

# **AUTOMATED BIRD SONG RECORDER**

*A Project Report*

*submitted by*

**M SAILENDRA**

*in partial fulfilment of the requirements  
for the award of the degree of*

**MASTER OF TECHNOLOGY**



**DEPARTMENT OF ELECTRICAL ENGINEERING  
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.**

**May 2015**

# THESIS CERTIFICATE

This is to certify that the thesis titled **AUTOMATED BIRD SONG RECORDER**, submitted by **M Sailendra**, to the Indian Institute of Technology, Madras, for the award of the degree of **MASTER OF TECHNOLOGY**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

**Dr.Anil Prabhakar**  
Research Guide  
Professor  
Dept. of Electrical Engineering  
IIT-Madras, 600036

Place: Chennai

Date: 6th May 2015

## **ACKNOWLEDGEMENTS**

I take this opportunity to express my deepest gratitude to my project guide Dr. Anil Prabhakar for his valuable guidance and motivation when needed throughout the project. I am very grateful to him for providing his valuable time to guide me during the project.

It is a privilege to be a student in IIT Madras. I express special thanks to all my teachers for all the academic insight obtained from them. I also acknowledge the excellent facilities provided by the institute to the students.

Special Thanks to Yangala Laxmipathi and all my lab members for giving valuable suggestions through out the project.

I also thank my family for their unquestioned support and trust in me.

# **ABSTRACT**

Bird songs play a vital role in the communication between individuals and species. A Bird may listen to other birds and classify them as con-specific, neighbour or stranger, mate or non-mate, kin or non-kin. It may also sing to other birds for mate attraction, or territory defence.

Generally Bird song recordings are huge and these recordings contain a lot of inactive period and also background noise. The purpose of this project is to build a working prototype of an Automated field recorder which will record bird songs and eliminate the unnecessary signal using simple threshold Algorithm and the filtered recordings are sent through Cellular Packet Data Service to remote server.

The entire hardware is powered by lithium-polymer batteries and are charged daily through solar panel.

**KEYWORDS:** Raspberry Pi; Audio Recorder; Arduino; Solar.

# TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS</b>	<b>i</b>
<b>ABSTRACT</b>	<b>ii</b>
<b>LIST OF TABLES</b>	<b>v</b>
<b>LIST OF FIGURES</b>	<b>1</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Motivation . . . . .	2
1.2 Organisation . . . . .	2
1.2.1 Introduction . . . . .	2
1.2.2 Literature Review . . . . .	2
1.2.3 Implementation . . . . .	3
1.2.4 Working . . . . .	3
1.2.5 Results and Conclusion . . . . .	3
<b>2 Literature Review</b>	<b>4</b>
2.1 Raspberry Pi . . . . .	4
2.2 Microphones and Preamplifier's . . . . .	5
2.3 Technology . . . . .	6
2.3.1 Python . . . . .	6
2.3.2 Linux Shell Script . . . . .	6
2.3.3 Cron . . . . .	6
2.4 Arduino . . . . .	7
2.5 Proposed Hardware . . . . .	8
<b>3 IMPLEMENTATION</b>	<b>9</b>
3.1 Setting the Raspberrypi . . . . .	9
3.1.1 Installing Raspban . . . . .	9

3.1.2	Installing XRDP . . . . .	9
3.2	Setting USB Modem . . . . .	10
3.2.1	Installing Packages . . . . .	10
3.2.2	Testing Connection . . . . .	11
3.2.3	Automate Connection . . . . .	11
3.3	Connecting to Dropbox . . . . .	14
3.4	Arduino and its peripherals . . . . .	16
3.4.1	RTC . . . . .	16
3.4.2	16x2 LCD . . . . .	16
3.4.3	Lipo Fuel Guage . . . . .	16
3.5	Communication between Arduino and Raspberry Pi . . . . .	18
3.6	Solar charger . . . . .	19
3.7	Block Diagram . . . . .	19
3.8	Schematics . . . . .	19
<b>4</b>	<b>Working</b>	<b>22</b>
4.1	Flow Chart . . . . .	22
4.2	Process Flow . . . . .	23
<b>5</b>	<b>Results and Conclusion</b>	<b>24</b>
5.1	Overall Energy Consumption . . . . .	24
5.2	Battery Charging . . . . .	25
5.3	Threshold Output . . . . .	25
5.4	Assembled Device . . . . .	26
5.5	Future Scope . . . . .	26
<b>A</b>	<b>Main Code</b>	<b>27</b>
<b>B</b>	<b>Threshold Code</b>	<b>29</b>
<b>C</b>	<b>Arduino Code</b>	<b>32</b>
<b>D</b>	<b>Dropbox Script</b>	<b>36</b>

## LIST OF TABLES

3.1	List of Parts and Values . . . . .	21
5.1	Energy consumption. . . . .	24
5.2	Device On Time. . . . .	24

## LIST OF FIGURES

2.1	Raspberry Pi Model B. . . . .	4
2.2	Wolfson Audio Card Rev 1. . . . .	5
2.3	A screenshot of the Arduino IDE showing the Blink program. . . . .	7
3.1	A screenshot of the Raspian OS. . . . .	10
3.2	Sakis3g Interactive Screen. . . . .	12
3.3	Successful Connection Message. . . . .	12
3.4	Connection Information. . . . .	13
3.5	Connection Information. . . . .	13
3.6	Sakis3g Configuration File . . . . .	14
3.7	New Dropbox App . . . . .	15
3.8	Generate Access Token . . . . .	15
3.9	DS3231 Module . . . . .	16
3.10	Arduino - Tiny RTC Connection Diagram . . . . .	17
3.11	Arduino Lcd Connection Diagram . . . . .	17
3.12	Lipo Fuel Guage Front, Back . . . . .	18
3.13	Solar charger . . . . .	19
3.14	Block Diagram Of The Recorder . . . . .	20
3.15	Schematics Of The Interface Board . . . . .	20
4.1	Process Chart of the Recorder . . . . .	22
5.1	Battery Percentage vs Time. . . . .	25
5.2	Bird Song: Original, After threshold Generated, Deleted . . . . .	26
5.3	Assembled Recorder . . . . .	26



# CHAPTER 1

## Introduction

The introduction gives the outline of scope and context of the project.

**Project Title:** Autonomous Bird Song Recorder The purpose of this project is to build a working prototype of an Automated field recorder which will record bird songs and eliminate the unnecessary signal using simple threshold Algorithm and the filtered recordings are sent through Cellular Packet Data Service to remote server. The entire hardware is powered by Lithium-Polymer batteries and are charged daily through solar panel.

### 1.1 Motivation

Generally Bird song recordings are huge and these recordings contain a lot of inactive period and also background noise. and for recording one has to reach to the places and record daily. On completion of this project these two problems will be addressed.

### 1.2 Organisation

This project is been divided into five chapters mainly as follows:

#### 1.2.1 Introduction

In this section it gives a general idea of the project and also about the Aims and objectives of this project.

#### 1.2.2 Literature Review

This section contains information of the raspberry pi and arduino and proposed hardware

### **1.2.3 Implementation**

In this the hardware and the software tools required for the carrying out of the project is been specified also the interface designs for the project are also discussed.

### **1.2.4 Working**

In this a brief description of how the actual device works and its process flow.

### **1.2.5 Results and Conclusion**

This chapter contains the results, conclusion and the future work of the project.

# CHAPTER 2

## Literature Review

### 2.1 Raspberry Pi

The Raspberry Pi is a single-board computer the size of a credit-card-sized developed in the UK by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools. The Raspberry Pi is manufactured through licensed manufacturing deals with Element 14/Premier Farnell and RS Components.

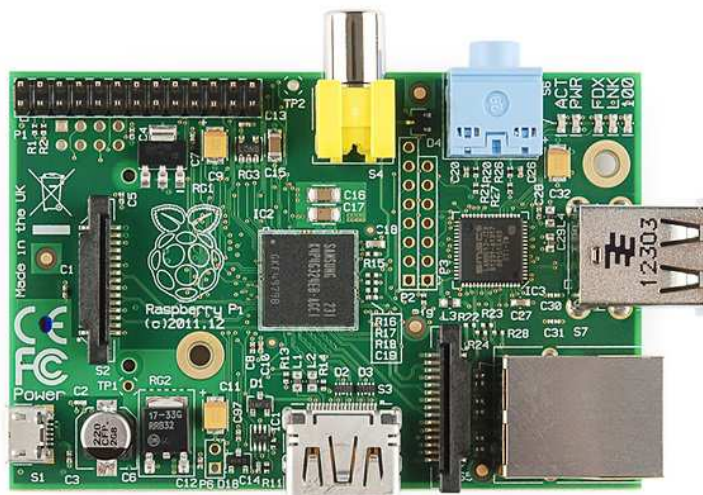


Figure 2.1: Raspberry Pi Model B.

Both of these companies sell the Raspberry Pi online. The Raspberry Pi has a Broadcom BCM2835 system on a chip (SoC), which includes an ARM1176JZF-S 700 MHz processor (The firmware includes a number of "Turbo" modes so that the user can attempt overclocking, up-to 1 GHz, without affecting the warranty), VideoCore IV GPU, and originally shipped with 256 megabytes of RAM, later upgraded to 512MB. It does not include a built-in hard disk or solid-state drive, but uses an SD card for booting and long-term storage.

The pi can be installed with Debian and Arch Linux ARM which can be downloaded from the raspberry pi website.

## 2.2 Microphones and Preamplifier's

Since we need to Record BirdSongs and most birds fall under 10khz category. And our recorder needs to record from all the directions so we should choose an omnidirectional Microphone

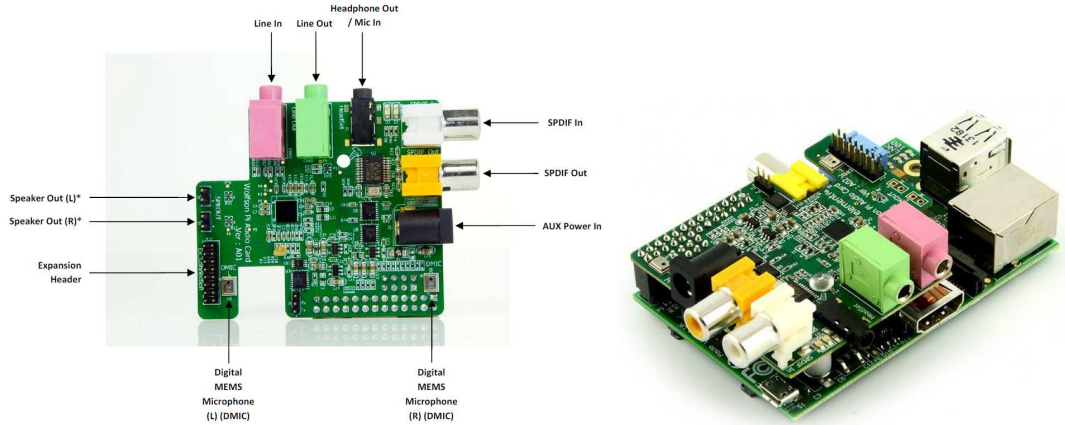


Figure 2.2: Wolfson Audio Card Rev 1.

And for preamplifier we used a wolfson audio card which has two DMIC's (Digital Mems Microphone) on the board. The Wolfson Audio Card is based on Wolfson WM5102 audio hub codes. The WM5102 is a highly-integrated low-power audio system for smartphones, tablets and other portable audio devices. It combines wideband telephony voice processing with a flexible, high-performance audio hub codec. The WM5102 digital core provides a powerful combination of fixed-function signal processing blocks with a programmable DSP. These are supported by a fully-flexible, all-digital audio mixing and routing engine with sample rate converters, for wide use-case flexibility. Two stereo headphone drivers each provide stereo ground-referenced or mono BTL outputs, with noise levels as low as  $2.3\mu V_{rms}$  for hi-fi quality line or headphone output. The codec also features stereo 2W Class-D speaker outputs, a dedicated BTL earpiece output and PDM for external speaker amplifiers. A signal generator for controlling haptics devices is included; vibe actuators can connect directly to the Class-D speaker output, or via an external driver on the PDM output interface. All inputs, outputs and system interfaces can function concurrently.

And most importantly it fits on top of Raspberry Pi perfectly which is ideal for a prototype.

## **2.3 Technology**

### **2.3.1 Python**

Python is a general-purpose, high-level programming language whose design philosophy emphasizes code readability. Python's syntax allows for programmers to express concepts in fewer lines of code than would be possible in languages such as C, and the language provides constructs intended to enable clear programs on both a small and large scale. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming styles. It features a fully dynamic type system and automatic memory management, similar to that of Scheme, Ruby, Perl and has a large and comprehensive standard library in which we used Dropbox library

### **2.3.2 Linux Shell Script**

Most of our operating systems including Linux can support different user interfaces (UI). The Graphical User Interface (GUI) is a user-friendly desktop interface that enables users to click icons to run an application. The other type of interface is the Command Line Interface (CLI) which is purely textual and accepts commands from the user. A shell, the command interpreter reads the command through the CLI and invokes the program. Most of the operating systems nowadays, provide both interfaces including Linux distributions. When using shell, the user has to type in a series of commands at the terminal. No problem if the user has to do the task only once. However, if the task is complex and has to be repeated multiple times, it can get a bit tedious for the user. Luckily, there is a way to automate the tasks of the shell. This can be done by writing and running shell scripts. A shell script is a type of file which is composed of a series and sequence of commands that are supported by the Linux shell.

### **2.3.3 Cron**

The software utility Cron is a time-based job scheduler in Unix-like computer operating systems. People who set up and maintain software environments use cron to schedule jobs (commands or shell scripts) to run periodically at fixed times, dates, or

intervals. It typically automates system maintenance or administration although its general-purpose nature makes it useful for things like connecting to the Internet and downloading email at regular intervals.

## 2.4 Arduino

Arduino is an opensource software for programming AVR microcontrollers specifically for Atmega328.

There are many hardware prototyping platforms available but Arduino is a good choice as it is flexible, offers various digital and analog inputs, SPI, I2C, a serial interface and digital and PWM outputs. It is backed up by a growing on-line community, lots of source code is already available and ready to be used.

Since it has Serial Communication we can use any USB to UART converter and connect to Raspberry Pi.

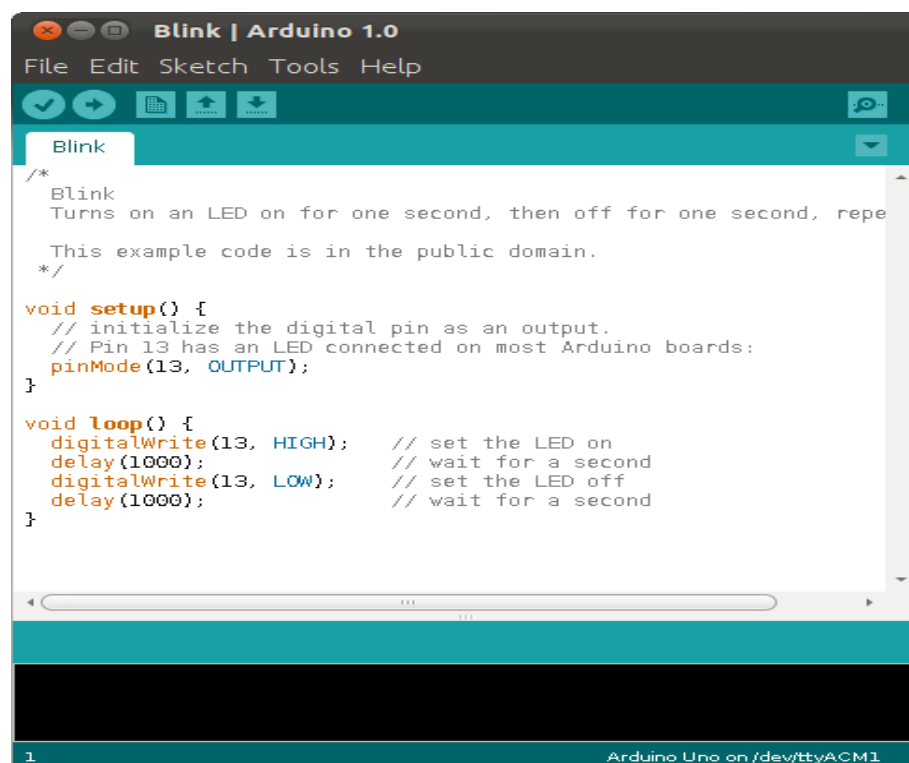


Figure 2.3: A screenshot of the Arduino IDE showing the Blink program.

## 2.5 Proposed Hardware

The following is the list of Hardware proposed

- **Raspberry Pi Model B.**  
It's the heart of the project, does all the processing, uploads data to the remote server
- **Wolfson Audio Card.**  
It has preamplifier and it's the one which records and sends audio data to Raspberry Pi.
- **Huawei e352s HSPA+ USB Modem.**  
Device to transfer data through cellular radio service
- **Arduino Pro Mini development board.**  
It's a microcontroller development board to control power on/off of raspberry pi and connect to peripherals to pull/push data through them.
- **Tiny RTC Module.**  
Keeps time even when Rasperrypi and Arduino are off. which is essential to determine Raspberri Pi's power on/off cycles
- **16x2 IIC LCD Module.**  
Display the current status of the device, battery percentage and relevent information about next recording.
- **Lithium-polymer Battery.**  
A 7.4V 10000mAh Battery is needed to power the device for oneday usage.
- **Max17043 Lipo Fuel Guage.**  
To get the battery percentage and pass it on to microcontroller so that it can take action accordingly.
- **Solar Panel.**  
A 15 Watt Solar Panel will be sufficient to charge our Batteries in a sunny day.
- **Voltage Regulator.**  
Electronics are very sensitive to high voltage and needs a regulator as the battery voltage is higher than operational voltage of most components.
- **Solar Charger.**  
Its a adjustable PWM capable Buck-Boost device which gives constant voltage irrespective of the voltage. Preferred to charge li-po batteries in constant voltage method.

# CHAPTER 3

## IMPLEMENTATION

This chapter briefly describes both the hardware and software implementation of the prototype recorder. Firstly setting up the hardware and then setup communication between one another.

### 3.1 Setting the Raspberrypi

#### 3.1.1 Installing Raspbian

Raspberry pi supports a lot of different flavors of linux. It is recommended to use Raspbian OS as it was developed by their own team. But for this OS we need to compile Wolfson audio card hardware drivers separately, So it is suggested to use the image provided by wolfson

- Download image from [http : //downloads.element14.com/wolfson/](http://downloads.element14.com/wolfson/)
- Extract the zip file.
- Extracted image file is copied to sdcard by using Win32Diskimager Software
- The SD card is plugged into the raspberry pi and powered on. After a few seconds the raspbian loads up.

#### 3.1.2 Installing XRDP

XRDP is used for remote desktop connection and can be installed from terminal in Raspbian OS by the following command.

```
% sudo apt-get update
% sudo apt-get install xrdp
```



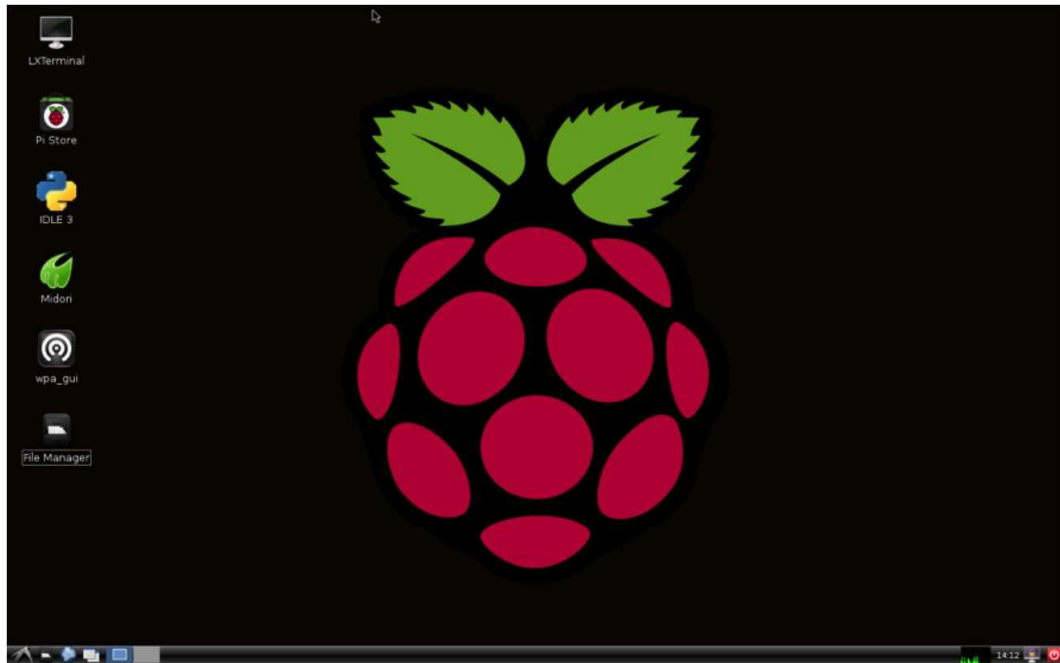


Figure 3.1: A screenshot of the Raspian OS.

## 3.2 Setting USB Modem

### 3.2.1 Installing Packages

USB Modem needs a separate software packages to work correctly. most USB Modems these days come up with a slot to Micro-SD card, Because of this linux will read the single device as two. to solve this problem we need to install a software called USB-ModeSwitch.

```
% sudo apt-get update
% sudo apt-get install usb-modeswitch
```

Now for making the connection to modem device we use this script by Sakis3g. Install this script by following commands. Now download the Sakis3g package.

```
% sudo wget "http://www.sakis3g.com/downloads/sakis3g.tar.gz"
```

Then unzip the file.

```
% sudo tar -xzf sakis3g.tar.gz
```

Make the file executable

```
% sudo chmod +x sakis3g
```

For making the script automatic and not ask for admin password each time it is used, we can change the script to a system default app and giving super user rights. To do so move the sakis3g file to /opt. using the following commands

Enter the following script on terminal

```
% sudo mkdir -p /opt/sakis3g/  
% sudo mv sakis3g /opt/sakis3g  
% sudo chown root:root /opt/sakis3g/sakis3g  
% sudo ln -s /opt/sakis3g/sakis3g /usr/bin
```

Now that we are capable of connecting to device, we need to install one more program called "PPP" which is a protocol for communicating to Cell Towers. Enter the following script on terminal

```
% sudo apt-get install ppp
```

### **3.2.2 Testing Connection**

Firstly call the sakis3g script by:

```
% sudo sakis3g --interactive
```

Which opens an interactive terminal like in Fig.3.2

Follow the on screen instuctions. select "connect to 3G" option and then Choose Appropriate "APN" Once you reached to the place like shown in Fig.3.3. Go to "More Options" and select "Generate Report". An example report is shown in Fig.3.4

### **3.2.3 Automate Connection**

To automate the above process create a configuration file for sakis3g script. Create a Configuration file.

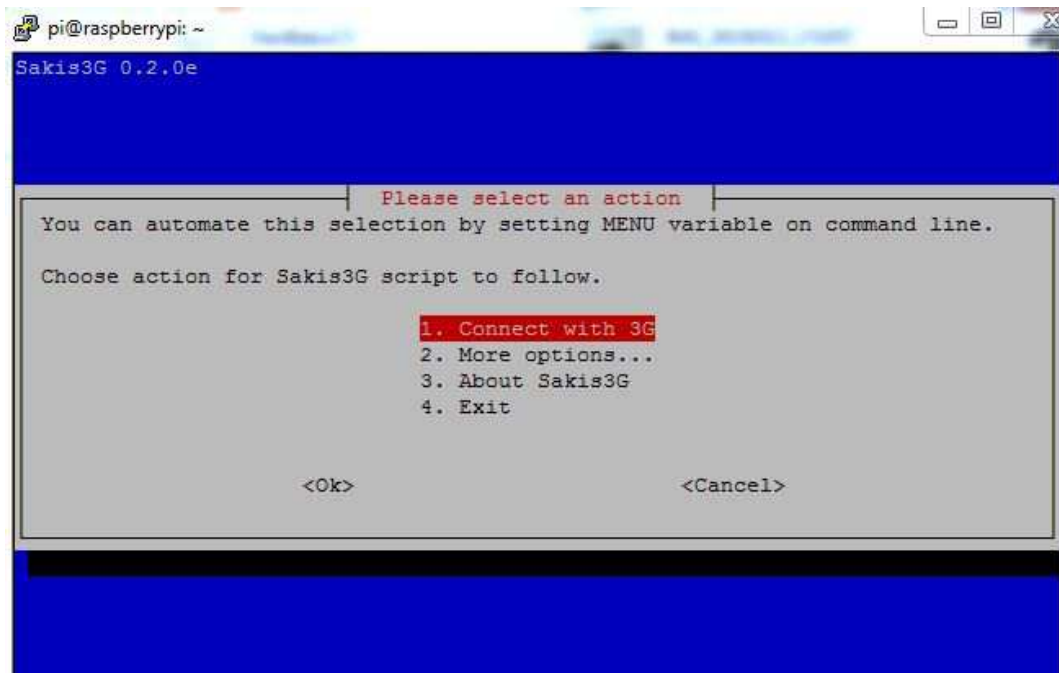


Figure 3.2: Sakis3g Interactive Screen.

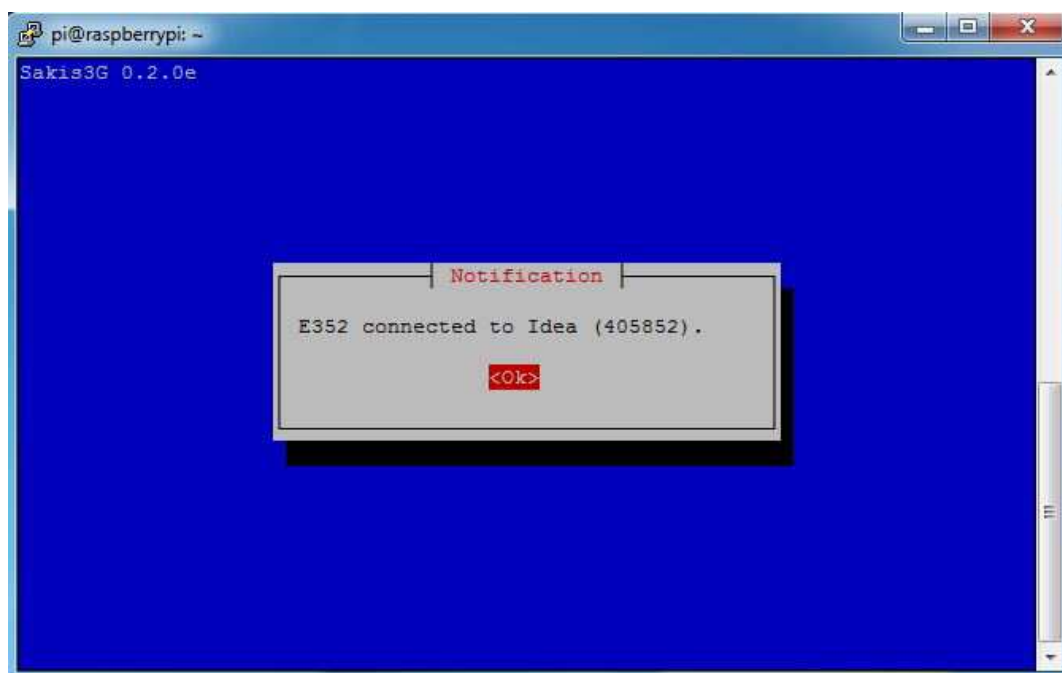


Figure 3.3: Successful Connection Message.

Add the following variables. You'll get these variables from Fig.3.4 and Modem ID by typing "lsusb" in the terminal. and look for the Modem ID in place of yellow box as in Fig.3.5

```
% sudo nano /etc/sakis3g.conf
```

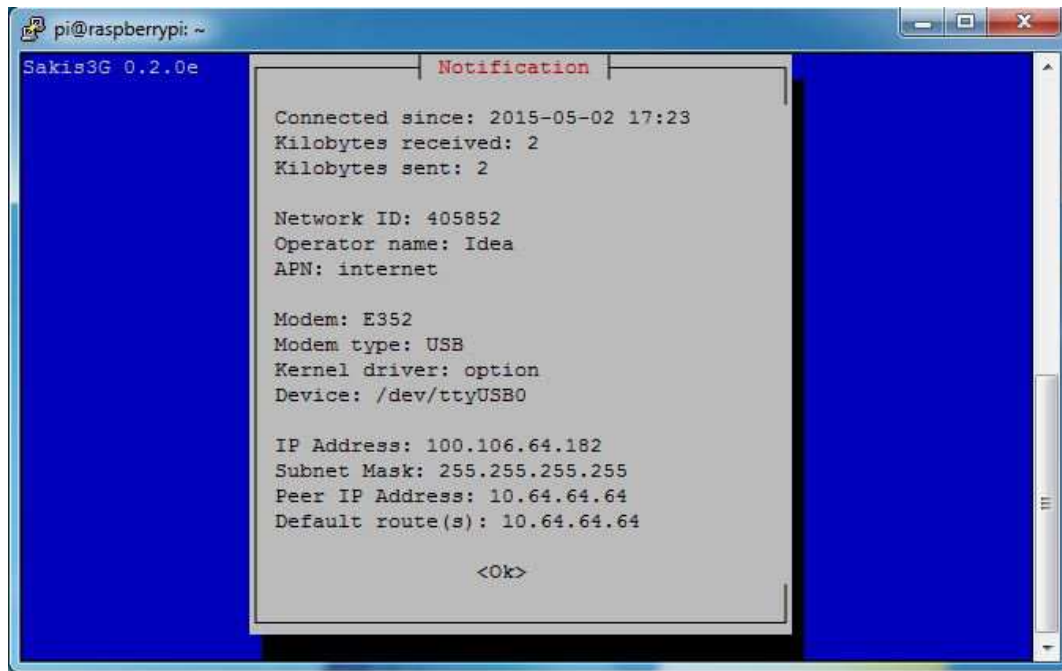


Figure 3.4: Connection Information.

```
pi@raspberrypi ~ $ lsusb
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 005: ID 12d1:1506 Huawei Technologies Co., Ltd. E398 LTE/UMTS/GSM Modem/
Networkcard
```

Figure 3.5: Connection Information.

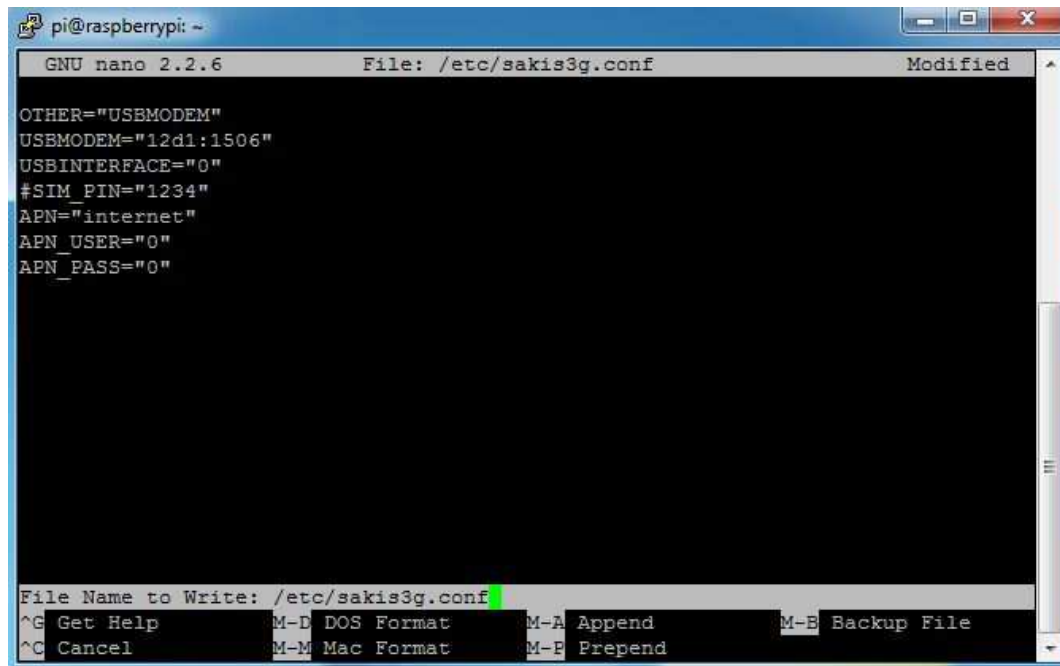
Add Following Variables. Fig.3.6 and save it by pressing "CTRL+X".

```
OTHER="USBMODEM"
USBMODEM="12d1:1506"
USBINTERFACE="0"
#SIM_PIN="1234"
APN="internet"
APN_USER="0"
APN_PASS="0"
```

Because every time you call sakis3g script it'll look into this configuration file and make a connection accordingly.

Now it can connect to internet with a single command.

```
% sudo sakis3g connect
```



```
pi@raspberrypi: ~
GNU nano 2.2.6 File: /etc/sakis3g.conf Modified
OTHER="USBMODEM"
USBMODEM="12d1:1506"
USBINTERFACE="0"
#SIM_PIN="1234"
APN="internet"
APN_USER="0"
APN_PASS="0"

File Name to Write: /etc/sakis3g.conf
^G Get Help      M-D DOS Format  M-A Append      M-E Backup File
^C Cancel        M-M Mac Format  M-P Prepend
```

Figure 3.6: Sakis3g Configuration File

### 3.3 Connecting to Dropbox

To be able to Upload data to dropbox with out typing username and password we needd to create a Dropbox Application which will serve as a back door for accessing the account. For creating the dropbox see in Fig.3.7 After creating the app we need to generate Access Token as in Fig.3.8 to enter through the back door.

Here is an example python code to upload a log file

```
# Include the Dropbox SDK
import dropbox
import json
# Generated Access Token
client =
    dropbox.client.DropboxClient('9UGIODPGhwYAAAAAAAAAqXKXTX')
print 'linked account:', client.account_info()
# For Uploading
f = open('C:/Users/Administrator/Desktop/RASPI/log.txt', 'rb')
response = client.put_file('/Bird_calls/log.txt', f)
print 'uploaded: ', response[u'path']
'''
decoded_data = json.loads(response)
print decoded_data
'''
```

## Create a new Dropbox Platform app

What type of app do you want to create?

 <b>Drop-ins app</b> Chooser or Saver	 <b>Dropbox API app</b>
---	--

To create a Dropbox for Business app, visit [the Dropbox for Business app creation page](#).

Can your app be limited to its own folder?

☐ Yes — My app only needs access to files it creates.

☒ No — My app needs access to files already on Dropbox.

What type of files does your app need access to?

☐ Specific file types — My app only needs access to certain file types, like text or photos.

☒ All file types — My app needs access to a user's full Dropbox.

Provide an app name, and you're on your way.

[Create app](#)

Figure 3.7: New Dropbox App

OAuth 2

**Redirect URIs**

[Add](#)

**Allow implicit grant** ⓘ

**Generated access token** ⓘ

9UGIODPGhwYAAAAAAAAAvnpUbvG3xsrYumkILBEYtmdODapCurozFma\_1NuNa6AA

This access token can be used to access your account (sailendra.marripudi@gmail.com) via the API. Don't share your access token with anyone.

Figure 3.8: Generate Access Token

## 3.4 Arduino and its peripherals

Setting up Arduino is straight forward just plug in USB to PC. In Arduino IDE select appropriate "Board" and "COM Port".

### 3.4.1 RTC

RTC stands for real time clock. It keeps track of time even though Arduino is powered off. RTC is powered by a coin cell and is expected to run for more than 3years. The module we used is "Tiny RTC" which is powered by DS3231SN chip which has native support to I2C Communication. It has ultra precise temperature compensated oscillator which gives accurate time. It has two alarms which on triggered pulls down the interruptPin on Arduino. Connection diagram is in Fig.3.10. For DS3231 we need to import RTCLib library from Ayars (2013–) into Arduino IDE

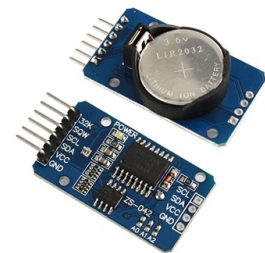


Figure 3.9: DS3231 Module

### 3.4.2 16x2 LCD

Lcd needs lot of GPIO pins on arduino which makes alot of wires and eventually connection problems. so used a 8bit shift register board which is controlled through I2C. It requires a library from Adafruit (2013–) Connection Diagram in Fig3.11

### 3.4.3 Lipo Fuel Guage

Lipo Fuel Guage Fig.3.12 is necessary for measuring the battery percentage It is based on Max17043 chip which has complex battery measuring algorithms built-in and can be accessed through I2C. which requires a library from Welters (2013–).

All the Peripherals are connected to the same I2C Bus.

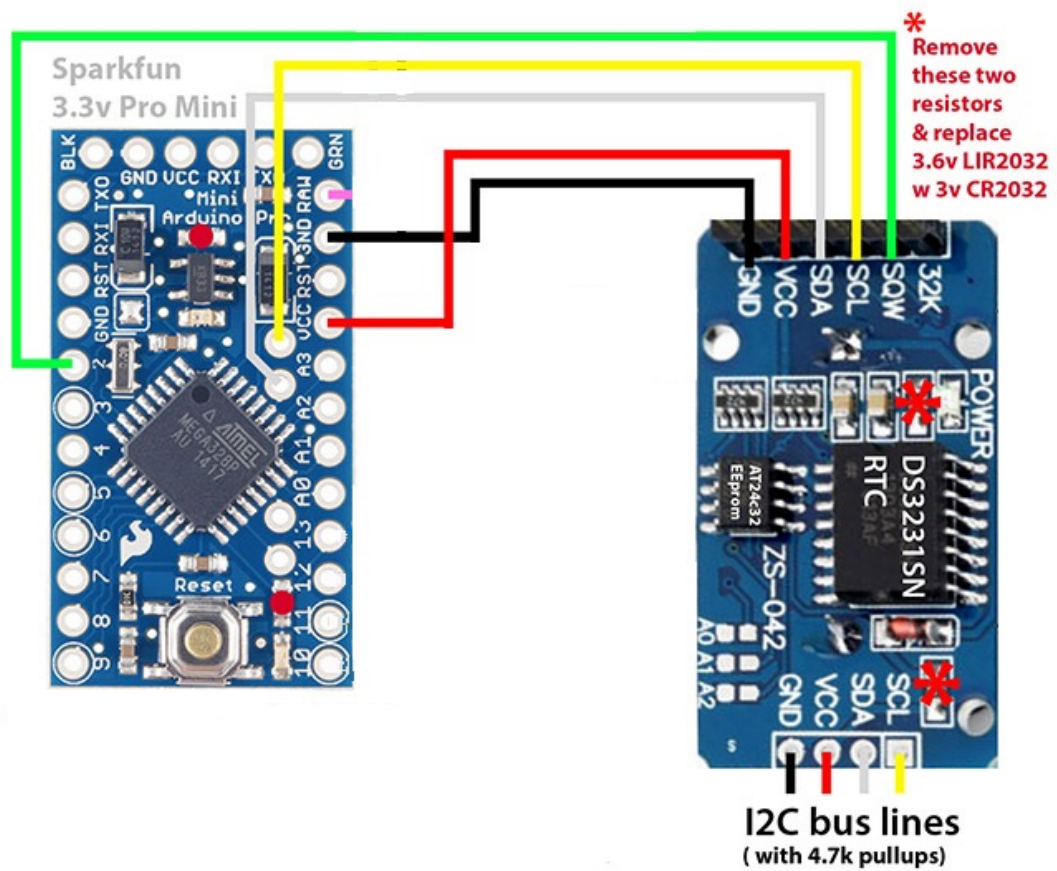


Figure 3.10: Arduino - Tiny RTC Connection Diagram

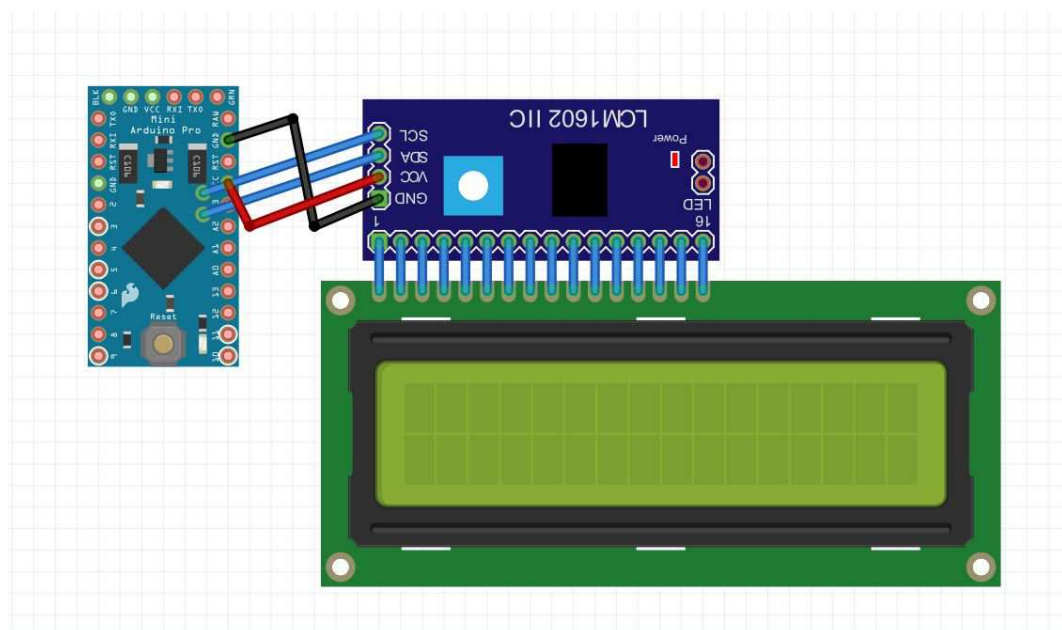


Figure 3.11: Arduino Lcd Connection Diagram



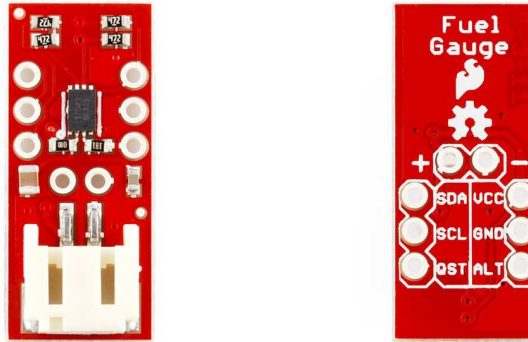


Figure 3.12: Lipo Fuel Gauge Front, Back

### 3.5 Communication between Arduino and Raspberry Pi

Because of Wolfson audio card all the GPIO pins of Raspberry Pi are blocked and we are only left with UART. So Arduino is connected to Raspberry Pi using a USB to UART connector chip CP2102 IC and is plugged directly to USB port of Pi.

For making a connection we need to install PySerial Python package using the following terminal command

```
% sudo apt-get update
% sudo apt-get install python-serial
```

On successful installation we can use python to communicate to Arduino using serial communication.

Here is a sample python code which will generate log of what Arduino is sending.

```
#!/usr/bin/env python
# This Program logs what ever is coming through Serial Port

import serial
import string
#import time
import datetime

test = serial.Serial("/dev/ttyUSB0",19200)
test.open()

try:
```

```

while 1:
    response = test.readline()
    file = open("/home/pi/Desktop/logfile.txt", 'a')
    file.write(datetime.datetime.now().strftime('%Y-%m-%d
            %H:%M:%S') + " ")
    # fromtimestamp(time.time())
    file.write(response)
    file.close()

except KeyboardInterrupt:
    test.close()

```

## 3.6 Solar charger

It's a Dc-Dc converter module based on XL6009 chip. capable of both Buck and Boost. It's auto adjustable Pulsewidth Modulation at  $400kHz$  keeps the output ripple stable and small. For small solar applications like this these type of modules are well suited as the maximum conversion efficiency is above 94%. It has wide input voltages from  $3.8V$   $32V$ . And output voltage varies from  $1.25V$   $35V$ . Max current supported is  $4A$ . which makes it ideal for prototyping. Since the Battery Voltage at maximum capacity is  $8.4V$  we adjusted the solar charger output to be  $8.5V$ , which gives better efficiency of the solar panel as the solarpanel voltage fluctuates around  $9.5V$  to  $10V$ .



Figure 3.13: Solar charger

## 3.7 Block Diagram

This is the overall Block Diagram of the Project

## 3.8 Schematics

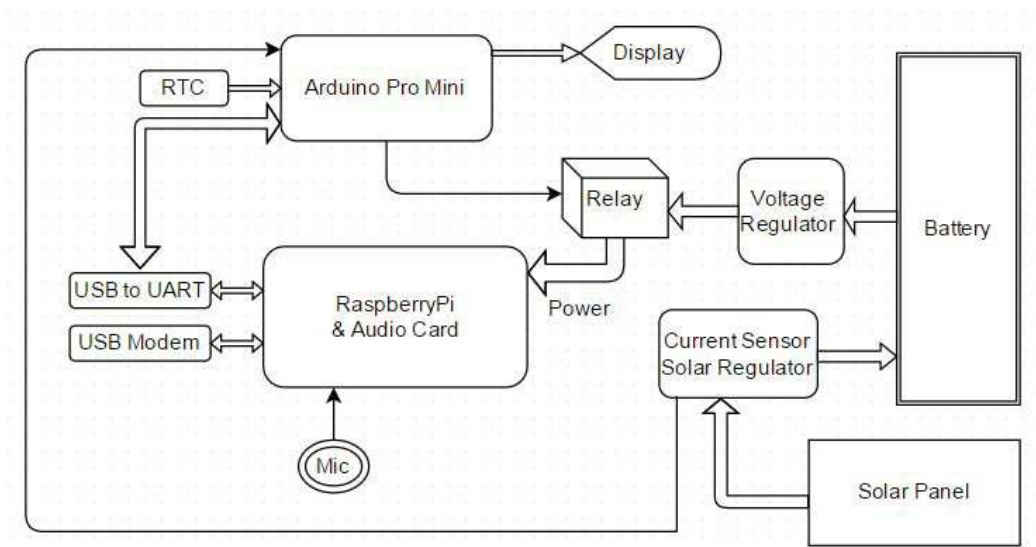


Figure 3.14: Block Diagram Of The Recorder

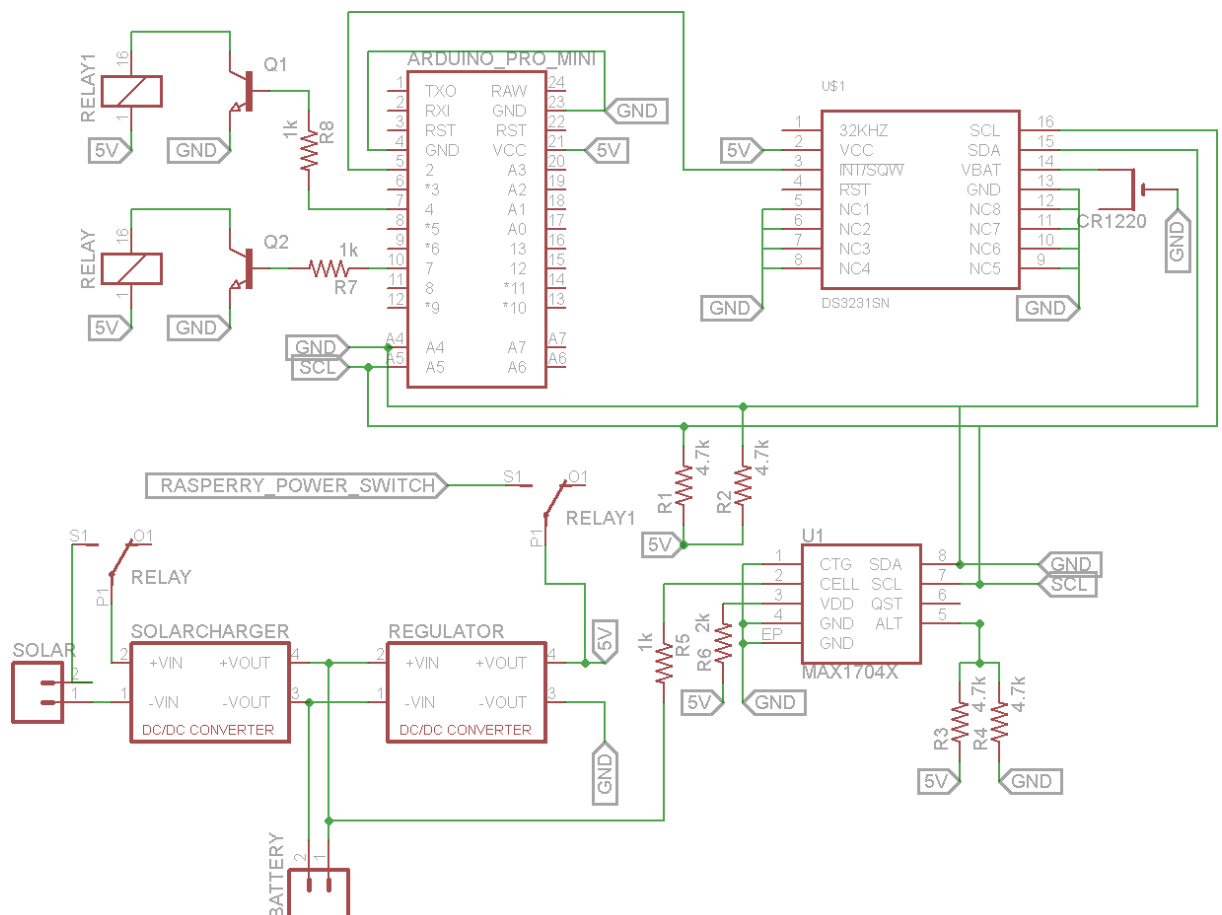


Figure 3.15: Schematics Of The Interface Board

Table 3.1: List of Parts and Values

Name	Value
$R1$	$4.7K$
$R2$	$4.7K$
$R3$	$4.7K$
$R4$	$4.7K$
$R5$	$1K$
$R6$	$2K$
$R7$	$1K$
$R8$	$1K$

## CHAPTER 4

## Working

## 4.1 Flow Chart

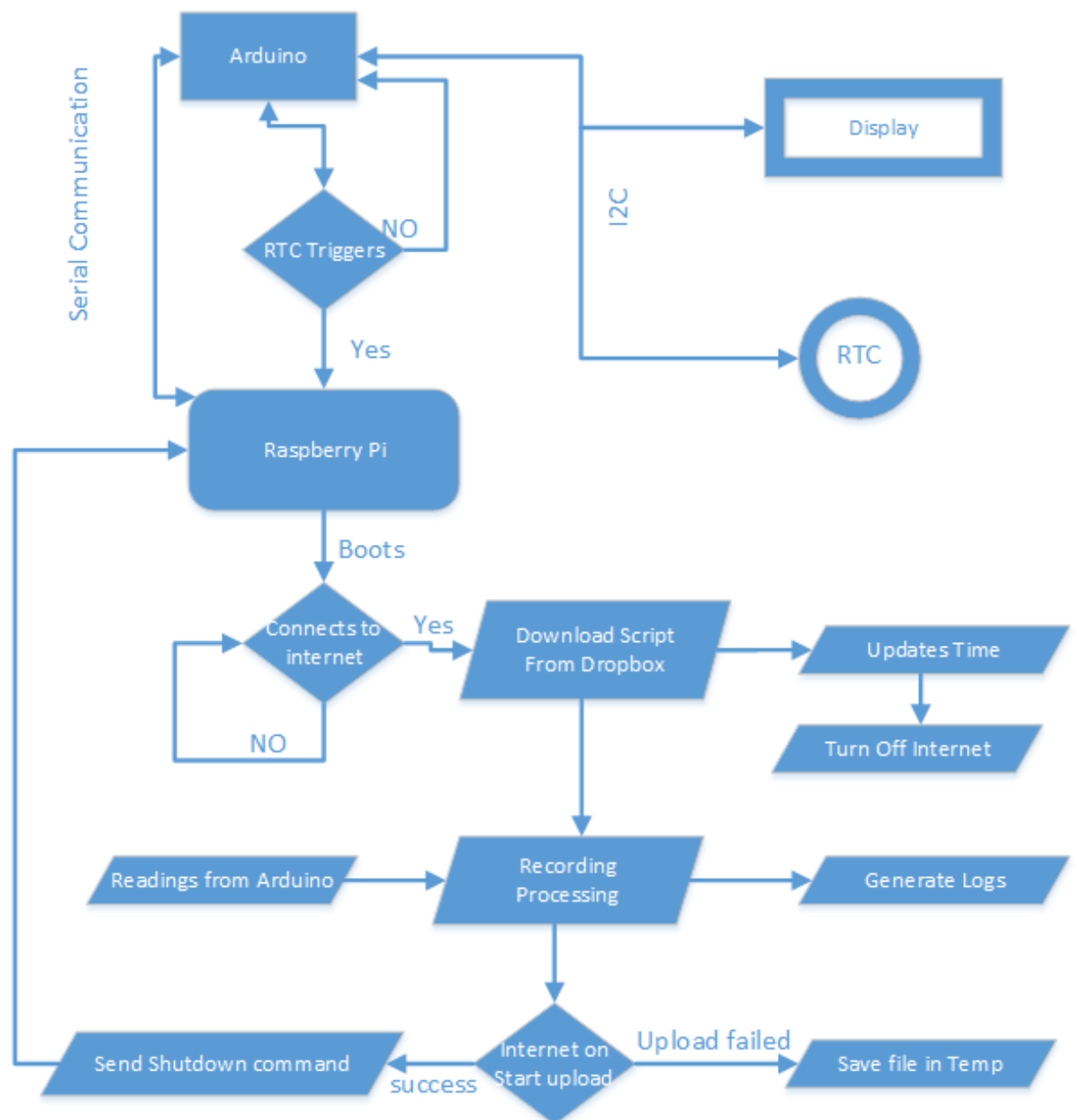


Figure 4.1: Process Chart of the Recorder

## 4.2 Process Flow

Everyday birds generally start singing in a specific time, so accordingly arduino wakes up raspberry pi. As per the request we make that twice a day at 6AM and 6PM. As soon as the raspberry wakes up it first tries to connect to internet(DialUp USB Modem), If that is Successful then it'll get the configuration file from dropbox which has all the necessary variables like

- Recording rate
- Recording Duration
- Cut of Threshold
- Volume Gain
- External Mic or Internal Mic
- Dropbox Folder Path

Once the configuration file is successfully downloaded. It connects to NTP server and gets time and parallelly look for any files in temprary folder to be uploaded. If yes it'll upload and deletes that file from the folder. And closes the internet connection. Gets data from Arduino about the battery Percentage, Temperature and Humidity etc. Updates the Log files.

The recording song is directly Piped to a python script which will perform a simple threshold algorithm with the desired threshold from the user and saves it in a file and checks the interent connection. If not connected dials the USB Modem and uploads the saved file to the desired Dropbox Folder. Updates the Log, and uploads the Log and sends the Shut down Command to Arduino.

Arduino will turn off the Power to Raspbery Pi. updates the status in LCD and waits for the next trigger time.

# CHAPTER 5

## Results and Conclusion

### 5.1 Overall Energy Consumption

The Solar Panel we have is  $15Whr$  It is generating around  $45w$  to  $49w$  in a day according to the sunlight on that particular day.

Device	Consumption
Raspberry Pi with Audio Card	$2.2Whr$
Arduino Pro mini with Pheripherals	$0.6Whr$
Arduino when Sleep	$0.09Whr$
Arduino With LCD Backlight On	$1.4Whr$
USB Data card Idle	$0.5Whr$
USB Data card Transferring	$2.5Whr$

Table 5.1: Energy consumption.

Device	ON Time
Raspberry Pi with Audio Card	$5hrs$
Arduino Pro mini with Pheripherals	$5hrs$
Arduino when Sleep	$18hrs$
Arduino With LCD Backlight On	$5mins$
USB Data card Idle	$30mins$
USB Data card Transferring	$1hr$

Table 5.2: Device On Time.

From the energy consumption table and the typical device usage in a day total energy consumed in a day is  $27.75W$  approximately  $30W$ .

## 5.2 Battery Charging

Fig.5.1 shows the battery percentage with time when charged with a solar panel for 1hour and 20minutes. Since we are operating only in constant Voltage charging mode battery will not reach its maximum capacity. It'll reach till 85percent.

So total time taken to charge from 0-85 is around 6hrs.

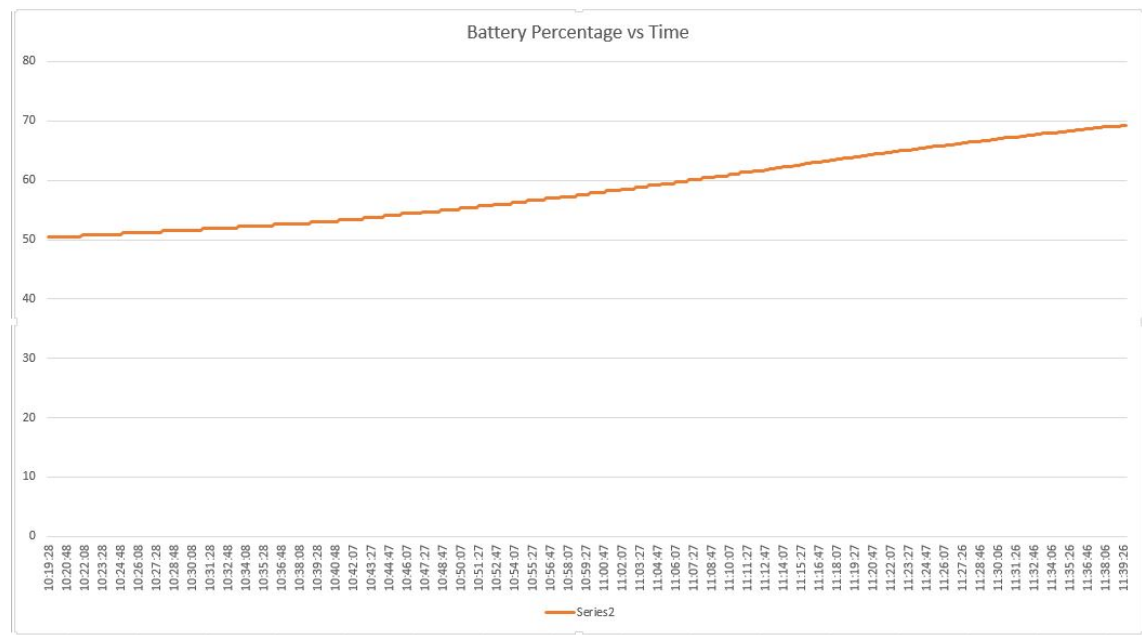


Figure 5.1: Battery Percentage vs Time.

## 5.3 Threshold Output

Fig.5.2 shows the Original Magpie Robin recorded song and the generated song after thresholding, and the deleted song. The generated song was only 51sec as the original song was 152sec. The rest 100sec data is pure Noise. we'll eliminate this noise and transfer only the generated Song.



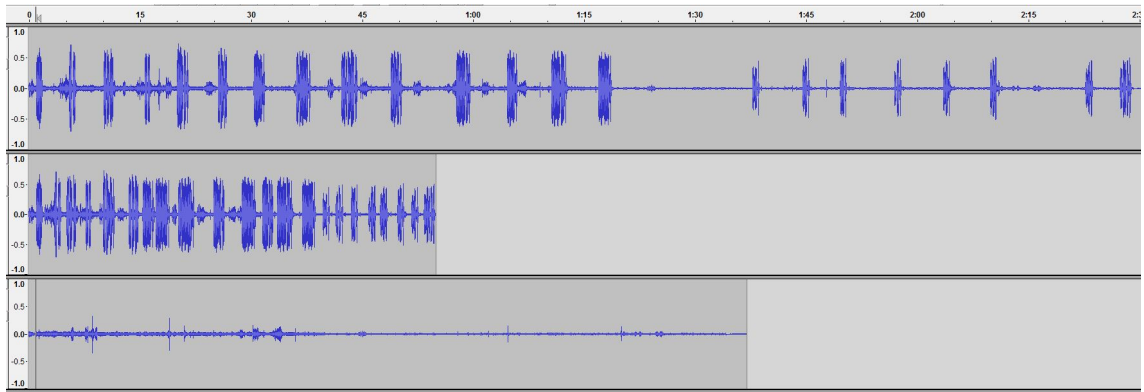


Figure 5.2: Bird Song: Original, After threshold Generated, Deleted

## 5.4 Assembled Device

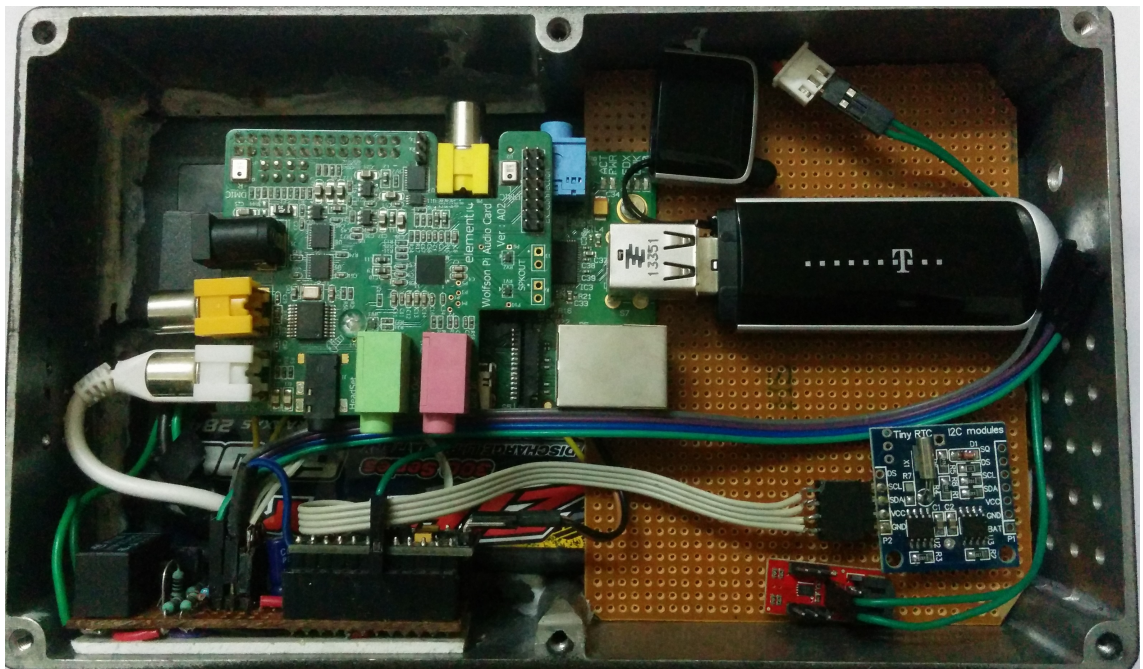


Figure 5.3: Assembled Recorder

## 5.5 Future Scope

List of Improvements in the future.

- Make a single board with all the peripherals on it as in the Schematics.
- Implement bird recognition algorithm, then and there to upload the data.
- Still reduce the amount of data to upload by after performing intelligent way of threshold algorithm.

# APPENDIX A

## Main Code

```
#!/usr/bin/env python
# Include the Dropbox SDK
# Importing the necessary packages
import dropbox
import os
import serial
import string
import time
import datetime

arduino = serial.Serial("/dev/ttyUSB0",19200)
#Connect to Internet
os.system("sudo sakis3g connect")

#update time
os.system("sudo dpkg-reconfigure ntp")

time.sleep(10)

day = datetime.datetime.now().strftime('%Y%m%d')
logfile = '/home/pi/logs/'+day+'.txt'

#Connect to Dropbox
client =
    dropbox.client.DropboxClient('9UGIODPGhwYAAAAAAAAAqXKXTXdkUnUG1XH')
#Downloading Script
f, metadata =
    client.get_file_and_metadata('/Bird_calls/settings.sh')
out = open('/home/pi/Desktop/settings.sh', 'wb')
out.write(f.read())
out.close()
#Updating Log
file = open(logfile, 'a')
file.write(datetime.datetime.now().strftime('%Y-%m-%d
    %H:%M:%S'))
file.write("-----Hello System
    Started!-----\n")
file.write("\n")
file.write(string(metadata))
file.write("\n")
```

```
arduino.open()
arduino.write("get_data")
time.sleep(1)
file.write(arduino.readline())
file.write("\n")
file.close()
arduino.close()

#Give Admin Preivilages
os.system("sudo chmod +x /home/pi/Desktop/settings.sh")
os.system("sudo chown root:root /home/pi/Desktop/settings.sh")
os.system("sudo /home/pi/Desktop/settings.sh")
```

# APPENDIX B

## Threshold Code

```
#!/usr/bin/env python
# Include the Dropbox SDK
# Importing the necessary packages
import wave, struct
import numpy as np
import sys
import time
import os
import serial
import string
import dropbox
import datetime

total_duration = int( sys.argv[1])
Morning_hour = int( sys.argv[2])
Morning_minute = int( sys.argv[3])
Evening_hour = int( sys.argv[4])
Evening_minute = int( sys.argv[5])
threshold = int( sys.argv[6])
day = datetime.datetime.now().strftime('%Y%m%d')
dt = datetime.datetime.now().strftime('%Y%m%d_%H%M')

logfile = '/home/pi/logs/'+day+'.txt'
proc_rec = '/home/pi/Recordings/proc_rec'+dt+'.wav'
del_rec = '/home/pi/Recordings/del_rec'+dt+'.wav'

arduino = serial.Serial("/dev/ttyUSB0",19200)
arduino.open()
arduino.write("lcd")
time.sleep(1)
arduino.write("Recording")
arduino.close()

file = open(logfile, 'a')
file.write(datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S') + " ")
# fromtimestamp(time.time())
file.write("Started_Recording")
file.close()

# Reading the .Wav file using wave library and get the all
parameters
```

```

infile = wave.open(sys.stdin)
num_channels = infile.getnchannels()
sampling_rate = infile.getframerate()
sample_width = infile.getsampwidth()
#num_frames = infile.getnframes()
#print num_channels,sample_width,sampling_rate,num_frames
num_frames = total_duration*sampling_rate
# section for declaring variables
_const = 1000
#threshold =0
window = sampling_rate # 1sec of data

# create output files
outfile = wave.open(proc_rec,'w')
outfile.setparams((num_channels, sample_width, sampling_rate,
    0, 'NONE', 'not compressed'))
delfile = wave.open(del_rec,'w')
delfile.setparams((num_channels, sample_width, sampling_rate,
    0, 'NONE', 'not compressed'))
# condition on number of channels

data = []
for i in xrange(int(num_frames/window)):
    try:
        wavedata = infile.readframes(window)
        data_segment =
            struct.unpack('{}h'.format((window)*num_channels),
                wavedata)
        data_segment = list(data_segment)
        gen = np.array(data_segment)
        gen = gen/_const
        gen2 = np.square(gen)
        energy_segment = np.sum(gen2[0:window])
        if energy_segment>=threshold:
            outfile.writeframes(wavedata)
        else:
            delfile.writeframes(wavedata)
    except KeyboardInterrupt:
        pass
infile.close()
outfile.close()
delfile.close()

# Files are generated we need to upload them
file = open(logfile, 'a')
file.write(datetime.datetime.now().strftime('%Y-%m-%d
    %H:%M:%S') + " ")
    # fromtimestamp(time.time())
file.write("Recording, Processing Done.")
file.close()

#Conect to Internet
#os.system("sudo sakis3g connect")

```

```

#Dropbox from here only.
client =
    dropbox.client.DropboxClient('9UGIODPGhwYAAAAAAAAAqXKXTXdkUnUG1XH')
file = open(logfile, 'a')
file.write(datetime.datetime.now().strftime('%Y-%m-%d
    %H:%M:%S') + " ")
file.write("Linked Account :")
file.write(client.account_info())
file.close()

f = open(proc_rec , 'rb')
path = "/Bird_calls/Recordings/"+proc_rec+dt+".wav"
response = client.put_file(path, f)
file = open(logfile, 'a')
file.write(datetime.datetime.now().strftime('%Y-%m-%d
    %H:%M:%S') + " ")
file.write("Uploaded :\n")
file.write(response)
arduino.open()
arduino.write("get_data")
file.write(datetime.datetime.now().strftime('%Y-%m-%d
    %H:%M:%S') + " ")
file.write(arduino.readline())
file.write("\n")
file.write("Setting Alarms!\n")
arduino.write("setalarm_1")
arduino.write(Morning_hour+', '+Morning_minute)
file.write(arduino.readline())
time.sleep(1)
arduino.write("setalarm_2")
arduino.write(Evening_hour+', '+Evening_minute)
file.write(arduino.readline())
file.write("Alarms Set!\n")
file.write("Shutting Down!\n")
file.write("-----")
file.close()
arduino.close()
g = open(logfile, 'rb')
path2 = '/Bird_calls/logs/log'+day+'.txt'
response_log = client.put_file(path2, g)

time.sleep(10)
#Send Shutdown Command to Arduino
arduino.open()
arduino.write("lcd")
time.sleep(1)
arduino.write("Sleep Mode On")
time.sleep(1)
arduino.write("shutdown")
arduino.close()
# Shutting Down
os.system("sudo shutdown -h now")

```

# APPENDIX C

## Arduino Code

```
// Date, Time and Alarm functions using a DS3231 RTC connected
// via I2C and Wire lib
// Include Libraries //
#include <Wire.h>
#include <SPI.h> // not used here, but needed to prevent a
    RTCLib compile error
#include <avr/sleep.h>
#include <RTCLib.h>
#include <LiquidCrystal_I2C.h>
#include <MAX17043.h>

// Declare Objects //
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
    // Set the LCD I2C address
RTC_DS3231 RTC;
MAX17043 fuelGauge;

int INTERRUPT_PIN = 2;
volatile int state = LOW;
int currentPin = A2;
int currentValue = 0;
int relayPin = 5;
char character;
char character1;

String cmd = "";
String cmd1 = "";

void setup() {
    pinMode(INTERRUPT_PIN, INPUT);
    pinMode(relayPin, OUTPUT);

    //pull up the interrupt pin
    digitalWrite(INTERRUPT_PIN, HIGH);

    Serial.begin(19200);
    Wire.begin();
    delay(10);
    RTC.begin();
    delay(10);
    fuelGauge.begin();
    delay(10);
```

```

    lcd.begin(16,2);
    delay(10);
    //RTC.adjust(DateTime(__DATE__, __TIME__));
    batterystate();
    delay(10);
    //DateTime now = RTC.now();
    //RTC.setAlarm1Simple(21, 15);
    //RTC.setAlarm2Simple(21, 14);
    //RTC.turnOnAlarm(1);
    //RTC.turnOnAlarm(2);
    delay(100);
    if (RTC.checkAlarmEnabled(1) && RTC.checkAlarmEnabled(2)) {
        Serial.println("Alarms Enabled");
    }
    //attachInterrupt(0, logData, LOW);
    delay(100);
    lcd.write("Hello World!");
}

void loop() {
    //Serial.println("entering loop");
    //delay(100);
    if(RTC.checkIfAlarm(1) || RTC.checkIfAlarm(2)){
        Serial.println("RaspberryPi ON");
        digitalWrite(relayPin, HIGH);
        RTC.turnOffAlarm(1);
        RTC.turnOffAlarm(2);
    }

    if(Serial.available()){
        delay(100);
        while(Serial.available()) {
            character = Serial.read();
            cmd.concat(character);
        }
        //Serial.println(cmd);
    }
    if(cmd.equals("shutdown")){
        delay(10000);
        Serial.println("RaspberryPi OFF");
        digitalWrite(relayPin, LOW);
        sleepNow();
    }
    else if(cmd.equals("get_data")){
        getdata();
    }
    else if(cmd.equals("get_time")){
        gettime();
    }
    else if(cmd.equals("setalarm_1")){
        delay(1000);
        int value1 = Serial.parseInt();
        delay(10);
    }

```



```

    int value2 = Serial.parseInt();
    delay(10);
    String text1 = "Alarm_1 is set at ";
    text1.concat(value1);
    text1.concat(":");
    text1.concat(value2);
    Serial.println(text1);
    RTC.setAlarm1Simple(value1, value2);
    delay(10);
    RTC.turnOnAlarm(1);
}
else if(cmd.equals("setalarm_2")){
    delay(1000);
    int value3 = Serial.parseInt();
    delay(10);
    int value4 = Serial.parseInt();
    delay(10);
    String text2 = "Alarm_2 is set at ";
    text2.concat(value3);
    text2.concat(":");
    text2.concat(value4);
    Serial.println(text2);
    RTC.setAlarm2Simple(value3, value4);
    delay(10);
    RTC.turnOnAlarm(2);
}
else if(cmd.equals("lcd")){
    delay(1000);
    while(Serial.available()) {
        character1 = Serial.read();
        cmd1.concat(character1);
    }
    lcd.clear();
    //lcd.write(cmd1);
}
cmd = "";

}

void sleepNow() {
    Serial.println("Going to Sleep");
    delay(50);
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    sleep_enable();
    attachInterrupt(0, logData, LOW);
    sleep_mode();
    //HERE AFTER WAKING UP
    sleep_disable();
    detachInterrupt(0);
}

void logData() {

```

```

    //do something quick, flip a flag, and handle in loop();
}

void batterystate(){
    Serial.print("Battery Voltage: ");
    Serial.println(fuelGauge.getBatteryVoltage());
    delay(10);
    Serial.print("Battery Percentage: ");
    Serial.println(fuelGauge.getBatteryPercentage());
    //Serial.flush();
    //while(Serial.available()>0) Serial.read();
}

void getdata(){
    /*float currentValue = 0;
    for(int i = 0; i < 1000; i++) {
        currentValue = currentValue + (0.0264 * analogRead(currentPin)
            -13.51);
        delay(1);
    }
    */
    // this will send battery_voltage,battery_percentage.
    String stringser = "";
    stringser.concat("Battery Voltage :");
    stringser.concat(fuelGauge.getBatteryVoltage());
    stringser.concat("V, Battery Percentage :");
    stringser.concat(fuelGauge.getBatteryPercentage());
    stringser.concat("%");
    Serial.println(stringser);
}

void gettime(){
    DateTime now = RTC.now();
    Serial.print(now.hour(), DEC);
    Serial.print(':');
    Serial.print(now.minute(), DEC);
    Serial.print(':');
    Serial.print(now.second(), DEC);
    Serial.println();
}

```

# APPENDIX D

## Dropbox Script

```
#!/bin/bash
Vol1="128" # 128 is (0db)
Vol2="32" # 32 is (0db)
Recording_rate="16000"
Rec_duration="60" # 2hrs = 7200
Morning_hour="06"
Morning_minute="00"
Evening_hour="18"
Evening_minute="00"
Threshold="1000"
#-----Recording from External Microphone-----
amixer -Dhw:0 cset name='IN1R Volume' $Vol1
amixer -Dhw:0 cset name='AIF1TX1 Input 1' IN1R
amixer -Dhw:0 cset name='AIF1TX1 Input 1 Volume' $Vol2
amixer -Dhw:0 cset name='AIF1TX2 Input 1' IN1R
amixer -Dhw:0 cset name='AIF1TX2 Input 1 Volume' $Vol2
amixer -Dhw:0 cset name='Headset Mic Switch' on
sudo arecord -d $Rec_duration -r $Recording_rate -c 1 -f
    S16_LE | sudo python threshold.py $Rec_duration
    $Morning_hour $Morning_minute $Evening_hour $Evening_minute
    $Threshold
```

## REFERENCES

1. **Adafruit** (2013–). Arduino i2c lcd library. URL <https://github.com/adafruit/LiquidCrystal/>.
2. **Ayars, E.** (2013–). Arduino ds3231 library. URL <https://github.com/darkazazeal/rtcLib/>.
3. **Dropbox** (2012–). Dropbox python sdk. URL <https://www.dropbox.com/developer/>.
4. **Ragner.Jenson** (2014–). Wolfson audiocard community. URL <http://www.element14.com/community/>.
5. **Welters, A.** (2013–). Arduino lipo-fuelguage library. URL <https://github.com/awelters/LiPoFuelGauge/>.