax = 5101
      self.rangeBand = 10
      self.rangeSet = 2
      self.lowPin = 12
      self.medPin = 17
      self.highPin = 24
      GPIO.setup(self.lowPin,GPIO.IN)
      GPIO.setup(self.medPin,GPIO.IN)
      GPIO.setup(self.highPin,GPIO.IN)
      GPIO.output(self.medPin,GPIO.HIGH)
      GPIO.output(self.highPin,GPIO.HIGH)
      GPIO.output(self.lowPin,GPIO.HIGH)

        def run(self):
                        while True:
                                try:
            self.rangeSet = self.selectRange()
            if self.rangeSet == -1:
               self.resistance = -1
               self.emit(QtCore.SIGNAL("updateRes(float,int)"),self.resistance,0)
            elif self.rangeSet == 3:
               self.resistance = -1
               self.emit(QtCore.SIGNAL("updateRes(float,int)"),self.resistance,3)
            else:
               self.resistance = abs(round((self.getReading() - self.offset)/self.scale,3))
               print "Sense: " + `self.resistance`

self.emit(QtCore.SIGNAL("updateRes(float,int)"),self.resistance,self.rangeSet)
                                        sleep(0.2)
```

`

```
                except (KeyboardInterrupt,SystemExit):
                    GPIO.cleanup()
                    sys.exit()

    def getReading(self):
        return self.sensor.get_weight(5)

    def selectRange(self):
        GPIO.output(self.lowPin,GPIO.HIGH)
        GPIO.output(self.medPin,GPIO.HIGH)
        GPIO.output(self.highPin,GPIO.LOW)
        data = self.getReading()
        if data > self.rangeMax:
            return 3
        if data in range(self.rangeMin):    #Too low for HighRange
            GPIO.output(self.highPin,GPIO.HIGH)
            GPIO.output(self.medPin,GPIO.LOW)
            data = self.getReading()
            if data in range(self.rangeMin):    #Too low for MedRange
                GPIO.output(self.medPin,GPIO.HIGH)
                GPIO.output(self.lowPin,GPIO.LOW)
                data = self.getReading()
                if data in range(self.rangeMin):
                    return -1
                else:
                    return 0
            else:
                return 1
        else:
            return 2

def main():
    global boldFont
    global normalFont
    boldFont = QtGui.QFont()
    boldFont.setBold(True)
    boldFont.setUnderline(True)
    normalFont = QtGui.QFont()
    normalFont.setBold(False)
    normalFont.setUnderline(False)
    try:
        app = QtGui.QApplication(sys.argv)
        ex = Milliohmmeter()
        sys.exit(app.exec_())
    finally:
        GPIO.cleanup()

if __name__== '__main__':
    GPIO.setmode(GPIO.BCM)
            main()
```