## **Embedded Systems Development from the Ground Up**

**Santhosh Jayarajan**

**Preface**

The field of electronics and microcontrollers have advanced so fast that it is now possible to program almost anything, if you have the imagination.

The microcontroller options available today are mind boggling and choosing the right controller for the job can become frustrating and overwhelming. The Arduino platform itself offers multiple options and choosing the right option can be difficult. This book is divided into three parts – The first part starts with the small 8 pin microcontroller the aTiny85 which can be used for small but powerful projects. The second part moves up a step to the Arduino with a Python interface and brings out the power of the Arduino and the programming power of Python. The third and final part deals with the ESP 32 and programming with Micro python which is a scaled down version of Python for the microcontroller environment.

This book is not about theory or data sheets but about actual construction and coding and getting the work done with the minimum of resources.

Once anyone has covered the three parts it would be easy to decide the platform you need for your next big project. The complexity and the size of the project will decide the platform to use, but however big and complex your project may be, you can decide the platform to use.

Programming languages are now slowly penetrating the microcontroller domain and with these languages comes power and scalability.

Welcome to the fascinating world of electronics, programming and imagination!

**Introduction**

This book is about doing and is not about the theory which can be mundane when we are in the field of microcontrollers.

Microcontrollers are designed to do thing and do them fast.

This Book is divided into 3 Parts which covers the various levels and sizes of the microcontroller from the very small to the very powerful.

* Part-1 is about a*” little giant*” of the microcontroller world called the **aTiny85** which has small size very few I/O but can be used to develop very powerful and useful projects. The project that will be developed in this part is called “My Magic Lamp” and is a Bluetooth enabled table lamp that can be controlled from a mobile phone. You can set the intensity and even schedule the on time using your mobile phone. This part also introduces n online resource to program an android phone called App Inventor which is an amazing platform from MIT and also how to program the Atiny 85 using the Arduino ISP (*In Circuit Serial Programmer*).
* Part-2 is about the **Arduino UNO** which is a work horse in the Arduino world. This microcontroller has been the star of many useful and sophisticated projects. This part will interface the Arduino with Python to create a GUI based program to view and plot the Analog channel values which are sent on the serial port using ***Firmata*** *on the Arduino and* ***PyFirmata*** *in the Python program*.

Matplotlib is the library that renders the plots. This part also introduces a wonderful IDE for Python called PyCharm that is a “charming” way to program in Python.

This book is not about Python programming so the details and syntax of python programming will not be discussed.

* The Part-3 is the final part that uses the **ESP 32** from Xpresif is the most powerful microcontroller system around. This microcontroller can be programmed in Micropython making it an ideal choice for large sophisticated projects.

So, there you have it the entire microcontroller spectrum from the smallest to the most powerful all with simple but powerful projects that you can make.

Welcome to a world of microcontroller development from the ground up!

**Contents**

1. **Part-1 – The ATiny 85 Microcontroller**
2. Introduction and basic details
3. Making the ATiny 85 programmer with Arduino as ISP.
4. My Magic Lamp Program and circuit on the ATiny 85.
5. App Inventor for Mobile interface.
6. Conclusion of Part-1
7. Resources for the Project
8. **Part-2 – Arduino UNO meets Python**
9. Introduction and basic details.
10. The Arduino Uno
11. The Standard Firmata Library
12. Python and the Pyfirmata Library
13. Datalogger project with Matplotlib and Excel.
14. The Python program in detail
15. Conclusion of Part-2
16. Resources for the Project
17. **Part-3 – ESP 32 programming with MicroPython**
18. Introduction and basic details.
19. The ESP-32
20. The Standard Libraries
21. Programming the ESP-32 with Thonny
22. Weather Station with ESP-32
23. Conclusion of Part-3
24. Resources for the Project
25. **Part-4 – Introduction to IoT (Internet of Things)**

i. Introduction and basic details.

1. **Conclusion**
2. **Development Ideas**
3. **Index**

**Part-1**

**ATtiny 85 - Power in Small Packages**



1. **Introduction and Basic details**

The ATtiny 85 is a 8 Pin microcontroller that is small and powerful and ideal for small projects that need only the basic of interfaces. Even with this small foot print the microcontroller manages of have Analog to Digital (ADC) pins and a few digital Pins.

The basic details of the microcontroller are as follows:

|  |  |  |
| --- | --- | --- |
| Sl.No | Parameter | Value |
| 1. | CPU Core | AVR |
| 2. | Program Memory Size | 8 KB |
| 3. | Input Outputs | 6 |
| 4. | Operating Voltage | 2.7 to 5.5 V |
| 5. | RAM Size | 512 Bytes |
| 6. | Max Clock | 20 Mhz |

***Table-1: ATtiny 85 Basic details***

The striking feature of this miniature chip is the wide range of supply voltage and the high operating frequency possible. The program memory is also decent for small projects but if many variables are used the small RAM may become a matter of concern.

The maximum clock of 20 Mhz is decent for small projects.

The basic Pin details are as follows:

|  |  |  |
| --- | --- | --- |
| Sl.No | Pin | Allocation |
| 1. | 1 | Reset Pin |
| 2. | 2 | Analog-3 /GIO-3 |
| 3. | 3 | Analog-2 /GIO-4 |
| 4. | 4 | Supply Ground |
| 5. | 5 | GPIO-0(PWM) |
| 6. | 6 | GPIO-1(PWM) |
| 7 | 7 | Analog-1/GPIO-2 |
| 8. | 8 | Supply VCC |

***Table-2: ATtiny 85 Pin allocation***

The Pin -1 is a reset pin used for programming and a low level will generate a reset. The Pins-4 and 8 are the supply ground and VCC respectively.

This chip has 3 ADC channels at pin 2,3 and 7.The General Input Out Pins(GPIO) are at pins 2,3,5,6,7.These can be configured for Digital Inputs, Outputs and pins 5 and 6 can be used for PWM(Pulse Width Modulation).Pulse width modulation is a timer based control that can control the ON time of an output with reference to its OFF time or its duty cycle. This technique is very useful for controlling the speed of motors and intensities of LEDs etc.

The VCC pin will accept the voltage input to the chip and is normally 5 V from a 5 V regulator. Another great feature of this chip is that it has a built-in crystal and hence no external components are used to provide the clock.

There are many more characteristics and details of this chip which are not relevant to the discussion and are not needed for any of the projects. The summary of the 200-page odd data sheet is available for download here:

[***https://www.alldatasheet.com/datasheet-pdf/pdf/174761/ATMEL/ATTINY85.html***](https://www.alldatasheet.com/datasheet-pdf/pdf/174761/ATMEL/ATTINY85.html)

The Pin details of the ATtiny 85 chip are given below:

1

8

5

4

**GPIO0(PWM)**

**GPIO1(PWM)**

**ADC-1/GPIO2**

**VCC(5V)**

**Ground**

**ADC-2/GPIO4**

**Reset**

**ADC-3/GPIO3**

**ATtiny 85**

***Fig-1 ATtiny 85 Pin Details***

1. **Making the ATiny Programmer with Arduino ISP**

Before you can do anything sensible with the ATtiny 85 it has to be programmed. The easiest way to program the ATiny 85 is to use an Arduino Uno and use it in the ISP Mode (In system Programming).

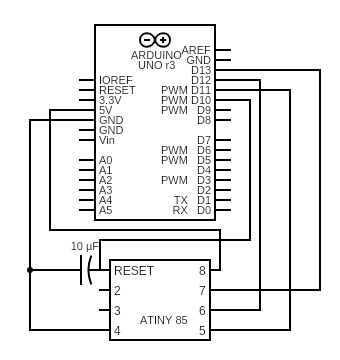
The programmer can be prepared using the following steps:

* Wiring the circuit between the ATtiny 85 and the Arduino ISP.
* Programming the Arduino with the Arduino ISP sketch.
* Installing the necessary ATtiny 85 libraries.

The circuit diagram for the ATiny85 programmer is as shown below. The connection of the various pins are as given in the table below:

|  |  |  |
| --- | --- | --- |
| **ATtiny85 IC Pin** | **Arduino Uno Pin** | **Remarks** |
| 7 | 13 |  |
| 6 | 12 | ATiny85 GPIO-1 |
| 5 | 11 | ATiny85 GPIO-0 |
| 1 | 10 | Reset pin for the ATiny85 |
| 4 | GND | Ground Pin |
| 8 | +5V | VCC Pin Power Positive |

***Table-3: ATtiny85 Programmer pin connections with Arduino UNO***



***Fig 2: ATtiny 85 – Arduino ISP Programmer***

Once the circuit connections have been made it is time to configure the Arduino IDE to load the sketch to the Arduino UNO.

The Arduino UNO can be downloaded from **https://www.arduino.cc/en/software**and the latest version of the IDE to be downloaded can be selected for either the Windows machine of the Mac. Clicking on the downloaded file will install the software on the computer and can be used to install the Arduino ISP sketch on the Arduino UNO. Once the Arduino IDE has been selected the ISP sketch can be selected by choosing **File>Examples>Arduino ISP**. This program is then burnt into the Arduino UNO. With these steps we now have the Arduino with the ISP program for programming the ATiny85 and the ATiny85 connected to the Arduino Uno for programming. It may be noted that the ATiny85 will be connected to the Arduino via a 8 PIN IC base to prevent a soldering and desoldering of the ATiny85 each time it has to be programmed.

***Note:***

***Using the Arduino IDE for programming the Arduino UNO may require installation of USB drivers details of which are mentioned in the Arduino website and in the forums. The procedure for this is different for the Mac, Windows and Linux operating systems.***

Now with the Arduino setup as a programmer it is time to ensure that the ATiny85 libraries are installed correctly. For this go to Files>Preferences and populate the Additional Board Mangers URL’s with: ***https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package\_damellis\_attiny\_index.json***

The now select ***Sketch>Include Libraries>Manage Libraries*** and in the search, box enter ***attiny*** to display the ***attiny*** core by David A Millis. Click on INSTALL to install the cores. After restarting the Arduino IDE, the ATtiny 25/45/85 will appear in the **Tools>Boards**. Select the ATiny 85 board (in this case microcontroller).

The IDE is now ready to program the ATtiny85.Make sure that the ***Tools>Programmer*** is set to Arduino as ISP. With the IDE set up and the ATtiny85 connected it is now time to start our first project which is called “My Magic Lamp” which is a Atiny85 based LED table lamp with a Bluetooth interface to control from your mobile phone. You can set the brightness of the lamp from your mobile phone and also program a time when the lamp will turn on like an alarm clock.

1. **My Magic Lamp Program and circuit on the ATiny 85**

The My Magic Lamp project is a Bluetooth interfaced LED table lamp which can be totally controlled by your mobile phone. The mobile phone will be used to switch on the lamp, switch off the lamp and schedule the lamp on. The project has a Arduino program running on the ATtiny85 and a mobile phone App developed using the App Inventor.

The App inventor is an online app that is developed by MIT and can be used using your Google account. The program can be used to develop a QR code link which can be scanned to install the App on your phone.

First the Arduino program can be discussed. The Mobile phone App will send a comma separated code via Bluetooth which is picked up by the HC-05 connected to the ATtiny85 and then the program decides on the action. The command is in the form

***Command, Parameter***.

When the particular button is pressed on the mobile phone, the phone sends a two-number command that are separated by a comma.

|  |  |  |
| --- | --- | --- |
| **Command Value** | **Parameter Value** | **Meaning** |
| 1 | 0 | Lamp ON |
| 2 | 0 | Lamp Off |
| 3 | XXX (0 to 255) | Set Lamp Brightness in PWM range of 0 to 255. |

***Table 4: Command set for My Magic Lamp control***

For example, if the Mobile phone sends a command, parameter value of 1,0 the Lamp will turn ON and if the value is 3,100 the lamp brightness is set at about 50 % which is a PWM value of 100 out of 255.

Bluetooth Communication

Mobile Phone

ATtiny 85

HC 06 Bluetooth

***Fig 3: Block Diagram of The My Magic Lamp Project***

In the above block diagram the ATtiny85 has a Bluetooth Interface through the HC -05 Module and is used to channel the Bluetooth signal from the mobile phone to the ATtiny85 which will control the lamp.Depending on the command,parameter value the lamp can be controlled.The ATtiny85 will read the Bluetooth command and then control the LED lamp and can set it PWM value from 0-255 where 0 is Off and 255 is fully bright.

The Arduino ATtiny85 program is as follows:

/\*

***MY MAGI LAMP By Santhosh Jayarajan***

***1. Uses ATTiny85 at 8 Mhz Internal***

***2.Uses a HC 06 TO communicate by Bluetooth to the Tiny 85***

***3.LED connected to Port 2***

***4 HC -06 TO Port - 1and 3***

\*/

#include <SoftSerial.h>

#include <TinyPinChange.h>

#include <EEPROM.h>

#define SUPLED 2

#define PWMOUT 0

const byte BTRX = 4;

const byte BTTX = 1;

***//VARIABLES***

double LEDCount;

int BTCommand;

int BTParameter;

byte LightBrightness;

byte LightBrightnessRaw;

SoftSerial BTRead(BTRX,BTTX);

void setup()

{

pinMode(SUPLED,OUTPUT);

pinMode(PWMOUT,OUTPUT);

pinMode(BTRX,INPUT);

pinMode(BTTX,OUTPUT);

BTRead.begin(9600);

EEPROM.write(10,255);

LightBrightnessRaw=EEPROM.read(10);

}

***//LIGHT ON***

void Light\_On()

{

LightBrightnessRaw=EEPROM.read(10);

}

***//LIGHT OFF***

void Light\_Off()

{

LightBrightnessRaw=0;

}

***//BT READ SUBROUTINE***

void BTDataRead()

{

if(BTRead.available()>1)

{

digitalWrite(SUPLED,HIGH);

delay(100);

digitalWrite(SUPLED,LOW);

BTCommand=BTRead.parseInt();

BTParameter=BTRead.parseInt();

}

if(BTCommand==1){Light\_On();}

if(BTCommand==2){Light\_Off();}

if(BTCommand==3)

{

LightBrightnessRaw=constrain(BTParameter,0,255);

EEPROM.write(10,LightBrightnessRaw);

if(LightBrightness!=EEPROM.read(10))

{

EEPROM.write(10,LightBrightness);

}

}

}

***// MAIN LOOP***

void loop()

{

BTDataRead();

**//Ramp up**

if(LightBrightnessRaw>LightBrightness)

{

while(LightBrightnessRaw>LightBrightness){LightBrightness=LightBrightness+1;delay(30);analogWrite(PWMOUT,LightBrightness);}

}

***//Ramp down***

if(LightBrightnessRaw<LightBrightness)

{

while(LightBrightnessRaw<LightBrightness){LightBrightness=LightBrightness-1;delay(30);analogWrite(PWMOUT,LightBrightness);}

}

***//Steady State Value***

if(LightBrightnessRaw==LightBrightness)

{

analogWrite(PWMOUT,LightBrightness);

}

}

The program starts with the definition of the variables where the LED light is connected to GPIO Pin 0. The light intensity can be changed by changing the value in the analogWrite command. The main loop constantly calls the ***BTDataRead()*** function which checks for the data from the mobile phone. When the data arrives the Command and Parameter values are separated using the

BTCommand=BTRead.parseInt();

BTParameter=BTRead.parseInt();

The command parseInt() will separate the two comma separated integers into the two variables. Depending on the command values of 1,2,3 the various sub routines are executed like so:

if(BTCommand==1){Light\_On();}

if(BTCommand==2){Light\_Off();}

if(BTCommand==3)

{

LightBrightnessRaw=constrain(BTParameter,0,255);

EEPROM.write(10,LightBrightnessRaw);

if(LightBrightness!=EEPROM.read(10))

{

EEPROM.write(10,LightBrightness);

The PWM value is constrained to a range of 0 to 255 and stored into the EEPROM address 10 so that on start up the brightness value is maintained. The EEPROM value is read and if it is not stored properly it is rewritten. The command from the mobile phone via Bluetooth are:

1= Turn LED light ON

2=Turn LED Light OFF

3=Set the LED Light brightness in PWM range of 0-255.

The LED light is ramped up or down depending on if the set value is below or above the existing value by the code below:

**//Ramp up**

if(LightBrightnessRaw>LightBrightness)

{

while(LightBrightnessRaw>LightBrightness){LightBrightness=LightBrightness+1;delay(30);analogWrite(PWMOUT,LightBrightness);}

}

***//Ramp down***

if(LightBrightnessRaw<LightBrightness)

{

while(LightBrightnessRaw<LightBrightness){LightBrightness=LightBrightness-1;delay(30);analogWrite(PWMOUT,LightBrightness);}

}

***//Steady State Value***

if(LightBrightnessRaw==LightBrightness)

{

analogWrite(PWMOUT,LightBrightness);

}

At the start of the program the LightBrightnessRaw value is read from EEPROM address 10 which is where the brightness value is stored.When there is a request for the brightness change the

LightBrightnessRaw value gets updated by the read value with the line LightBrightnessRaw = constrain(BTParameter,0,255);ensuring that the value is between 0 and 255.

The Ramp Up and Ramp Down sub routines has a 30 Msec delay so that the LED light intensity will slowly increase or decrease until the LightBrightnessRaw value is equal to the LightBrightness value.

When the two values are equal the PWM function will send the brightness value to the LED by the analogWrite() function like so:

***//Steady State Value***

if(LightBrightnessRaw==LightBrightness)

{

analogWrite(PWMOUT,LightBrightness);

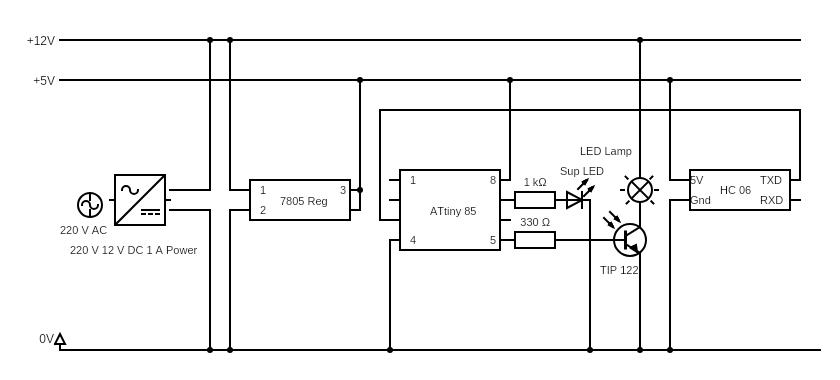
}

}

The analogWrite() function takes two parameter the first is the pin to write the PWM value and the next parameter the PWM value in a range 0-255 where 0 is OFF and 255 is full output.

As is evident from the above code the program in the ATtiny85 is very small and efficient and runs continously and when it receives a Bluetooth signal from the mobile phone inteprets the command and performs the required opeartion.

The circuit diagram for the project is as follows:



***Fig:4 My Magic Lamp Circuit Diagram***

The 220 V power is fed to a 220 V 12 V 1 A Power supply which is used for the LED supply.The 7805 IC converts the 12 V to 5 V which is used for the ATtiny 85 and HC -06 power supply.

The PWM is connected to GPIO 0 which is used to control the brightness of the LED lamp.The Supervisory LED is connected to GPIO 2 or ATtiny 85 pin 7.Theis LED is used to indicate when the Bluetooth Module HC-06 receives data from the mobile phone.

The Bluetooth module HC-06 has 4 pins of which 2 are for Power +5 V and GND and the other two are for receive and transmit.The receive pin RXD is not used.The TIP 122 is a NPN transistor that can handle about 1 A of collector current to power up the LED light.When the HC-06 is powered up the on board LED will blink and will become steady once the module has been paired with the mobile phone.It would be helpful to put a heatsink for TIP 122.

The components for the circuit is as below:

|  |  |  |
| --- | --- | --- |
| **Sl.No** | **Component** | **Specifications** |
| 1. | 12 V power Supply | 12 V ,1 A Power Supply |
| 2. | Microcontoller | ATTiny 85,8 Pin Microcontroller |
| 3. | Regulator IC | 7805 ,5 V Regulator IC  (Pin -1 Input,Pin 2 Common Ground,Pin-3 – 5 V Output |
| 4. | LED | Red LED for Bluetooth activity indication |
| 5. | Resistor- 1K | ¼ W Resistor for LED Current Limiting |
| 6. | Resistor 330 Ohms | ¼ W Resistor for Transistor TP 122 base drive |
| 7. | HC-06 | Bluetooth Module for communication with Mobile phone.  (4 Pin) |
| 8. | LED Lamp | 12 V LED Lamp,3 W with heat sink. |

***Table 5: Componenrs list for My Magic Lamp Project.***

This compltes the circuit and ATtiny 85 program for the project and now brings us to th emost intersting part of the project which is the App that will rin on the mobile Phone.

1. **App Inventor for Mobile interface**

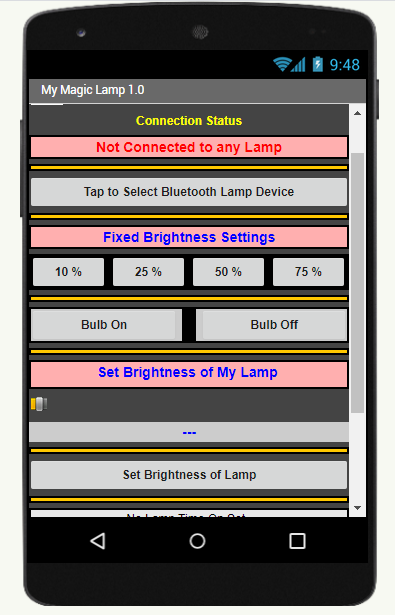
Developing the App for the mobile phone can be done in many ways,but the easiest and best way is to use the App Inventor from MIT and can be accesses using the Google or the Facebook account.Please note that this is not the only platform to make this design but is certainly the easiest and most convenient to use.

The link to App Inventor is: ***http://ai2.appinventor.mit.edu/***

This book is not a guide on how to use App Inventor.We will be going through the various block that are required for the program and an explainantion of the same.To start let us look at the requirements of our App.

* Display the available Bluetooth devices for pairing.
* Indicate when the connection to our magic lamp has been established.
* Provide buttons for ON/OFF.
* Provide the presets for 25%,50%,75% and 100% brightness settings.
* Provide a slider for setting brightness at random.
* Provide a time for switch on setting.
* Provide a button to exit.

This is how the mobile user interface looks like:



***Brightness Control Wiper***

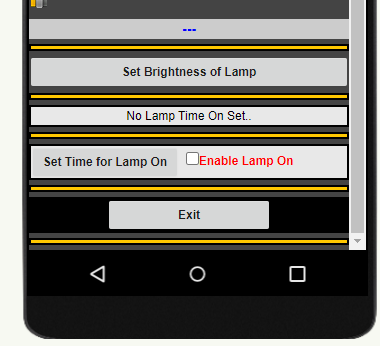
***Connection Establish Switch***

***Lamp ON/OFF Switch***

***Brightness Pre-set Switches***

***Connection Status***

***Fig 5: Mobile App for My Magic Lamp Project(Top)***



**Exit Button**

**Timer ON Set Button**

**Timer ON status Button**

***Fig 6: Mobile App for My Magic Lamp Project(Bottom)***

The mobile App will first look for all the open Bluetooth connections and will list the available connections.Our project uses the HC-06 and thus this will show up on the list which can be selected to establish the bluetooth connection.Once the connection is established the App can communicate with the lamp.

For control like ON/OFF and setting the brightness that App uses a two data comma sepeated format like below:

***Command,Parameter***

The command can be 1,2, or 3 and Parameter can be from 0-255.Our Magic Lamp program that uses the ATtiny85 will deciper the command,parameter pair and perform the required action.

The table below shows the command,parameter set:

|  |  |  |
| --- | --- | --- |
| **Command Value** | **Parameter Value** | **Meaning/Action** |
| 1 | 0 | Lamp ON |
| 2 | 0 | Lamp Off |
| 3 | XXX (0 to 255) | Set Lamp Brightness in PWM range of 0 to 255. |

***Table 6: Commad,Parameter List for My Magic Lamp Controls***

For example to Switch on the lamp the App will send a Bluetooth command of 1,0 and to switch it off the pair will be 2,0.To set the brightness of 50% the pair will be 3,127.(255 is the PWM value to full brightness ,so 50 % will be 127).

With this general process for controlling the lamp by Bluetooth let us now look at the various blocks in AppInventor.Please note that this is not a book on the use of App Inventor but only on its use.Please note that with the general principle the project can be implemented in any other Android development IDE.Before we start the expalination of the blocks below is atable outlining the values to be sent for 255,50 % ,75% and 1005 brightness levels.

|  |  |  |
| --- | --- | --- |
| **Set Value(%)** | **Parameter Value** | **Meaning/Action** |
| 25 | 64 | 25 % of a maximum of 255 |
| 50 | 128 | 50 % of a maximum of 255 |
| 75 | 191 | 75 % of a maximum of 255 |
| 100 | 255 | 100 % of a maximum of 255 |

***Table 7: Commad,Parameter List for My Magic Lamp Brightness***

Now let us look athe the various blocks that are programmed in App Inventor:

1. **Variable Declaration**

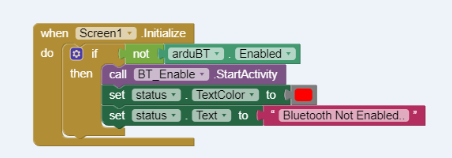


***Fig 7: Mobile App for My Magic Lamp Project Variable Initilisation***

These blocks are used to set various variables that are used in the program to some initial value.

* Alarm Enables is set to false and will only be set to true when the alarm is switched on in the program.
* Brightness\_2 is a internal vraiable used to calculate the brightness.
* BTReceive is the Bluetooth Receive buffer when is set to Null.
* Time\_Match is a varaible that is used in the program if the set time and the actual time have matched to set the lamp on.
* Hour\_Set,Minute\_Set are varaibles to store the alarm Hour and minute values.
* Time\_Now is a varaible used to get the present time.

1. **Screen Initilize**



***Fig 8: Mobile App for My Magic Lamp Project Screen Initilization***

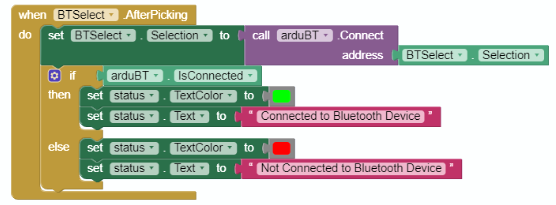
In this block the code is run after the screen initises and if the Bluetooth object is not enabled it starts the activity of listing the active devices and sets the text colour and sets the stataus text to Bluetooth Not Enabled.After this step the list of available Bluetooth devices will be shown and the user can select the bluetooth device pertaining to the Magic lamp Project.

**3) Before Picking**



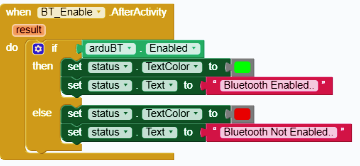
In this block the Select box elements are updated with the address and names of the available Bluetooth devices. This option is needed so that the user can then select the BT device that is connected to the Magic Lamp circuitry.

**4) After Picking**



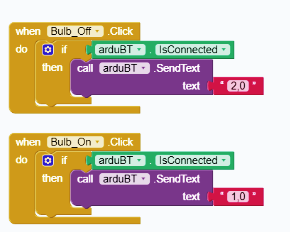
This block is executed after the Bluetooth device selection has been made. The selected BT device is then used to make a connection and if the connection is successful the status text colour changes to green and the text appears in the status box announcing the successful connection. If the connection is not successful then the text colour is changed to red and the status text is changed to announce a failure in the connection. If the connection is successful then the App is ready for the command to be sent to the Magic Lamp circuit.

**5) Bluetooth Enabled**



This block is to indicate if the Bluetooth enabling has been successful. The BT\_Enable AfterActivity function checks the result and if true it will set the text colour to green and display that Bluetooth is Enabled and will display in red if the Bluetooth is not enabled. This is a necessary step to ensure that the phone will enable the Bluetooth option. This enabling is very necessary for the other control s of the lamp. Now with the Bluetooth in the phone enabled and the Magic Lamp Bluetooth device selected it prepares the system for all controls from the mobile app.

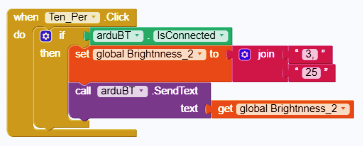
**6) Lamp OFF/ON**



Now with the Bluetooth interface all connected and running the lamp is ready to receive the commands. The command format is a comma separated text with the first parameter as the command and the next as an optional parameter.

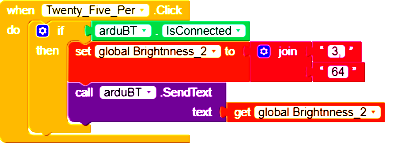
The Lamp ON/OFF command do not have a parameter and hence are set to zero but command parameter is 2 for OFF and 1 for ON. This block also ensures that the Bluetooth devices are connected with the ***IsConnected*** parameter before issuing the command.

**7) Set Brightness to 10%**



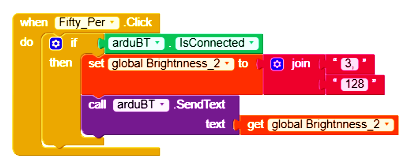
This block is used to send the command for 10% brightness level. For setting the brightness in the lamp to two variables are used. The command parameter of 3 means that the brightness is being send and the next variables will send the value of the brightness. The PWM (Pulse Width Modulation) values range from 0-255 and hence 10 % brightness corresponds to a PWM value of 25.A text variable called ***Brightness\_2*** is used to join the two variables with a comma using the join command. Then a call to ***arduBT.SendText*** commandwill send the command via the Bluetooth interface.

**7) Set Brightness to 25 %**



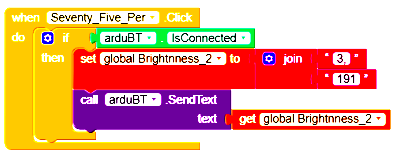
This block is used to send the command for 25% brightness level. For setting the brightness in the lamp to two variables are used. The command parameter of 3 means that the brightness is being send and the next variables will send the value of the brightness. The PWM (Pulse Width Modulation) values range from 0-255 and hence 25 % brightness corresponds to a PWM value of 255 x0.25 = 64.A text variable called ***Brightness\_2*** is used to join the two variables with a comma using the join command. Then a call to ***arduBT.SendText*** commandwill send the command via the Bluetooth interface.

**8) Set Brightness at 50%**



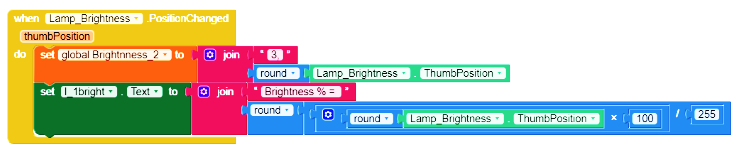
This block is used to send the command for 50% brightness level. For setting the brightness in the lamp to two variables are used. The command parameter of 3 means that the brightness is being send and the next variables will send the value of the brightness. The PWM (Pulse Width Modulation) values range from 0-255 and hence 50 % brightness corresponds to a PWM value of 255 x 0.5 = 128.A text variable called ***Brightness\_2*** is used to join the two variables with a comma using the join command. Then a call to ***arduBT.SendText*** commandwill send the command via the Bluetooth interface.

**9) Set Brightness to 75 %**



This block is used to send the command for 75% brightness level. For setting the brightness in the lamp to two variables are used. The command parameter of 3 means that the brightness is being send and the next variables will send the value of the brightness. The PWM (Pulse Width Modulation) values range from 0-255 and hence 75 % brightness corresponds to a PWM value of 255 x 0.75 = 191.A text variable called ***Brightness\_2*** is used to join the two variables with a comma using the join command. Then a call to ***arduBT.SendText*** commandwill send the command via the Bluetooth interface.

**10) Setting the Brightness level from Slider**

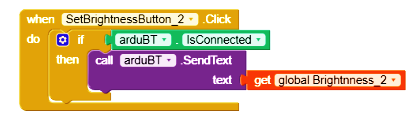


In this block the slider position from 0-100% is taken and converted to a PWM value using the formula:

***Brightness\_2 = 3, Slider Position (0-100%) x100/255***

The value is then rounded off set to the variable ***Brightness\_2*** along with 3 joined to it. This value is then joined with the command of 3 for set brightness level and waits for the value to be sent to the lamp by clicking on a Set Brightness Button which is explained in the next block. In summary to set the lamp brightness use the slider to set the brightness in range 0-100% and click the set Brightness button to send the command to the lamp. The PWM value is calculated by this block.

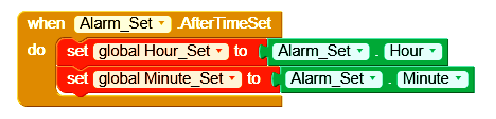
**11) Set Brightness**



After the Brightness has been set using the slider it has to be sent to the lamp. This is done by the set brightness button.

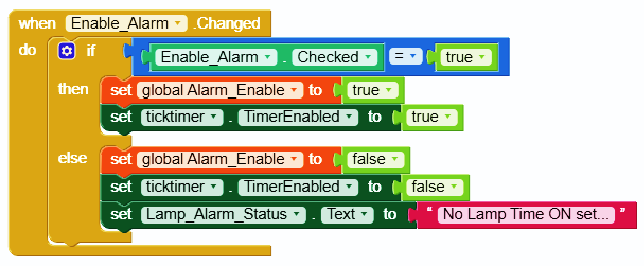
This block accepts the button press and sends the stored value of ***Brightness\_2*** to the device using the command 3. Note that the ***Brightness\_2*** value would be set from the slider value in the earlier block. Then a call to ***arduBT.SendText*** commandwill send the command via the Bluetooth interface.

**12) Setting the Alarm**



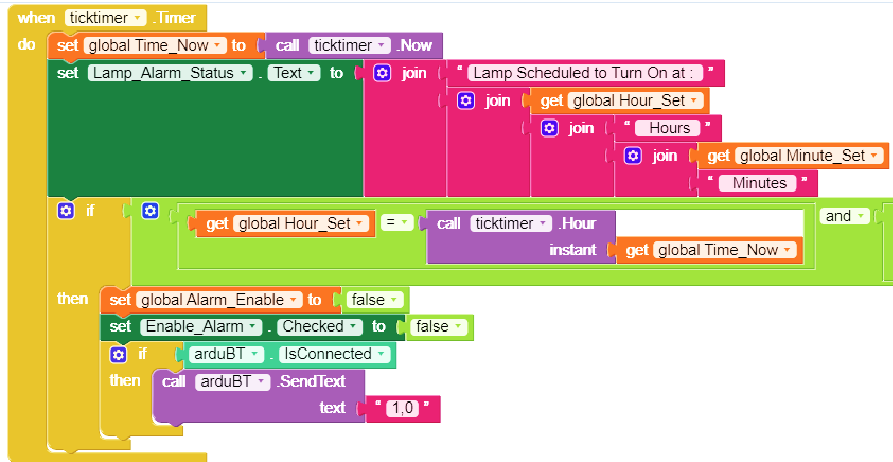
The My Magic Lamp project also has a alarm option where the time for the lamp switch on can be scheduled. Thus, by setting the Hour and minute the schedule can be set and when this time is reached the App will send a Lamp switch on command. The global variables ***Hour\_Set*** and ***Minute\_Set*** are set with the hour and minute values of the function. These variables will then be compared every second with the present time and when the hour and minute match the Lamp ON command is send.

**13) Alarm Enable**



In this block when the Enable Alarm is clicked it sets the global variable ***Alarm\_Enable*** to TRUE and also starts a Timer Control named ***ticktimer*** which is a block that will be run ever second. Thus, enabling the scheduling option is a two step process the first being setting the hour and minute for the lamp to turn on and secondly to enable the alarm option which will enable the 1 second tick timer which runs every second. When the ***ticktimer*** block runs it checks for the hour and minute match to send the lamp ON command.

**14) Tick Timer Block**



**A**



**A**

***Note: This block is a very long one and hence has been made into two pictures that are joined at the point A.***

This block will run every second and will load the present time in hours and minutes into two variables for hour and minute and compare the two. When the two are equal meaning that the scheduled time has been reached the App will send a Lamp ON command.

Recall that 1,0 is the command for Lamp ON.

**15) Exit Application**



This block when executed will close the application.

**v) Conclusion of Part-1**

This covers all the blocks that are used by the application to provide full control of ***My Magic Lamp*** using Bluetooth. The multiple modules are used to open the Bluetooth connection, display the status and provide functions to set the brightness as fixed values like 10 %,50 % etc and also as a slider for values in between. The timer scheduler uses a Tick timer which runs every second to check the scheduled time and switch on the lamp at the scheduled timer. This project is a very useful project that demonstrates the power of the ATtiny 85 for small yet powerful applications.

This brings us to the end of part-1 of this book and we now move on to a more powerful microcontroller platform the Arduino Uno and its interface with the most popular programming language Python.

Software and Hardware will be meeting in the next project with Arduino Uno, Python and libraries like xlwrt, Matplotlib and PyFirmata.

**vi) Resources for Part-1**

**1. Components**

All components can be purchased from:

[www.amazon.in](http://www.amazon.in)

[www.in.element14.com](http://www.in.element14.com)

**2. Circuit Diagram/Program listing**

Git Hub:

<https://github.com/SJayarajan/Embedded-Systems-Development-from-the-Ground-Up.git>

(This Git Hub link provides access to the circuit diagrams, ATTiny 85 program and the Application APK file.

**3. Further Readings**

3.1 ATtiny Data Sheet:

<https://github.com/SJayarajan/Embedded-Systems-Development-from-the-Ground-Up.git>

3.2 DigiSpark – DigiStump – There is a product out there that has the ATtiny 85 with a USB chip to make programming easier. Just plug in the device in the USB port and load the drivers and start programming. There are various resources on the internet that deal withthis device. **More details at:** **http://digistump.com/products/1**

**4. Other Project Ideas**

* Bluetooth Controller LED Ceiling Lamp
* Water irrigation system for garden plants.
* PWM based DC drill speed control.
* NIMH battery charger.
* Water level controller



**“I think it is important to find the little things in everyday life that make you happy”**

***- Paula Cole***

**Part-2**

**Arduino UNO meets Python**

**i)**  **Introduction and basic details**

We had made good progress in Part-1 and in that section, you were introduced to a small but powerful microcontroller that though good for small projects will soon run out of space and power for larger projects. In this section we will program a larger more powerful microcontroller system that has been around for some time, the Arduino UNO. This microcontroller comes with its own IDE (Integrated Development Environment) and hence programming and creating projects is a breeze. But what will be power of the UNO be without some software support? This is where Python comes in and, in this section, we will create a **6 Channel Data Logger** that will display graphs in real time and also store data in an Excel file for further analysis. This is not a book on ***Python, Matplotlib or Firmata*** and hence the basic programming in Python and these libraries will not be covered here. There are many books and resources to fill that gap.

ii) **The Arduino Uno**

The Arduino Uno is a mid-level microcontroller system that comes with a reasonably good program memory space, enough of RAM and EEPROM for a mid – level project.

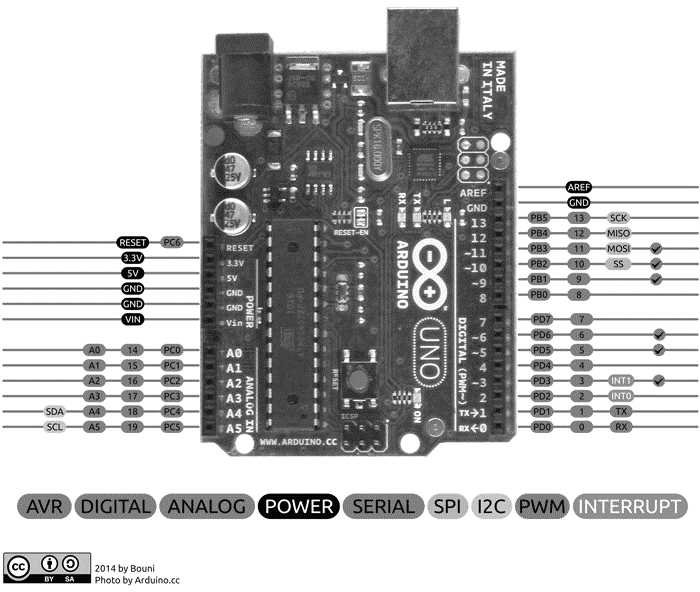
The advantage of the Uno is that it comes with a USB programming facility and a powerful IDE that can be freely down loaded from the Arduino website. The table below shows the main features of the Arduino Uno.

|  |  |  |
| --- | --- | --- |
| Sl.No | Parameter | Value |
| 1. | CPU Core | ATMega 328P |
| 2. | Program Memory Size | 32 KB (0.5 KB Used for bootloader. |
| 3. | Input Outputs | 14(0-13) |
| 4. | Operating Voltage | 2.7 to 5.5 V |
| 5. | RAM Size | 2 KB |
| 6. | Max Clock | 16 Mhz |
| 7. | PWM Pins | 5(Pin3,5,6,9,11) |
| 8. | Analog Inputs | 6(AN0-AN5) |

***Table-1: Arduino Uno Basic details***

For the Datalogger program we will be using the 6 Analog Inputs which will be scaled and displayed on graph and stored in Excel as CSV (Comma Separated Value) file.

The figure and pin assignment of the Arduino UNO is given below:



***Fig-1: Arduino Uno Pin Assignment***

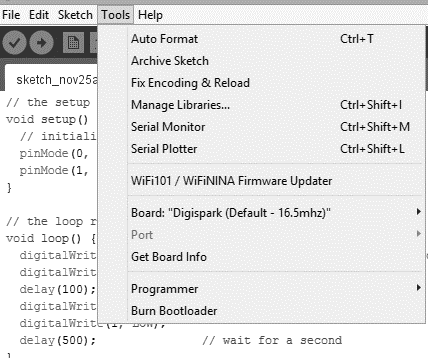
***(Fig Source:www.component101.com)***

At the top of the board on the left is the power input jack that can be connected to power up to 12 V safely. The box like connector on the top right is for the USB cable that is used for the connection of the USB cable to the IDE for programming and data despatch. This port is very useful in this project because the data is sent via this USB port to the Python program.

The Python program will read the data sent by the Arduino into a USB port and display it on the GUI (Graphical User Interface) based Python program, send the data to Matplotlib for graph display and save the CSV file. The Python program thus is the interface between the Arduino and the PC or laptop. The power for the Arduino comes from the USB port and hence in this project no external power is required.

The IDE (Integrated Development Environment) used for this project can be downloaded from [***www.arduino.cc.*The**](http://www.arduino.cc.The) download is free (unless you want to make a donation!). Navigate to the software page and download the latest IDE which is version 1.8.13.

After the software has be installed the correct Board and Port must be selected. An easy way to check if the USB drivers have been installed is to see the Port option enabled in the IDE with the Port Number (USB port) displayed.



Port number will be displayed here if the drivers are correctly installed.

***Fig-2: Arduino Uno Port Assignment***

**iii) The Standard Firmata Library**

The heart of the project is the Arduino Uno and the library that it runs. This a great method to talk to the Arduino Uno through Python. The library does not need any modifications and can be used as such and has got a rich collection of methods that can be used to communicate with the Arduino and read and write the analog and digital values. For the project in this part, we will be using the ***read analog function*** which will interface the Arduino Uno and the Python program. All details for this library can be got from the link below:

***https://roboticsbackend.com/arduino-standard-firmata-tutorial/***

Loading the Standard Firmata program into the Uno is fairly simple. From the Arduino IDE select ***Files>Examples> Firmata> Standard Firmata***. This file can be downloaded with a small change. The communication with the Python Program the speed for the USB serial is normally set to 9600 which is a good speed for our purpose. The default is normally set at the highest(57600) which can be changed by changing the following line in setup().

**Firmata.begin(57600); to Firmata.begin(9600);**

The file can then be downloaded to the UNO and the Arduino is ready for interfacing with the Python program that provided GUI, serial communication and graph plotting.

**Iv) Python and Pyfirmata Library**

This is not a book on Python programming and hence there will be little information here on use of Python and IDE’s used. To prepare us for the project in the next section a good Python IDE can be used. My personal favourite is ***Pycharm*** which is available as a free downloadable Community version which is ideal for Python programming.

The ***PyCharm IDE*** is available from: [**https://www.jetbrains.com/pycharm/**](https://www.jetbrains.com/pycharm/)**.**

After downloading and installation the following libraries have to be included in preparation for the project in the next section:

import xlwt  
import matplotlib.pyplot as plt  
import datetime  
from tkinter import \*  
from tkinter import messagebox  
from pyfirmata import Arduino,util

***Fig 3: Python Library Imports***

* The first import is for Excel write which is the interface between Python and Excel.
* The second import is for MatPlot library which is used for the graph plotting.
* The third import is used to name the file for Excel data saving.
* The fourth import is the Tkinter GUI library to provide the Graphical User Interface.
* The Fifth import will provide message box interfacing with the Python program to provide information to the user.
* The sixth import will provide connection to the Arduino Firmata library for data exchange.

With all these imports we are now ready with the Arduino and Python to build a 6 Channel Data logger with Excel saving and graphing using MatPlotlib Library.

**v) 6 Channel Datalogger project with Matplotlib and Excel.**

The Arduino Uno in this project serves as a conduit to transfer the analog values to the PC which is then read by the Python program to display, plot and store the values in the CSV (Comma Separated values). The CSV format is a very easy to use format that can be read by Excel, Panda to name a few.

The block diagram of the project is as given below. The Analog inputs from the outside world are connected to the 6 channels of the Arduino Uno in a range of 0-5 V which corresponds to a A/D value of 0-1023.But in the Firmata Library the range is from 0.00 to 1.00 which can then be scaled with the program by entering the slope and offset values. The output that is scaled with be in the form:

**Y=mx + c** where Y is the scaled output, m is the slope and c is the offset.

**Data Storage (CSV), Excel, Graphing**

**Arduino UNO with Firmata**

**PC Running Python** Program

**Analog Inputs**

***Fig 3: Block Diagram for 6 Channel Data Logger.***

The role of the Arduino Uno in this project is that of a server that will serve up the data that is requested by Python program. The Arduino Uno does not have any control features and can be considered to be intermediate between the outside world of voltages and values that can be scaled, saved and plotted.

The Python program shown below runs on the PC and uses the various libraries to receive the data from the Arduino, plot and save the data in the CSV format. The Tkinter library is used to provide the GUI(Graphical user interface) to the program.

Explanations of each of the project blocks are in **(*ITALICS UPPERCASE)***.

##\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

***#ARDUINO TO EXCEL PYTHON INTERFACE DESIGNED AND PROGRAMMED BY:***

***#SANTHOSH JAYARAJAN.***

***#PROGRMING START ON 27/2/2020.***

***#NO EARLIER REVISIONS.***

***#NOTES:***

***#TESTED ON 10-3-2021***

***#***

***#***

***#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****

***#(IMPORTING OF ALL THE REQUIRED LIBRARIES)***

import xlwt

import matplotlib.pyplot as plt

import datetime

from tkinter import \*

from tkinter import messagebox

from pyfirmata import Arduino,util

***#(DATA VARIABLE INITIALISATION)***

inc=0

X1data=[]

Y1data=[]

X2data=[]

Y2data=[]

X3data=[]

Y3data=[]

X4data=[]

Y4data=[]

X5data=[]

Y5data=[]

X6data=[]

Y6data=[]

***#(OPENING MATPLOT LIBRARY)***

plt.ion()

plt.show()

***#(OPENING EXCEL WORKSHEET)***

Current\_Time = datetime.datetime.now()

Hr = Current\_Time.hour

Min = Current\_Time.minute

Sec = Current\_Time.second

File\_Name\_Excel = "Arduino\_Excel\_Interface " + str(Hr) + "\_" + str(Min) + "\_" + str(Sec)+".xls"

Excel\_Workbook = xlwt.Workbook()

Excel\_WorkSheet = Excel\_Workbook.add\_sheet('Data at '+str(Hr)+'\_'+str(Min)+'\_'+str(Sec))

***#(SETTING OF FONT0 FOR EXCEL)***

Font0=xlwt.Font()

Font0.name="Calibri"

Font0.bold=True

Font0.height=600

#SETTING FONT1

Font1=xlwt.Font()

Font1.name="Calibri"

Font1.bold=True

Font1.height=250

***#(SETTING OF FONT2 FOR EXCEL)***

Font2=xlwt.Font()

Font2.name="Calibri"

Font2.height=220

***#(SETTING STYLES FOR EXCEL)***

Style0=xlwt.XFStyle()

pattern = xlwt.Pattern()

pattern.pattern = xlwt.Pattern.SOLID\_PATTERN

pattern.pattern\_fore\_colour = xlwt.Style.colour\_map['light\_green']

Style0.pattern = pattern

Style0.font=Font0

Style1=xlwt.XFStyle()

Style1.font=Font1

Style2=xlwt.XFStyle()

Style2.font=Font2

Excel\_WorkSheet.write\_merge(0,0,0,6,"Arduino Excel Interface Data",Style0)

Excel\_WorkSheet.write(2,0,"Channel Name>>",Style1)

Excel\_WorkSheet.write(3,0,"Sl.No",Style1)

Excel\_WorkSheet.write(3,1,"Date",Style1)

Excel\_WorkSheet.write(3,2,"Time",Style1)

Excel\_WorkSheet.write(3,3,"Channel-1",Style1)

Excel\_WorkSheet.write(3,4,"Channel-2",Style1)

Excel\_WorkSheet.write(3,5,"Channel-3",Style1)

Excel\_WorkSheet.write(3,6,"Channel-4",Style1)

Excel\_WorkSheet.write(3,7,"Channel-5",Style1)

Excel\_WorkSheet.write(3,8,"Channel-6",Style1)

***# (CREATING A TKINTER WINDOW)***

master = Tk()

reading=0

master.title('Arduino Based Excel Interface Ver 1.0')

***#(ABOUT THE PROGRAM)***

def About\_prog(event):

messagebox.showinfo("About the Arduino Excel Interface","This Program was developed by Santhosh Jayarajan in March 2020 Version 1.0")

***#(QUIT THE PROGRAM)***

def Quit\_prog(event):

master.destroy()

plt.close()

Excel\_Workbook.save(File\_Name\_Excel)

***#(START READING SUB ROUTINE - FROM ARDUINO)***

def Start\_Read(event):

global reading,ch1\_scale\_val1,ch1\_scale\_val2,ch1\_scale\_val3

global ch2\_scale\_val1, ch2\_scale\_val2, ch2\_scale\_val3

global ch3\_scale\_val1, ch3\_scale\_val2, ch3\_scale\_val3

global ch4\_scale\_val1, ch4\_scale\_val2, ch4\_scale\_val3

global ch5\_scale\_val1, ch5\_scale\_val2, ch5\_scale\_val3

global ch6\_scale\_val1, ch6\_scale\_val2, ch6\_scale\_val3

if reading==0:

reading=1

start\_read\_but.config(text='Stop Reading')

stat\_ind.config(text="Started Reading from Arduino...",background='seagreen1',foreground='blue4')

if (a1\_sp\_off.get() == "" or a2\_sp\_off.get() == "" or a3\_sp\_off.get() == "" or a4\_sp\_off.get() == "" or a5\_sp\_off.get() == "" or a6\_sp\_off.get() == ""):

reading=0

start\_read\_but.config(text='Start Reading')

stat\_ind.config(text="Enter all Channel Details--Data Entry Error...", background='red', foreground='yellow')

messagebox.showinfo("Channel Data Entry Incomplete","Enter values for all channels as Span,Offset and Channel Name as Comma Seperated Values...")

return

else:

reading=0

start\_read\_but.config(text='Start Reading')

stat\_ind.config(text="Stopped Reading from Arduino...", background='red', foreground='yellow')

plt.close()

***#(EXTRACTION OF CHANNEL ENTRY VALUES SPAN,OFFSET AND CHANNEL NAME)***

ch1\_scale = a1\_sp\_off.get().split(",")

ch2\_scale = a2\_sp\_off.get().split(",")

ch3\_scale = a3\_sp\_off.get().split(",")

ch4\_scale = a4\_sp\_off.get().split(",")

ch5\_scale = a5\_sp\_off.get().split(",")

ch6\_scale = a6\_sp\_off.get().split(",")

***#(CHANNEL -1 DETAILS)***

ch1\_scale\_val1 = float(ch1\_scale[0])

ch1\_scale\_val2 = float(ch1\_scale[1])

ch1\_scale\_val3 = ch1\_scale[2]

***# (CHANNEL -2 DETAILS)***

ch2\_scale\_val1 = float(ch2\_scale[0])

ch2\_scale\_val2 = float(ch2\_scale[1])

ch2\_scale\_val3 = ch2\_scale[2]

***#(CHANNEL -3 DETAILS)***

ch3\_scale\_val1 = float(ch3\_scale[0])

ch3\_scale\_val2 = float(ch3\_scale[1])

ch3\_scale\_val3 = ch3\_scale[2]

***# (CHANNEL -4 DETAILS)***

ch4\_scale\_val1 = float(ch4\_scale[0])

ch4\_scale\_val2 = float(ch4\_scale[1])

ch4\_scale\_val3 = ch4\_scale[2]

***# (CHANNEL -5 DETAILS)***

ch5\_scale\_val1 = float(ch5\_scale[0])

ch5\_scale\_val2 = float(ch5\_scale[1])

ch5\_scale\_val3 = ch5\_scale[2]

***#(CHANNEL -6 DETAILS)***

ch6\_scale\_val1 = float(ch6\_scale[0])

ch6\_scale\_val2 = float(ch6\_scale[1])

ch6\_scale\_val3 = ch6\_scale[2]

***#(MATPLOT LIBRARY VALUES)***

Read\_Analog()

***#(READ ANALOG VALUES FUNCTION)***

def Read\_Analog():

global ch1\_scale\_val1, ch1\_scale\_val2,ch1\_scale\_val3

global ch2\_scale\_val1, ch2\_scale\_val2, ch2\_scale\_val3

global ch3\_scale\_val1, ch3\_scale\_val2, ch3\_scale\_val3

global ch4\_scale\_val1, ch4\_scale\_val2, ch4\_scale\_val3

global ch5\_scale\_val1, ch5\_scale\_val2, ch5\_scale\_val3

global ch6\_scale\_val1, ch6\_scale\_val2, ch6\_scale\_val3,reading

global X1data

global Y1data

global X2data

global Y2data

global X3data

global Y3data

global X4data

global Y4data

global X5data

global Y5data

global X6data

global Y6data

global inc

if reading==1:

***#(READING THE ANALOG CHANNELS)***

a1\_raw.config(text=str(ach1.read()))

a2\_raw.config(text=str(ach2.read()))

a3\_raw.config(text=str(ach3.read()))

a4\_raw.config(text=str(ach4.read()))

a5\_raw.config(text=str(ach5.read()))

a6\_raw.config(text=str(ach6.read()))

***# (DISPLAYING THE SCALED ANALOG CHANNELS)***

a1\_sc\_dis.config(text=str(round(float(ach1.read()) \* ch1\_scale\_val1 + ch1\_scale\_val2,4)))

a2\_sc\_dis.config(text=str(round(float(ach2.read()) \* ch2\_scale\_val1 + ch2\_scale\_val2,4)))

a3\_sc\_dis.config(text=str(round(float(ach3.read()) \* ch3\_scale\_val1 + ch3\_scale\_val2,4)))

a4\_sc\_dis.config(text=str(round(float(ach4.read()) \* ch4\_scale\_val1 + ch4\_scale\_val2,4)))

a5\_sc\_dis.config(text=str(round(float(ach5.read()) \* ch5\_scale\_val1 + ch5\_scale\_val2,4)))

a6\_sc\_dis.config(text=str(round(float(ach6.read()) \* ch6\_scale\_val1 + ch6\_scale\_val2,4)))

***#(DISPLAYING THE CHANNEL NAME)***

a1\_scaled.config(text=ch1\_scale\_val3)

a2\_scaled.config(text=ch2\_scale\_val3)

a3\_scaled.config(text=ch3\_scale\_val3)

a4\_scaled.config(text=ch4\_scale\_val3)

a5\_scaled.config(text=ch5\_scale\_val3)

a6\_scaled.config(text=ch6\_scale\_val3)

excel\_file.config(text="Storing Excel Record No: "+str(inc) + " in File>>")

***#(MATPLOTLIB VALUES)***

Current\_Time=datetime.datetime.now()

Hr=Current\_Time.hour

Min=Current\_Time.minute

Sec=Current\_Time.second

Day=Current\_Time.day

Mon=Current\_Time.month

Yer=Current\_Time.year

Date\_Now=str(Day) +"/"+str(Mon)+"/"+str(Yer)

Time\_Now=str(Hr) + ":" + str(Min) + ":" + str(Sec)

inc=inc+1

Y1data.append(round(float(ach1.read()) \* ch1\_scale\_val1 + ch1\_scale\_val2,4))

Y2data.append(round(float(ach2.read()) \* ch2\_scale\_val1 + ch2\_scale\_val2, 4))

Y3data.append(round(float(ach3.read()) \* ch3\_scale\_val1 + ch3\_scale\_val2, 4))

Y4data.append(round(float(ach4.read()) \* ch4\_scale\_val1 + ch4\_scale\_val2, 4))

Y5data.append(round(float(ach5.read()) \* ch5\_scale\_val1 + ch5\_scale\_val2, 4))

Y6data.append(round(float(ach6.read()) \* ch6\_scale\_val1 + ch6\_scale\_val2, 4))

X1data.append(inc)

plt.plot(X1data,Y1data,marker='o',linestyle='solid',linewidth=2,color='red')

plt.plot(X1data, Y2data, marker='o', linestyle='solid', linewidth=2, color='green')

plt.plot(X1data, Y3data, marker='o', linestyle='solid', linewidth=2, color='blue')

plt.plot(X1data, Y4data, marker='o', linestyle='solid', linewidth=2, color='black')

plt.plot(X1data, Y5data, marker='o', linestyle='solid', linewidth=2, color='cyan')

plt.plot(X1data, Y6data, marker='o', linestyle='solid', linewidth=2, color='yellow')

plt.legend([ch1\_scale\_val3,ch2\_scale\_val3,ch3\_scale\_val3,ch4\_scale\_val3,ch5\_scale\_val3,ch6\_scale\_val3 ], loc="lower right")

plt.grid(True)

plt.title('Arduino Analog Data at :'+ Time\_Now)

plt.xlabel('Record No:', fontsize=14)

plt.ylabel('Analog Values', fontsize=14)

plt.draw()

plt.pause(0.05)

***#(EXCEL VALUES LOADING)***

Excel\_WorkSheet.write(inc + 3, 0, inc, Style2)

Excel\_WorkSheet.write(inc + 3, 1, Date\_Now, Style2)

Excel\_WorkSheet.write(inc + 3, 2, Time\_Now, Style2)

Excel\_WorkSheet.write(inc+3, 3, round(float(ach1.read()) \* ch1\_scale\_val1 + ch1\_scale\_val2,4), Style2)

Excel\_WorkSheet.write(inc+3, 4, round(float(ach2.read()) \* ch2\_scale\_val1 + ch2\_scale\_val2,4), Style2)

Excel\_WorkSheet.write(inc+3, 5, round(float(ach3.read()) \* ch3\_scale\_val1 + ch3\_scale\_val2,4), Style2)

Excel\_WorkSheet.write(inc+3, 6, round(float(ach4.read()) \* ch4\_scale\_val1 + ch4\_scale\_val2,4), Style2)

Excel\_WorkSheet.write(inc+3, 7, round(float(ach5.read()) \* ch5\_scale\_val1 + ch5\_scale\_val2,4), Style2)

Excel\_WorkSheet.write(inc+3, 8, round(float(ach6.read()) \* ch6\_scale\_val1 + ch6\_scale\_val2,4), Style2)

***#(ONE TIME DATA SENT TO EXCEL)***

if inc==1:

Excel\_WorkSheet.write(2, 3, ch1\_scale\_val3, Style1)

Excel\_WorkSheet.write(2, 4, ch2\_scale\_val3, Style1)

Excel\_WorkSheet.write(2, 5, ch3\_scale\_val3, Style1)

Excel\_WorkSheet.write(2, 6, ch4\_scale\_val3, Style1)

Excel\_WorkSheet.write(2, 7, ch5\_scale\_val3, Style1)

Excel\_WorkSheet.write(2, 8, ch6\_scale\_val3, Style1)

File\_Name\_Excel\_Dis = "Arduino\_Excel\_Interface " + str(Hr) + "\_" + str(Min) + "\_" + str(Sec) + ".xls"

excel\_file\_loc.config(text=File\_Name\_Excel\_Dis)

***#(REPEAT THE Read\_Analog ROUTINE EVERY 2 SECS)***

master.after(2000,Read\_Analog)

***# (START UP ROUTINE FOR DISPLAY OF LABELS)***

***#(ANALOG CHANNEL DISPLAY LABELS)***

title\_1=Label(master,text="Raw Analog Data",font=("Herlvetica",14))

title\_1.grid(row=1,column=0)

a1\_label = Label(master, text = "Analog Channel-0 (0-1.000)",relief=RIDGE,font=("Helvetica", 10),width=50,background='khaki1')

a1\_label.grid(row = 2, column = 0,padx=15)

a2\_label = Label(master, text = "Analog Channel-1 (0-1.000)",relief=RIDGE,font=("Helvetica", 10),width=50,background='khaki1')

a2\_label.grid(row = 3, column = 0)

a3\_label = Label(master, text = "Analog Channel-2 (0-1.000)",relief=RIDGE,font=("Helvetica", 10),width=50,background='khaki1')

a3\_label.grid(row = 4, column = 0)

a4\_label = Label(master, text = "Analog Channel-3 (0-1.000)",relief=RIDGE,font=("Helvetica", 10),width=50,background='khaki1')

a4\_label.grid(row = 5, column = 0)

a5\_label = Label(master, text = "Analog Channel-4 (0-1.000)",relief=RIDGE,font=("Helvetica", 10),width=50,background='khaki1')

a5\_label.grid(row = 6, column = 0)

a6\_label = Label(master, text = "Analog Channel-5 (0-1.000)",relief=RIDGE,font=("Helvetica", 10),width=50,background='khaki1')

a6\_label.grid(row = 7, column = 0)

***# (ANALOG CHANNEL RAW DATA DISPLAY IN 0-1.000 VALUES)***

a1\_raw = Label(master, text = "-----",relief=GROOVE,font=("Helvetica", 10),background="linen",width=10,justify=CENTER)

a1\_raw.grid(row = 2, column = 1)

a1\_raw.config(anchor=CENTER)

a2\_raw = Label(master, text = "-----",relief=GROOVE,font=("Helvetica", 10),background="linen",width=10,justify=CENTER)

a2\_raw.grid(row = 3, column = 1)

a2\_raw.config(anchor=CENTER)

a3\_raw = Label(master, text = "-----",relief=GROOVE,font=("Helvetica", 10),background="linen",width=10,justify=CENTER)

a3\_raw.grid(row = 4, column = 1)

a3\_raw.config(anchor=CENTER)

a4\_raw = Label(master, text = "-----",relief=GROOVE,font=("Helvetica", 10),background="linen",width=10,justify=CENTER)

a4\_raw.grid(row = 5, column = 1)

a4\_raw.config(anchor=CENTER)

a5\_raw = Label(master, text = "-----",relief=GROOVE,font=("Helvetica", 10),background="linen",width=10,justify=CENTER)

a5\_raw.grid(row = 6, column = 1)

a5\_raw.config(anchor=CENTER)

a6\_raw = Label(master, text = "-----",relief=GROOVE,font=("Helvetica", 10),background="linen",width=10,justify=CENTER)

a6\_raw.grid(row = 7, column = 1)

a6\_raw.config(anchor=CENTER)

***# (ENTRY OF SPAN, OFFSET AND CHANNEL NAME)***

***# (ANALOG CHANNEL RAW DATA DISPLAY IN 0-1.000 VALUES)***

sep\_line\_1=Label(master,text="------------------------------------------------------")

sep\_line\_1.grid(row=8,column=0)

title\_2=Label(master,text="Span,Offset and Channel Name Data",font=("Herlvetica",14))

title\_2.grid(row=9,column=0)

a1\_sp\_off\_lbl = Label(master, text = "CH-1 Enter Span,Offset,Channel Name(,Seperated)",relief=RIDGE,font=("Helvetica", 10),background="lightblue",width=50,justify=CENTER)

a1\_sp\_off\_lbl.grid(row = 10, column = 0)

a2\_sp\_off\_lbl = Label(master, text = "CH-2 Enter Span,Offset,Channel Name(,Seperated)",relief=RIDGE,font=("Helvetica", 10),background="lightblue",width=50,justify=CENTER)

a2\_sp\_off\_lbl.grid(row = 11, column = 0)

a3\_sp\_off\_lbl = Label(master, text = "CH-3 Enter Span,Offset,Channel Name(,Seperated)",relief=RIDGE,font=("Helvetica", 10),background="lightblue",width=50,justify=CENTER)

a3\_sp\_off\_lbl.grid(row = 12, column = 0)

a4\_sp\_off\_lbl = Label(master, text = "CH-4 Enter Span,Offset,Channel Name(,Seperated)",relief=RIDGE,font=("Helvetica", 10),background="lightblue",width=50,justify=CENTER)

a4\_sp\_off\_lbl.grid(row = 13, column = 0)

a5\_sp\_off\_lbl = Label(master, text = "CH-5 Enter Span,Offset,Channel Name(,Seperated)",relief=RIDGE,font=("Helvetica", 10),background="lightblue",width=50,justify=CENTER)

a5\_sp\_off\_lbl.grid(row = 14, column = 0)

a6\_sp\_off\_lbl = Label(master, text = "CH-6 Enter Span,Offset,Channel Name(,Seperated)",relief=RIDGE,font=("Helvetica", 10),background="lightblue",width=50,justify=CENTER)

a6\_sp\_off\_lbl.grid(row = 15, column = 0)

***#(ENTRY FOR SPAN,OFFSET AND CHANNEL NAME)***

a1\_sp\_off=Entry(master,font=("Helvetica",10),background="orange",width=30)

a1\_sp\_off.grid(row=10,column=1)

a2\_sp\_off=Entry(master,font=("Helvetica",10),background="orange",width=30)

a2\_sp\_off.grid(row=11,column=1)

a3\_sp\_off=Entry(master,font=("Helvetica",10),background="orange",width=30)

a3\_sp\_off.grid(row=12,column=1)

a4\_sp\_off=Entry(master,font=("Helvetica",10),background="orange",width=30)

a4\_sp\_off.grid(row=13,column=1)

a5\_sp\_off=Entry(master,font=("Helvetica",10),background="orange",width=30)

a5\_sp\_off.grid(row=14,column=1)

a6\_sp\_off=Entry(master,font=("Helvetica",10),background="orange",width=30)

a6\_sp\_off.grid(row=15,column=1)

sep\_line\_2=Label(master,text="------------------------------------------------------")

sep\_line\_2.grid(row=16,column=0)

title\_2=Label(master,text="Scaled Data",font=("Herlvetica",14))

title\_2.grid(row=17,column=0)

***#(SCALED DATA DISPLAY)***

a1\_scaled = Label(master, text = "Analog Channel-0 Scaled",relief=GROOVE,font=("Helvetica", 10),width=50,background='seagreen1')

a1\_scaled.grid(row = 18, column = 0)

a2\_scaled = Label(master, text = "Analog Channel-1 Scaled",relief=GROOVE,font=("Helvetica", 10),width=50,background='seagreen1')

a2\_scaled.grid(row = 19, column = 0)

a3\_scaled = Label(master, text = "Analog Channel-2 Scaled",relief=GROOVE,font=("Helvetica", 10),width=50,background='seagreen1')

a3\_scaled.grid(row = 20, column = 0)

a4\_scaled = Label(master, text = "Analog Channel-3 Scaled",relief=GROOVE,font=("Helvetica", 10),width=50,background='seagreen1')

a4\_scaled.grid(row = 21, column = 0)

a5\_scaled = Label(master, text = "Analog Channel-4 Scaled",relief=GROOVE,font=("Helvetica", 10),width=50,background='seagreen1')

a5\_scaled.grid(row = 22, column = 0)

a6\_scaled = Label(master, text = "Analog Channel-5 Scaled",relief=GROOVE,font=("Helvetica", 10),width=50,background='seagreen1')

a6\_scaled.grid(row = 23, column = 0)

***# (ANALOG CHANNEL SCALED DATA)***

a1\_sc\_dis = Label(master, text = "-----",relief=GROOVE,font=("Helvetica", 10),background="cyan",width=10,justify=CENTER)

a1\_sc\_dis.grid(row = 18, column = 1)

a1\_sc\_dis.config(anchor=CENTER)

a2\_sc\_dis = Label(master, text = "-----",relief=GROOVE,font=("Helvetica", 10),background="cyan",width=10,justify=CENTER)

a2\_sc\_dis.grid(row = 19, column = 1)

a2\_sc\_dis.config(anchor=CENTER)

a3\_sc\_dis = Label(master, text = "-----",relief=GROOVE,font=("Helvetica", 10),background="cyan",width=10,justify=CENTER)

a3\_sc\_dis.grid(row = 20, column = 1)

a3\_sc\_dis.config(anchor=CENTER)

a4\_sc\_dis = Label(master, text = "-----",relief=GROOVE,font=("Helvetica", 10),background="cyan",width=10,justify=CENTER)

a4\_sc\_dis.grid(row = 21, column = 1)

a4\_sc\_dis.config(anchor=CENTER)

a5\_sc\_dis = Label(master, text = "-----",relief=GROOVE,font=("Helvetica", 10),background="cyan",width=10,justify=CENTER)

a5\_sc\_dis.grid(row = 22, column = 1)

a5\_sc\_dis.config(anchor=CENTER)

a6\_sc\_dis = Label(master, text = "-----",relief=GROOVE,font=("Helvetica", 10),background="cyan",width=10,justify=CENTER)

a6\_sc\_dis.grid(row = 23, column = 1,pady=0)

a6\_sc\_dis.config(anchor=CENTER)

sep\_line\_3=Label(master,text="")

sep\_line\_3.grid(row=24,column=0,pady=5)

***# (PROGRAM STATUS INDICATOR)***

stat\_ind = Label(master, text = "Status: Waiting.....",relief=GROOVE,font=("Helvetica", 10),width=50,background='gold')

stat\_ind.grid(row= 26, column = 0,pady=5)

excel\_file = Label(master, text = "Excel File Store Location:",relief=GROOVE,font=("Helvetica", 10),width=50,background='lightcyan')

excel\_file.grid(row= 25, column = 0)

excel\_file\_loc=Label(master, text = "--",relief=GROOVE,font=("Helvetica", 8),width=40,background='lightblue')

excel\_file\_loc.grid(row=25,column=1)

***# (BUTTON WIDGETS)***

***#(START READ BUTTON DEFINITION)***

start\_read\_but = Button(master, text = " Start Reading ")

start\_read\_but.pack

start\_read\_but.bind('<Button-1>',Start\_Read)

start\_read\_but.grid(row = 3, column = 2,padx=5)

***# (QUIT BUTTON DEFINITION)***

quit\_but=Button(master,text=" Quit Program ")

quit\_but.bind('<Button-1>',Quit\_prog)

quit\_but.grid(row = 5, column = 2,padx=5)

***# (ABOUT BUTTON DEFINITION)***

about\_but=Button(master,text=" About Program ")

about\_but.bind('<Button-1>',About\_prog)

about\_but.grid(row = 7, column = 2,padx=5)

***#(ARDUINO SETUP OF BAUD RATE AND COM PORT NUMBER)***

try:

ardboard = Arduino('COM3', baudrate=9600)

except:

messagebox.showinfo("COM Read Error", "Please check Cables and that you are connected to a Arduino with Firmata Loaded...")

exit()

it = util.Iterator(ardboard)

it.start()

ardboard.analog[0].enable\_reporting()

ach1=ardboard.get\_pin('a:0:i')

ach2=ardboard.get\_pin('a:1:i')

ach3=ardboard.get\_pin('a:2:i')

ach4=ardboard.get\_pin('a:3:i')

ach5=ardboard.get\_pin('a:4:i')

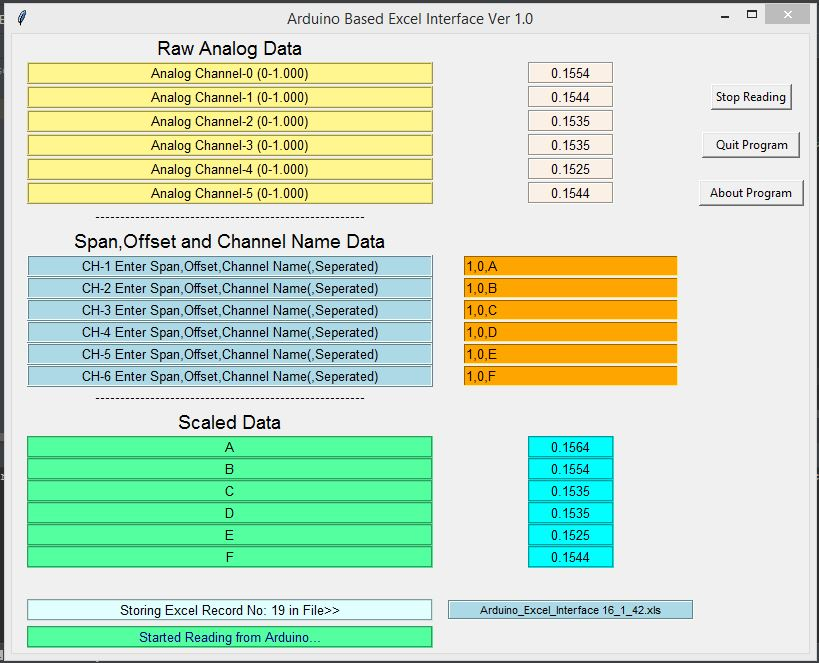
ach6=ardboard.get\_pin('a:5:i')

***#(MAIN LOOP ENTRY)***

mainloop()

Now let us look at how the actual program interface looks like. As mentioned in the earlier sections the program will accept the span, offset and channel name as values separated by commas and then use the span and offset to calculate the ***y=mx+c*** where y is the calculated output and m is the span or scale and c is the offset. The Arduino Firmata library will send out the Analog values in the range 0 t 1.00 which needs to be scaled appropriately. So, with a few hundred lines of Python code, we have a fully operational data logger that not only displays the scaled data in real time but stores it in an Excel file for further analysis. The excel file name is self-generated depending on the time and will be in the format :

**Arduino\_Excel\_Interface\_hh\_mm\_ss.xlsx** where hh is present hour and mm is present minute and ss is present second.



Excel File name

Scaled data display using y=mx+c

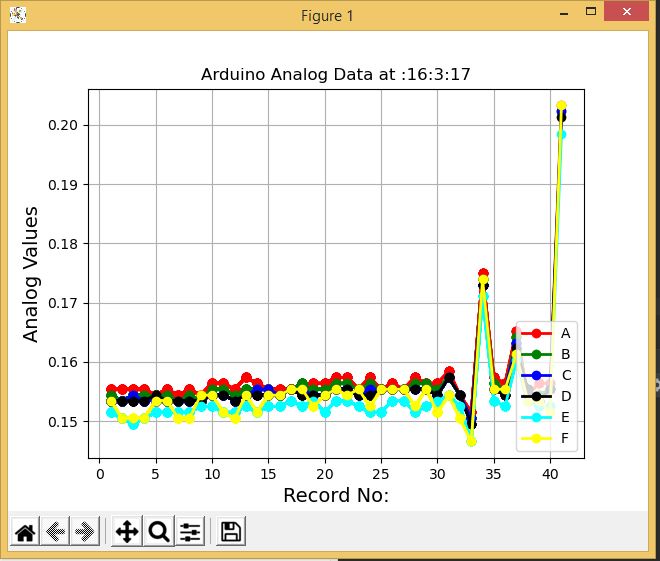
Raw Data display

Control buttons

Span, Offset Channel name data entry

***Fig 4: Python program output with raw data, scaled data and excel file store location.***

The above screenshot was the image of the program interface and in the next figure the Matplotlib library output is shown with all the different channels in different colours and with different point markers.



***Fig 5: Matplotlib output showing plot of various channels in different colours and names of channels as legends.***

The figure shows the matplotlib library output generated by the Python program with the different channels plotted in different colours and the name of the channels displayed at the bottom right. This plot is real time and will dynamically change the scaling as the values are being read from the Python program.

Now finally an excerpt of the Excel worksheet generated is shown below with all the channel names and time stamping done. The data stored in Excel can be used for further processing and generation of graphs and other data manipulation.



**Fig 6: Data stored in the Excel file**

The excerpt above shows the data stored in the Excel file with date, time stamping and the analog channel data stored with the channel names. The data thus stored in the excel file can be used for further manipulation.

**vi) Conclusion of Part-2**

This brings us to the end of Part-2 where we made rapid progress from a small Tiny 85 processor to an Arduino UNO with a Python interface. We built a 6 channel Data logger using an Arduino UNO and Python which even had data displayed , plotted and stored in an Excel file. The next part is even more advanced where we will be using the ESP 32 from Expressif and program it using Micropython which is the programming language of the future which is a scaled down version of Python for use on microcontrollers.

**vi) Resources for Part-2**

**1. Components**

All components can be purchased from:

[www.amazon.in](http://www.amazon.in)

[www.in.element14.com](http://www.in.element14.com)

**2. Circuit Diagram/Program listing**

Git Hub:

https://github.com/SJayarajan/Embedded-Systems-Development-from-the-Ground-Up.git

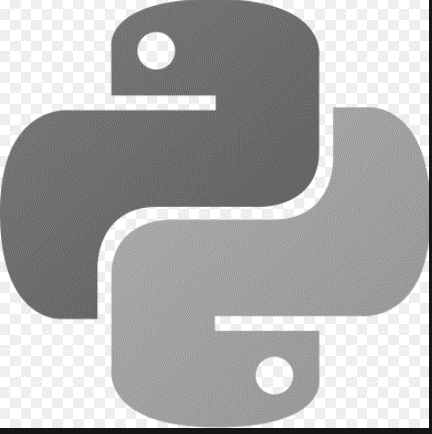
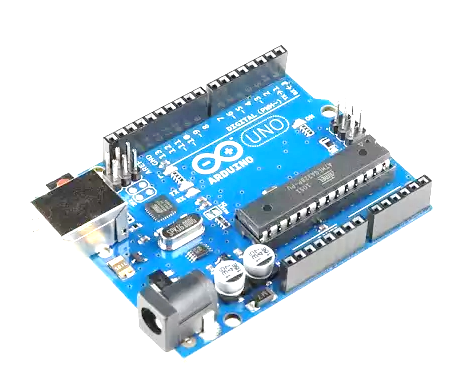
(Check files with Part-2 as Prefix in the file names)

**3. Further Readings**

For Python: https://www.python.org/

**4. Other Project Ideas**

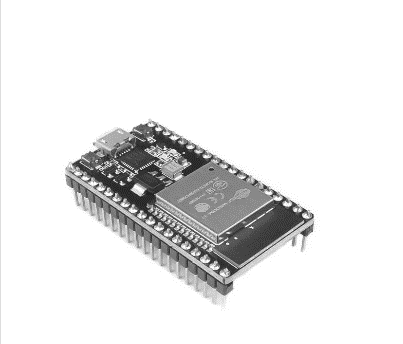
* Water irrigation system for garden plants.
* PWM based DC drill speed control.
* NIMH battery charger.
* Water level controller



"***Python is fast enough for our site and allows us to produce maintainable features in record times, with a minimum of developers*,"** Cuong Do, Software Architect, [YouTube.com](http://youtube.com/)**.**

**Part-3**

**ESP-32 programming with MicroPython**



**i) Introduction and basic details**

From the other two systems the ATtiny 85 and the Arduino Uno we move on to a very advanced microcontroller system called the ESP-32. The more amazing part of this system is that it can be programmed in MicroPython. This is not however a book with all details of the ESP-32. This Part-3 will introduce the readers to this remarkable micro-controller kit from Ekpressif. In the Part-1 and Part -2 we used rather primitive versions of microcontroller kits which can be used for simple projects. The turnaround time for the projects for Part-1, Part-2 is very short and most of the projects using the systems can be developed in no time. With the ESP-32 which uses MicroPython are slightly more complex and have numerous integrated systems like WiFi and Bluetooth. This particular part will take a detour from the details of Part-1,2 and move to a high-level programming language called Python. The ESP-32 uses a scaled down version of Python and provides many libraries to control the GPIO pins, WiFi, Bluetooth etc. For most of the microcontroller programmers, as their programs become more complex the low RAM and program memory soon catch up. The ESP-32 will not have such a problem since the RAM and the program memory are adequate for very complex programs. Moreover, there is no need to add WiFi and Bluetooth shields since they all are part of the ESP-32. So let us take a small detour and embark on the amazing journey of meeting and using the ESP-32.

**ii) The ESP-32**

The ESP -32 which is covered in this book is the Ekspressif WROOM -32 E which is the most commonly used board and will be the subject of this part of the book.

A brief description of the specifications are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.No** | **Parameter** | **Value** | **Remarks** |
| 1. | CPU | ESP32-D0WD-V3 embedded, Xtensa® dual-core  32-bit LX6 microprocessor, up to 240 MHz |  |
| 2. | ROM | 448 KB for Booting and Core Functions |  |
| 3. | SRAM | 520 KB for Data and Instructions. |  |
| 4. | Hardware Interfaces. | SD card, UART, SPI, SDIO, I2C, LED  PWM, Motor PWM, I2S, IR, pulse counter, GPIO,  capacitive touch sensor, ADC, DAC, Two-Wire  Automotive Interface (TWAI®, compatible with  ISO11898-1) | On Board SD card not provided. |
| 5. | Oscillator | 40 MHz |  |
| 6. | Operating Voltage | 3.0 to 3.6 V DC |  |
| 7. | Bluetooth | V4.2 BR/EDR and Bluetooth LE  Specification |  |

**Table: 3.1 ESP -32 Basic Specifications**

The important thing to keep in mind here is that the input voltage is 5 V to be supplied at the Vin pin. The system is a 3.3 V device and all interface used should be of 3.3 V levels. Using 5 Volts as inputs to the pins can damage the device. In this section we will be programming the chip with MicroPython which is a scaled down version of Python that is used exclusively for microcontrollers. The main thing about the MicroPython is that it is loaded directly into the device and resides in two files named ***boot.py and main.py***. The main.py name is standard and the system will only execute the statements in the main.py file. There are many IDE’s that can be used to program the ESP-32 including Arduino but to program using MicroPython the best and easy to use IDE is called ***Thonny***. It can be downloaded from the link below:

[**https://thonny.org/**](https://thonny.org/)

This is a simple and light weight IDE and has all the tools to program and run the programs on the microcontroller.

After the Thonny IDE has been downloaded and installed (there are Windows, Mac and Linux Versions) we are ready to start working with this amazing microcontroller system.

**iii) The Thonny IDE**

Before the ESP -32 can be used the boot.py file must be loaded into the ESP-32.

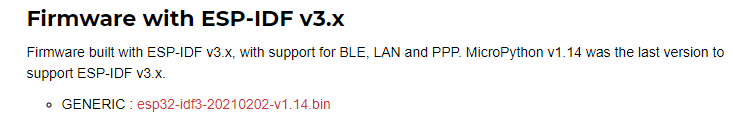
After connecting the ESP-32 to your computer at the USB port check the COM number that is displayed when the device is connected to the USB port.

If the COM port is displayed once the ESP-32 is connected then the unit is ready to install the firmware and make it ready to use.

The firmware for the ESP-32 can be downloaded from the link:

[**https://micropython.org/download/esp32/**](https://micropython.org/download/esp32/)

From this page select the Generic Firmware and download the file to a known location.



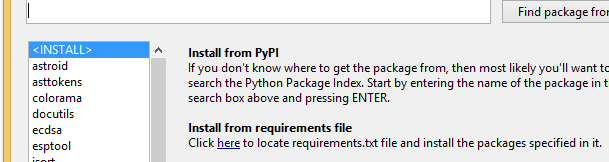
**Fig:3.1 Generic Firmware for ESP-32 from MicroPython website.**

With the firmware downloaded the file is ready to be installed in the device.

The Thonny IDE uses the esptool plugin to download the firmware.

The install the plugin Select **Tools > Manage Plug ins**.

The screen will open to the available plugins and from the box on the left select esptool.



Select the esptool Plugin to enable firmware download.

**Fig: 3.2 Managing Plugins in Thonny and selecting the esptool plugin.**

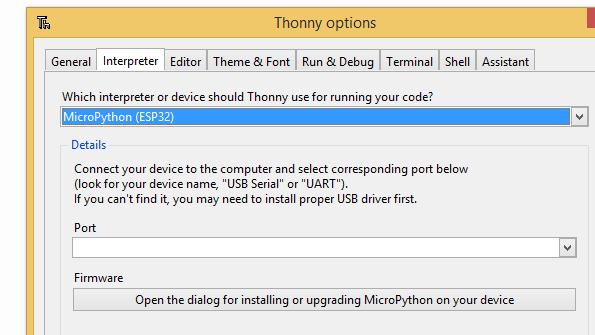
After installing the esptool plugin the Thonny IDE is ready for download of the firmware.

To download the firmware, click **Tools > Options > Interpreter**.

On this page select the COM Port that the ESP-32 is connected to and select the firmware file that was downloaded from the MicroPython website.

Click OK to start download and press the Boot push button of the ESP-32 to start the download. The button can be released after download starts.

Select the MicroPython (ESP-32) interpreter



Select the Port where the ESP-32 is connected

Select the firmware File downloaded

**Fig 3.3: Selection of Port, Interpreter and Firmware**

After the firmware has been downloaded the three > sins will appear in the console that is called **REPL(READ-EVALUATE-PRINT-LOOP)** which is a full-fledged Python (in this case MicroPython) prompt.

At the >>> prompt try the following commands:

* >>> print (” Hello World”) ***Enter***

Output: Hello World.

* >>> 5+6 ***Enter***

11

Congratulations!! If you have reached so far then the ESP-32 is ready to be programmed with Thonny. The ESP-32 will only have one file called boot.py and all programming should be done in a Python file named **main.py**.

This has to be followed without any change.

**iv) The Standard Libraries**

Before we start to program the ESP-32 it would be good to use the interpreter to display some data about the ESP-32. These commands are by no means exhaustive but are used to showcase some of the many capabilities of the ESP-32.

On the REPL prompt try the following commands:

1. **>>>help(“modules”)**

This command is used to display the help on the various modules or libraries used in the ESP -32. Two of the most important are the “uos” and the “machine” module that provide OS (Operating System) and access to hardware pins respectively.

1. **>>>import uos**

**>>>uos.listdir()**

This will display the directories/Files in the ESP-32 and for the default ESP-32 we will see only the boot.py file which is the default file.

***All programs that we write in the ESP-32 will need to eb in a file name main.py.***

1. **>>>from machine import Pin**

**>>>led=Pin(2,Pin.OUT)**

**>>>led.value(1)**---------The ON Board LED is turned ON.

**>>>led.value(0)**---------The On Board LED is turned OFF.

In this code snippet from the machine library, we import Pin which is used to control the hardware pins. Then we assign a “led” variable to on Board LED at pin 2 and set its value to 1 to turn it on and to 0 to turn it off.

Thus, we have seen that the ESP-32 is indeed a very powerful system that can be programmed to control Pins, send data and even read sensors.

We are thus equipped now to program our ESP-32 to do a real time project that will harness the true power of the ESP-32.

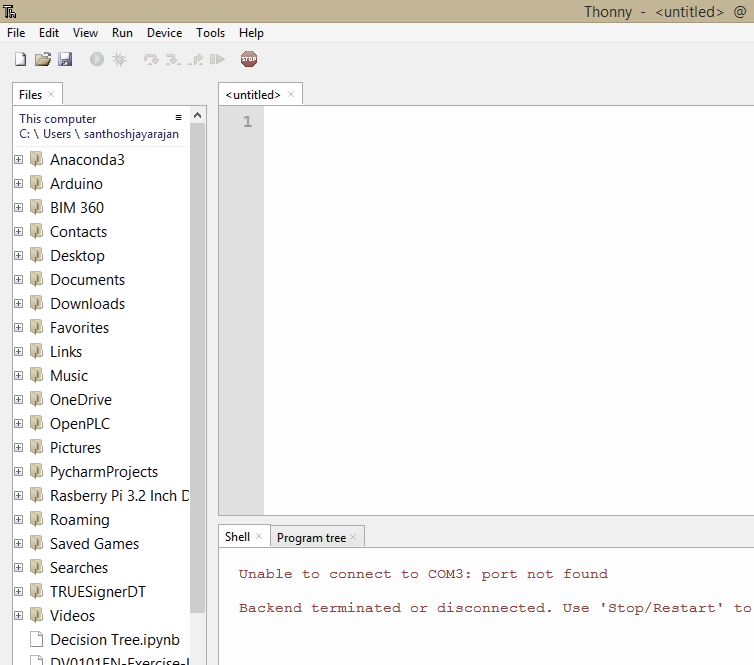
The project is a web-based Weather Station that will display the temperature and humidity on a web page and will have values updated in real time using AJAX. This project will take you through the following steps:

* Programming the ESP-32 in Thonny.
* Setting up a Wifi connection on your home Wifi network.
* Open a WebSocket connection to establish connection with the Client and server.
* Use AJAX code to update the web page without refreshing the web page.
* Future enhancements that are possible.

**v. Programming the ESP-32 with Thonny**

Before we start really programming the ESP -32 it would be a great idea to get used to the different work areas of the Thonny IDE.

The Thonny IDE when opened will present you with 3 distinct work or action areas that need to be understood before using the IDE.



Output or REPL interface

Program Entry space with file name as “untitled”

Project Files either on Device or on the System

**Fig 3.4: Thonny IDE Work and Action Areas**

Each of the areas mentioned has a separate and role to play.

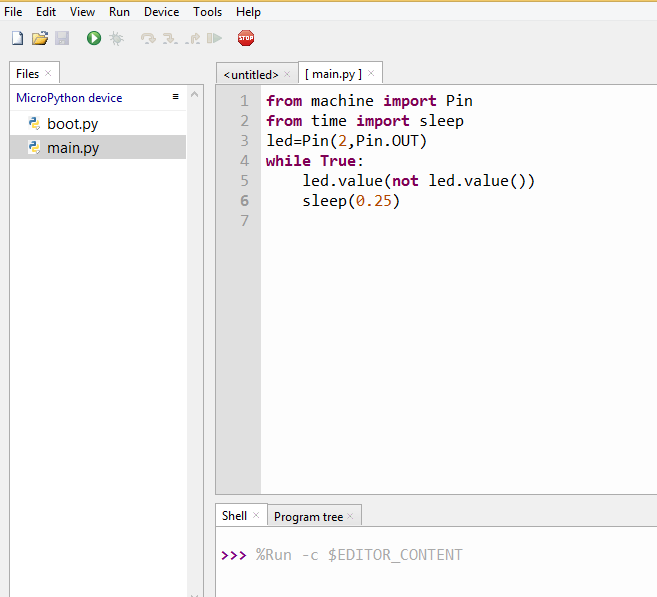
The use of each of the areas is as given below:

* File or Project Tree display will display the files on the device (when connected) or on the system which had been stored from a separate programming session. The ESP-32 will initially be configured and started will have only one file called **boot.py**. This file is required for initialisation and for the boot process. **This file should not be deleted or modified under any condition. All the programming should be done in a file named main.py.** No other name should be provided for this file or the ESP 32 will not function.
* The REPL interface will display the output of the program when the program in the ESP -32 is running. When the program has been stopped on the ESP-32 the REPL(Read- Evaluate-Print-Loop)is presented by >>> and sets the ESP-32 in the interface mode and step by step programming can be done on the ESP -32 in real time.
* The Program Entry place is the place where the program in MicroPython cn be actually written and can be downloaded to the ESP -32.

**Let us look at these various operations in some detail:**

When the ESP-32 is connected to the computer USB port the program that has been downloaded into the ESP-32 will start running and the REPL mode will be disabled.

To explain the actual conditions a small blink program has been loaded into the ESP-32 which will be explained shortly.



Disabled REPL >>> prompt

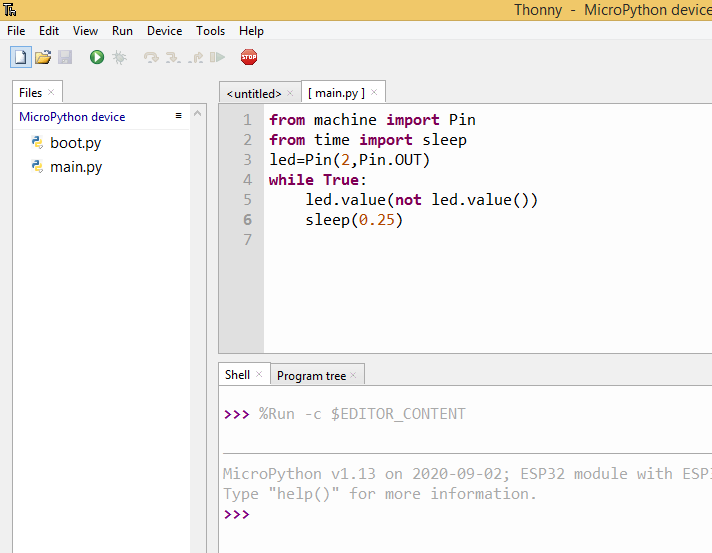
Files in the ESP-32.

boot.py and main.py which has the program

Program already running on the ESP -32.

**Fig 3.5: Thonny IDE with program running and REPL disabled.**

At this stage by clicking on the STOP button the ESP-32 will go into a halt mode and the programming can be done offline on the programming editor.After clicking STOP the display is as follows:



REPL Mode enabled.

STOP Button to stop the program running and enter REPL mode.

**Fig 3.6: Thonny IDE with program stopped and REPL enabled**

To check the REPL functioning enter the following at the >>>prompt

>>> 4+5

9

>>>

Now to restart the program click on the **RUN** button and the program will start running.

In the **STOP** mode the program can be modified and when saving in done the IDE will prompt if the saving has to be done on the device or on the computer. If Device is selected the main.py on the device will be overwritten or else the program can be saved for offline editing on the computer.

Before we start with the project using the ESP-32 a quick look at the sample program that is running on the ESP-32.This is a very simple program and is the “blink LED” program or the Hello World program of the microcontroller world.

The program is as follows:

**from machine import Pin**

**from time import sleep**

**led=Pin(2,Pin.OUT)**

**while True:**

**led.value(not led.value())**

**sleep(0.25)**

The first line import Pin functionality from the machine library. This is required to control and configure the microcontroller pins. In this case Pin(2) is the on board LED which is blue in colour. The next library imported in time where a sleep function is used to create a delay for the blink project.

The next statement **led=Pin(2,Pin.OUT)**will create a new object called led which is on Pin 2 and it is configured as an OUTPUT with **Pin.OUT**.

Then you enter into a indefinite loop with **while True:**

and toggle the LED state with **led.value(not led.value())**

The delay is included by **sleep (0.25) for 0.25 Secs or 250 Msec.**

That was the basics of using the Thonny IDE which we will using in our project which will be dealt with step by step in the next section. There is going to be HTML, Web Sockets and more. Please note that this is not a book on the various web technologies so we will be introducing only the web technologies at the very basic level and just enough to get you going.

For detailed online tutorials on web technologies a great web site is

**W3 Schools**

***www.w3schools.com***

**vi. Weather Station with ESP-32**

This project is a very small one that will provide a small web-based weather station. This weather station is expandable platform where the ESP-32 will measure the temperature and humidity and will predict the weather based on some set rules.

The web page will be constantly updated with new information and temperature and humidity will be updated constantly. The ESP-32 will use WiFi to connect to the router and a client can communicate with the server to read the data. The ESP-32 has the distinct advantage that it can be programmed in Micro Python and has all the built-in functionalities for communication. The various added libraries can help to read the DHT-22 sensor and also create a socket connection.

We will start by defining the wiring for the DHT 22 Sensor which is connected to pin 23 of the ESP=32.The **dht** library will provide the functions to read and return the temperature and humidity. The circuit diagram is given below:

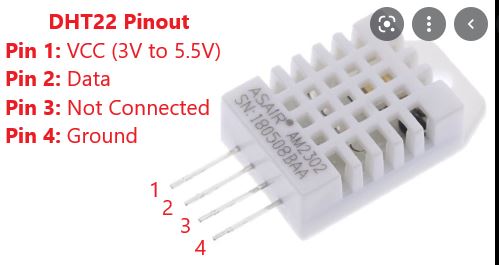


**Fig 3.7: DHT 22 to ESP -32 inter connections.**

The circuit consists of the 220 V to 5 V DC power supply connected to the Vin of the ESP-32 controller.

The data pin is connected to the GPIO23 of the ESP-32. The pins of the DHT-22 are as given below:

|  |  |
| --- | --- |
| **Pin** | **Use** |
| **1** | **VCC- 5 V DC Supply** |
| **2** | **Data – ESP-32 GPIO -23** |
| **3** | **Not Used** |
| **4** | **Power Ground** |



**Fig 3.8: DHT 22 Pinout (Figure Courtesy *IoT with Us*.)**

The steps to be followed in this project is as follows:

* Set up a Wifi connection with the router by providing the User name and password.
* Set up a socket connection to provide a channel for communications.
* Access the server from the browser using the IP address and port number.
* The data is read continually by the web page till the connection is terminated.

WiFi

PC as Client Enter IP address with port number

Router

WiFi

ESP -32 Running the Micro Python Program

**Fig 3.9: Web based Weather Station Block Diagram.**

Thus, with the basic information settled let us start the programming of the ESP-32 with Micro Python. The program sequence is as follows:

Documentation of code using # sign.

1. Include all the libraries.

# ESP 32 Base Weather Station Version-1

#Programmed by Santhosh Jayarajan

#Date: 23/8/2021

#Using Web Sockets and Wifi connection

#

#IMPORT LIBRARIES

**import machine**

Import required libraries.

**import time**

**import socket**

**import network**

**import Wifi\_Creds**

**import dht**

The libraries that are included are as follows:

* **Machine-**Library is used for the control of the LED to display the information of the connection.
* **time-L**ibrary is used to introduce the delays and get the time and data.
* **Socket-**Library to open and use the socket connection.
* **Network-**Library to control the internet network connection.
* **WiFi Creds** – This is a file we will create that holds the router user name and password.
* **dht**- Library to read and configure the DHT -22 Sensor.

*Note: In Python we use the # sign to comment out the text for documentation. In all forms of programming the use of extensive documentation cannot be over emphasised. Make it a habit to provide documentation for all parts of the code so that the reader can understand your thought process and also to refresh your memory at a later date.*

The program in its entirety is presented below with explanation of all the parts:

**# ESP 32 Base Weather Station Version-1**

**#Programmed by Santhosh Jayarajan**

**#Date: 4/11/2021**

**#Using Web Sockets and WifI connection**

**#**

**#IMPORT LIBRARIES**

import machine

import time

import socket

import network

import Wifi\_Creds

import dht

**# START OF CODE**

led = machine.Pin(2,machine.Pin.OUT)

led.off()

**# WIFI SETUP**

Wifi=network.WLAN(network.STA\_IF)

Wifi.active(True)

Wifi.connect(Wifi\_Creds.ssid,Wifi\_Creds.password)

if Wifi.isconnected()==True:

led.on()

print(Wifi.isconnected())

print(Wifi.ifconfig())

**# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**# CONFIGURE THE SOCKET CONNECTION**

**# OVER TCP/IP**

**# AF\_INET - USE INTERNET PROTOCOL V4 ADDRESSES**

**# SOCK\_STREAM MEANS THAT IT IS A TCP SOCKET.**

**# SOCK\_DGRAM MEANS THAT IT IS A UDP SOCKET.**

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

s.bind(('',180)) # specifies that the socket is reachable by any address the machine happens to have

s.listen(5) # max of 5 socket connections

**#DHT SENSOR INITIALIZATIONS**

d = dht.DHT22(machine.Pin(23))

**#WEB PAGE TO BE DISPLAYED**

def web\_page():

**#GET THE DHT READINGS**

d.measure()

Temp\_C=d.temperature()

Temp\_C=35.0

Hum\_Per = d.humidity()

Hum\_Per=75.0

Forecast1="Weather Forcast here"

Forecast2="Detailed Weather Forecast here"

**#TEMPERATURE FORECAST**

if Temp\_C<=10.0:

Forecast1="The Temperature is very Cold and "

if Temp\_C>10.0 and Temp\_C<=19.0:

Forecast1="The Temperature is Quite Cold and "

if Temp\_C>19.0 and Temp\_C<=29.0:

Forecast1="The Temperature is Marginally Cold and "

if Temp\_C>29.0 and Temp\_C<=39.0:

Forecast1="The Temperature is Quite Warm and "

if Temp\_C>39.0 and Temp\_C<=49.0:

Forecast1="The Temperature is Really Hot and "

**#HUMIDITY FORECAST**

if Hum\_Per<=10.0:

Forecast1 = Forecast1+"Very Dry"

if Hum\_Per>=10.0 and Hum\_Per<30.0:

Forecast1 = Forecast1+"Dry"

if Hum\_Per>=30.0 and Hum\_Per<50.0:

Forecast1 = Forecast1+"Slightly Dry"

if Hum\_Per>=50.0 and Hum\_Per<70.0:

Forecast1 = Forecast1+"Slightly Wet"

if Hum\_Per>=70.0 and Hum\_Per<90.0:

Forecast1 = Forecast1+"Very Wet"

**# DETAILED FORECAST**

**# TEMPERATURE FORECAST DETAILED**

if Temp\_C<=10.0:

Forecast2="The Weather is Chill- Wear warm Clothes and "

if Temp\_C>10.0 and Temp\_C<=19.0:

Forecast2="The Weather is a Bit Chill -Wear Warm Clothes and "

if Temp\_C>19.0 and Temp\_C<=29.0:

Forecast2="The Weather is Pleasent - Wear Light Clothes and "

if Temp\_C>29.0 and Temp\_C<=39.0:

Forecast2="The Weather is Warm - Wear very Light Clothes and "

if Temp\_C>39.0 and Temp\_C<=49.0:

Forecast2="The Weather is Really Warm - Wear Beach Clothes and "

**#HUMIDITY FORECAST DETAILED**

if Hum\_Per<=10.0:

Forecast2 = Forecast1+"the Air is Very Dry (Use Moisturizers)"

if Hum\_Per>=10.0 and Hum\_Per<30.0:

Forecast2 = Forecast1+"the Air is Dry (Use Moisturizers)"

if Hum\_Per>=30.0 and Hum\_Per<50.0:

Forecast2 = Forecast1+"the Air is Slightly Dry (Use Sun Screen)"

if Hum\_Per>=50.0 and Hum\_Per<70.0:

Forecast2 = Forecast1+"the Air is Slightly Moist (Use Sun Screen)"

if Hum\_Per>=70.0 and Hum\_Per<90.0:

Forecast2 = Forecast1+"the Air is Very Moist (Use Normal Cream)"

html\_page = """<!DOCTYPE html>

<html>

<head>

<style>

table {

width:100%;

}

table, th, td {

border: 1px solid black;

border-collapse: collapse;

border:ridge 5px red;

background-color:black; color:yellow;

}

th, td {

padding: 15px;

text-align: left;

}

TD{font-family: Calibri; font-size: 10pt;}

</style>

</head>

<meta http-equiv="refresh" content="5">

<body bgcolor="MediumSeaGreen">

<h1><center><font face="Calibri">Weather Station Ver 1.0</font></center></h1>

<table>

<tr>

<th><h2><center>Temperature in Deg C</center></h2></th>

<th><h2><center>Humidity in %</center></h2></th>

<th><h2><center>Forecast</center></h2></th>

</tr>

<tr>

<td><center>""" + str(Temp\_C) + """</center></td>

<td><center>""" + str(Hum\_Per) + """</center></td>

<td><center>""" + str(Forecast1) + """</center></td>

</tr>

</table>

<br>

<br>

<h1><center><font face="Calibri">Weather Forcast Details</font></center></h1>

""" + str(Forecast2) + """

<br><br>

</body>

</html>"""

return html\_page

while True:

**# SOCKET ACCEPT()**

conn, addr = s.accept()

print("Got connection from %s" % str(addr))

**#SOCKET RECEIVE()**

request=conn.recv(1024)

print("")

print("Content %s" % str(request))

**#SOCKET SEND()**

request = str(request)

**#CREATE A SOCKET REPLY**

response = web\_page()

conn.send('HTTP/1.1 200 OK\n')

conn.send('Content-Type: text/html\n')

conn.send('Connection: close\n\n')

conn.sendall(response)

**#SOCKET CLOSE()**

conn.close()

The program starts by including all the required libraries that will be used by the program. The libraries used and their explanation are already mentioned above.

After all the libraries have been imported the next section of the program is to connect to your home WIFI network.

For this we use a user defined file titled: ***Wifi\_Creds.py*** that contains all the credentials for the Wifi connection.

The entries in the above file are as below:

ssid="XXXXXX"

password="YYYYYYY"

Which are the Username and password of the home Wifi Network. Please replace the entries with the details of your network.

The connection is established by the following code:

**# START OF CODE**

**# SART WIFI AND ESTABLISH CONNECTION.**

led = machine.Pin(2,machine.Pin.OUT)

led.off()

Wifi=network.WLAN(network.STA\_IF)

Wifi.active(True)

Wifi.connect(Wifi\_Creds.ssid,Wifi\_Creds.password)

if Wifi.isconnected()==True:

led.on()

print(Wifi.isconnected())

print(Wifi.ifconfig())

The on-board LED is used as an indicator for the WiFi connection establishment.

We first initialise a class called Wifi and use the Station Interface Mode.

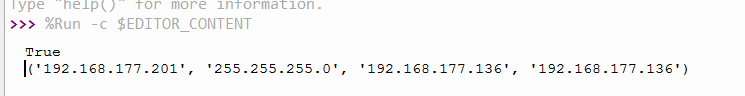
The Wifi is made active and the **connect** method is called to make the connection with the Wifi Credentials from the Wifi\_Cred.py file. If the Wifi is connected the on-board LED is turned on and the Config details printed to the console by the commands:

print(Wifi.isconnected())-**Will be True for good Connection**

print(Wifi.ifconfig())-**IP Address of ESP -32**

will mention the IP address of the ESP -32 allotted by the router. This IP address will be used to evoke the Web page from the browser using in this case 192.168.177.201:80.

IP Address that has been allotted by the Router



The next part of the program will establish a Socket connection so that the Web browser can communicate with the ESP-32. That is done by the following code:

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

s.bind(('',80))

s.listen(5)

The socket connection is initialised as a Station Interface and with streaming option and bound the port 80. This can be changed to any other available Port numbers.

The next part of the program will initialise the DHT 22 sensor which is connected to Port 23.

**# DHT sensor initializations**

d = dht.DHT22(machine.Pin(23))

The next part of the program is to define a function called **web\_page()** that will calculate the temperature and humidity values and also make some assumptions of the weather based on the temperature and humidity.

def web\_page():

**#GET THE DHT READINGS**

d.measure()

Temp\_C=d.temperature()

Temp\_C=35.0

Hum\_Per = d.humidity()

Hum\_Per=75.0

Forecast1="Weather Forcast here"

Forecast2="Detailed Weather Forecast here"

**#TEMPERATURE FORECAST**

if Temp\_C<=10.0:

Forecast1="The Temperature is very Cold and "

if Temp\_C>10.0 and Temp\_C<=19.0:

Forecast1="The Temperature is Quite Cold and "

if Temp\_C>19.0 and Temp\_C<=29.0:

Forecast1="The Temperature is Marginally Cold and "

if Temp\_C>29.0 and Temp\_C<=39.0:

Forecast1="The Temperature is Quite Warm and "

if Temp\_C>39.0 and Temp\_C<=49.0:

Forecast1="The Temperature is Really Hot and "

**#HUMIDITY FORECAST**

if Hum\_Per<=10.0:

Forecast1 = Forecast1+"Very Dry"

if Hum\_Per>=10.0 and Hum\_Per<30.0:

Forecast1 = Forecast1+"Dry"

if Hum\_Per>=30.0 and Hum\_Per<50.0:

Forecast1 = Forecast1+"Slightly Dry"

if Hum\_Per>=50.0 and Hum\_Per<70.0:

Forecast1 = Forecast1+"Slightly Wet"

if Hum\_Per>=70.0 and Hum\_Per<90.0:

Forecast1 = Forecast1+"Very Wet"

**# DETAILED FORECAST**

**# TEMPERATURE FORECAST DETAILED**

if Temp\_C<=10.0:

Forecast2="The Weather is Chill- Wear warm Clothes and "

if Temp\_C>10.0 and Temp\_C<=19.0:

Forecast2="The Weather is a Bit Chill -Wear Warm Clothes and "

if Temp\_C>19.0 and Temp\_C<=29.0:

Forecast2="The Weather is Pleasent - Wear Light Clothes and "

if Temp\_C>29.0 and Temp\_C<=39.0:

Forecast2="The Weather is Warm - Wear very Light Clothes and "

if Temp\_C>39.0 and Temp\_C<=49.0:

Forecast2="The Weather is Really Warm - Wear Beach Clothes and "

**# HUMIDITY FORECAST DETAILED**

if Hum\_Per<=10.0:

Forecast2 = Forecast2+"the Air is Very Dry (Use Moisturizers)"

if Hum\_Per>=10.0 and Hum\_Per<30.0:

Forecast2 = Forecast2+"the Air is Dry (Use Moisturizers)"

if Hum\_Per>=30.0 and Hum\_Per<50.0:

Forecast2 = Forecast2+"the Air is Slightly Dry (Use Sun Screen)"

if Hum\_Per>=50.0 and Hum\_Per<70.0:

Forecast2 = Forecast2+"the Air is Slightly Moist (Use Sun Screen)"

if Hum\_Per>=70.0 and Hum\_Per<90.0:

Forecast2 = Forecast2+"the Air is Very Moist (Use Normal Cream)"

This is a pretty straightforward apart of the program that based on the temperature and humidity makes a few assumptions on the weather and recommends the clothes to be worn and the type of cream or sun screen to be used.

The next part of the program is the web page which will be sent to the browser by the ESP-32 when evoked.

html\_page = """<!DOCTYPE html>

<html>

<head>

<style>

table {

width:100%;

}

table, th, td {

border: 1px solid black;

border-collapse: collapse;

border:ridge 5px red;

background-color:black; color:yellow;

}

th, td {

padding: 15px;

text-align: left;

}

TD{font-family: Calibri; font-size: 10pt;}

</style>

</head>

<meta http-equiv="refresh" content="5">

<body bgcolor="MediumSeaGreen">

<h1><center><font face="Calibri">Weather Station Ver 1.0</font></center></h1>

<table>

<tr>

<th><h2><center>Temperature in Deg C</center></h2></th>

<th><h2><center>Humidity in %</center></h2></th>

<th><h2><center>Forecast</center></h2></th>

</tr>

<tr>

<td><center>""" + str(Temp\_C) + """</center></td>

<td><center>""" + str(Hum\_Per) + """</center></td>

<td><center>""" + str(Forecast1) + """</center></td>

</tr>

</table>

<br>

<br>

<h1><center><font face="Calibri">Weather Forcast Details</font></center></h1>

""" + str(Forecast2) + """

<br><br>

</body>

</html>"""

return html\_page

This HTML file defines a table where the data will be displayed and will be returned when the ESP-32 IP address is evoked by the browser.

The web page is auto refreshed by the command below:

<meta http-equiv="refresh" content="5">

Where the content parameter is set to 5 which means a 5 second refresh.

The various variables like temperature and humidity from the DHT22 sensor are loaded to the web page by the HTML tags below:

<td><center>""" + str(Temp\_C) + """</center></td>

<td><center>""" + str(Hum\_Per) + """</center></td>

<td><center>""" + str(Forecast1) + """</center></td>

Where the DHT 22 values of **Temp\_C** , **Hum\_Per** and **Forecast1** are converted to strings before adding to the web page using + operators.

And finally, we come to the continuous loop that is executed by the ESP-32 controller and waits for any socket connections. When it receives any web page requests the web page that is stored in the ESP-32 is served up with all the variables like temperature, humidity and forecast.

while True:

**#SOCKET ACCEPT**

conn, addr = s.accept()

print("Got connection from %s" % str(addr))

**#SOCKET RECEIVE**

request=conn.recv(1024)

print("")

print("Content %s" % str(request))

**#SOCKET SEND**

request = str(request)

**#CREATE A SOCKET REPLY**

response = web\_page()

conn.send('HTTP/1.1 200 OK\n')

conn.send('Content-Type: text/html\n')

conn.send('Connection: close\n\n')

conn.sendall(response)

**#SOCKET CLOSE**

conn.close()

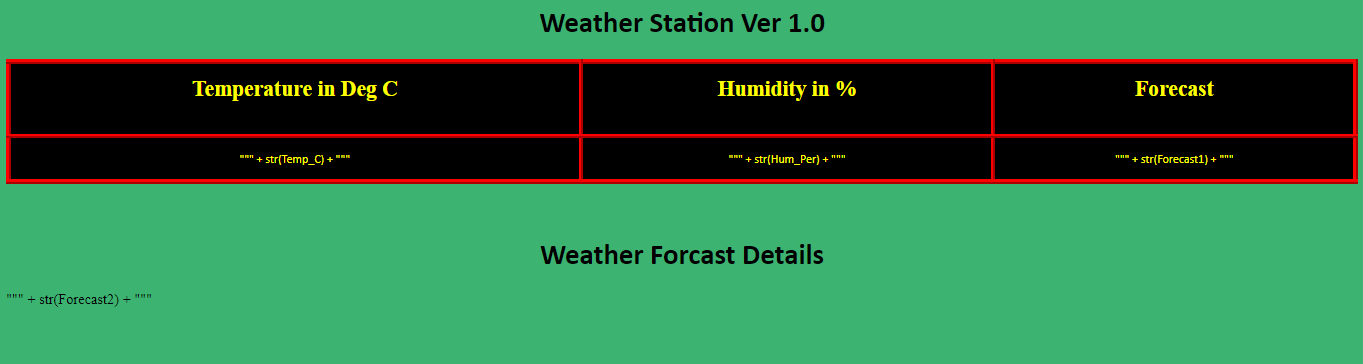
In this part of the program which is the continuous loop the system waits for any active connections and sends the response which in this case is the **web\_page** with updated variables and does this every 5 second with a refresh.

This is then completed by a conn.close() method to close the connection.

To sum up the various steps in the ESP-32 program written in Micro Python:

* Include all necessary libraries.
* Initialise all variables.
* Start up the Wifi connection and establish a Wifi connection.
* ESP-32 will be allotted an IP address by the Wifi Router which is needed to establish a socket connection.
* Start up and establish a Socket connection using the socket library.
* Move to a continuous loop and listen for browser request via the socket connection.
* If a request is received server up the web page with the updated temperature, humidity and forecast.

The screen shot of the Web Page is shown below:



**Fig 3.10: Web based Weather Station Web Page.**

The temperature, humidity and forecast are populated in the web page table and details are populated in the Web Forecast Details.

**vii Conclusion of Part-3**

This brings us to the end of section 3 where the ESP-32 is used to build a Weather Station that will display the temperature and humidity. Based on the values read from the DHT 22 sensor the system makes an estimate on the clothes to wear and the sun screen or cream to use. This is just an example of what can be done and can serve as a platform for inspirations for bigger projects.

The ESP-32 with Micro Python is a very powerful platform where many projects can be done. The ease in programming makes it a user-friendly platform.

The Thonny IDE platform used is by no means the only IDE for ESP-32. There are many IDEs that can be used and even Arduino can be used to program the ESP-32 (though not in Micro Python).

But Thonny is an ideal IDE for beginners who need to move fast and get on with some basic programming with the minimum of fuss.

We now move on to the last part of this book where the basics of IoT will be introduced. A word of caution-This is not a book on IoT and there are many books and resources on the web that introduce this vast subject. The advancement of microcontrollers has made the development of IoT rapid and development time shorter.

There are a large number of applications for IoT devices which make life easier and more fun and challenging for the developers.

**viii) Resources for Part-3**

**1. Components**

All components can be purchased from:

[www.amazon.in](http://www.amazon.in)

[www.in.element14.com](http://www.in.element14.com)

**2. Circuit Diagram/Program listing**

Git Hub:

https://github.com/SJayarajan/Embedded-Systems-Development-from-the-Ground-Up.git

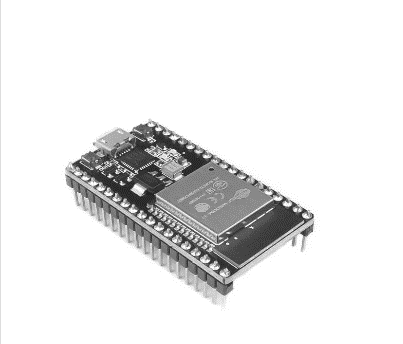
(Check files with Part-3 as Prefix in the file names)

**3. Further Readings**

For Python: https://www.python.org/

**4. Other Project Ideas**

* Solar Panel Controller.
* Data Logger



**Part-4**

**Introduction to IoT (Internet of Things)**

**i. Introduction**

With the advancement in the fields of microcontrollers and electronics it became very easy to incorporate a large amount of programmed content in a small package and this brought about the concept of interconnecting devices that will be intelligent, will talk to getter and make decisions.