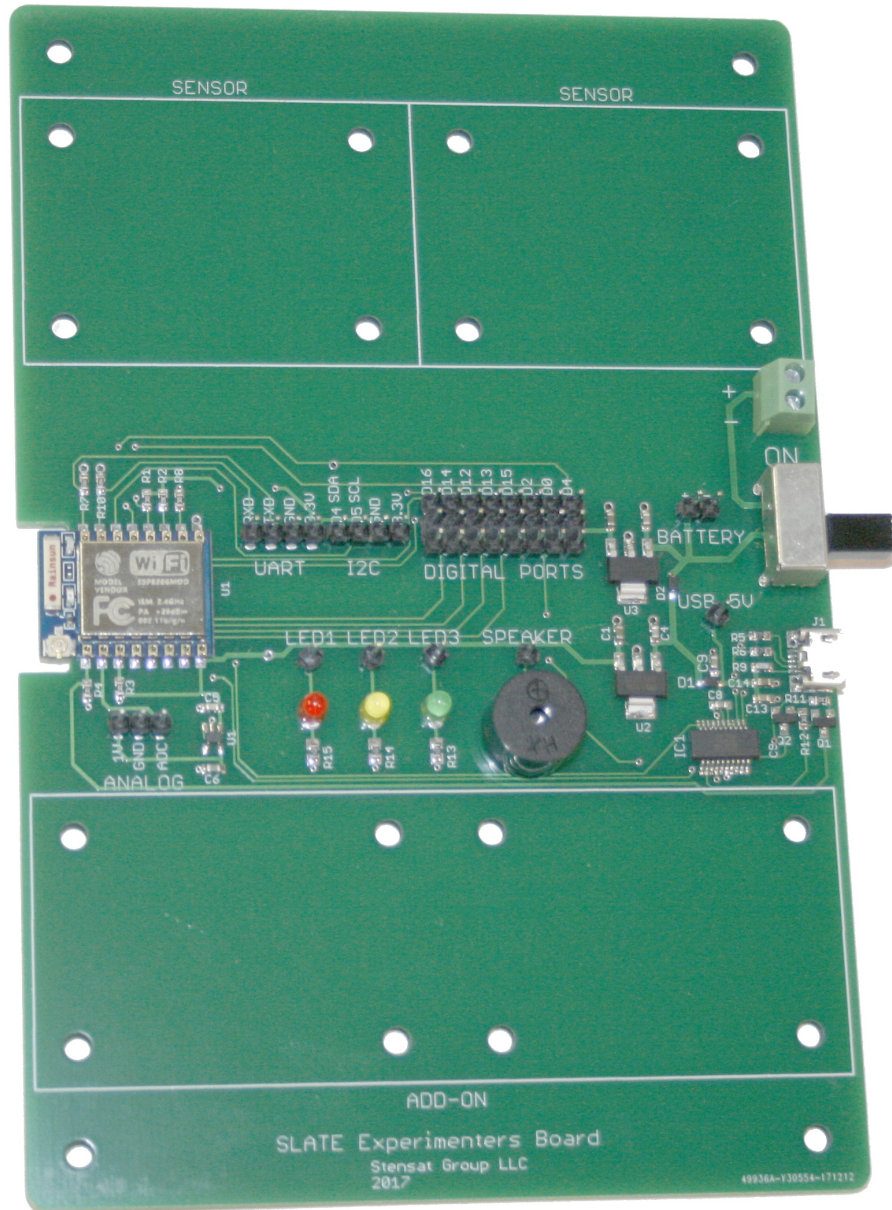


Sten-SLATE ESP Kit



Stensat Group LLC, Copyright 2020

Legal Stuff

Stensat Group LLC assumes no responsibility and/or liability for the use of the kit and documentation.

There is a 90 day warranty for the Sten-SLATE ESP kit against component defects. Damage caused by the user or owner is not covered.

Warranty does not cover such things as over tightening nuts on standoffs to the point of breaking off the standoff threads, breaking wires off the motors, causing shorts to damage components, powering the motor driver backwards, plugging the power input into an AC outlet, applying more than 9 volts to the power input, dropping the kit, kicking the kit, throwing the kit in fits of rage, unforeseen damage caused by the user/owner or any other method of destruction.

If you do cause damage, we can sell you replacement parts or you can get most replacement parts from online hardware distributors.

This document can be copied and printed and used by individuals who bought the kit, classroom use, summer camp use, and anywhere the kit is used. Stealing and using this document for profit is not allowed.

If you need to contact us, go to www.stensat.org and click on contact us.

Electrostatic Warning

The Sten-SLATE Experimenters Kit is sensitive to static electricity. Handle with care and be careful to discharge yourself before handling. Use the silver anti-static bag as an anti-static mat. Touch the bag before handling the processor to reduce the risk of damage.

Work on a flat surface, not the bed, a carpeted floor, other anywhere where static electricity can build up. The warranty does not cover damage due to electrostatic discharge. Fall and winter are times when electrostatic electricity is more prevalent.

Discharge yourself by touching a metal door frame, a metal lamp that is properly grounded, metal furniture, or some metal structure that is not connected to power.

Parts List

- Electronics Experimenters Base Board
- Sensor 1 Board with Ultrasound, light and temperature sensor
- Sensor 2 Board with accelerometer
- 20 F/F Jumpers
- 8 1/2" 4-40 screws
- 16 4-40 nuts
- 4 rubber pads
- Micro USB cable

Developer Kit

The Sten-SLATE ESP Experimenters Kit includes a 32-bit ARM processor with at least 512 Kbyte of program memory and 64 Kbyte of data memory. The processor operates at 80 MHz. WiFi is integrated into the processor.

The kit includes three LEDs and a speaker integrated. There are positions to install several sensor boards.

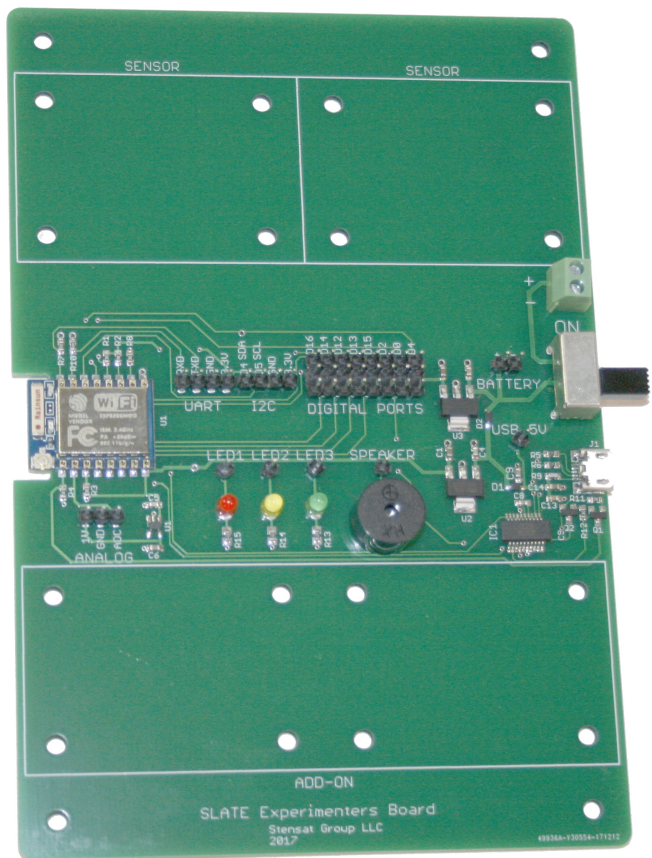
There are digital connections that can also operate up to eight servo motors.

There are two digital signals organized with power and ground to provide a convenient access to the I2C bus.

There are another two digital signals organized also with power and ground to provide convenient access to the UART interface. These interfaces will be explained later.

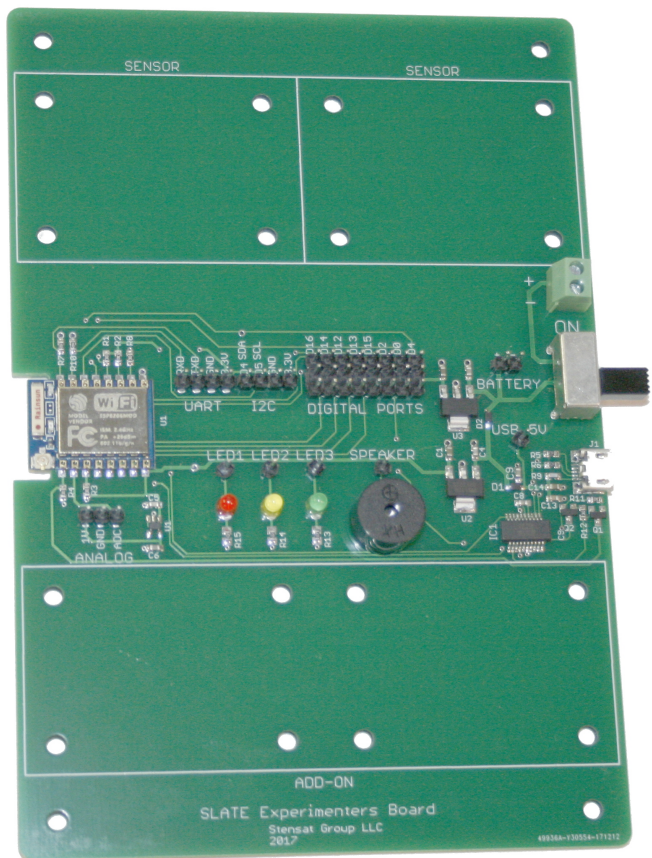
Power is applied through the USB port or through an external battery. For external power, a power switch is made available and a terminal screw block is provided to connect external power. Up to 12 volts can be applied.

The kit is designed to be modular so it can be more easily integrated.



Developer Kit

Take the four rubber pads and mount them into the four corners of the SLATE board. Make sure to not cover any holes.



Overview

In this section, you will be introduced to the processor board electronics and the arduino software.

At the end of this section, you will be able to write software, control things and sense the environment.

Arduino software location: www.arduino.cc

ESP8266 Arduino library information: <https://github.com/esp8266/Arduino>

Software

The SLATE ESP Experimenters Kit uses the arduino software. This software allows you to write programs, compile them and upload them to the processor. It also allows you to interact with the software running on the processor. Only one program can be installed and run at a time. The processor is small and does not have an operating system. Embedded computers are designed to perform a specific task and not operate like a desktop computer or laptop.

More information about the arduino software can be found at <http://www.arduino.cc>

Loading and Configuring Arduino Software

Download the proper version of the Arduino software from www.arduino.cc. Once downloaded and installed, start the program. Under the **File** menu, select **Preferences**. Find the text entry space to the right of **Additional Board Manager URLs**:

Enter the following into the text area

```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

Click OK.

Select the menu **Tools** and then **Boards:** and then **Boards Manager**.

A window will open. Scroll down until **esp8266** is located. Click on it and click Install.

The software will load the compiler for the processor board.

For Mac OSX users, you need to install python 3. Instructions are on page 64.

Configuring The Software

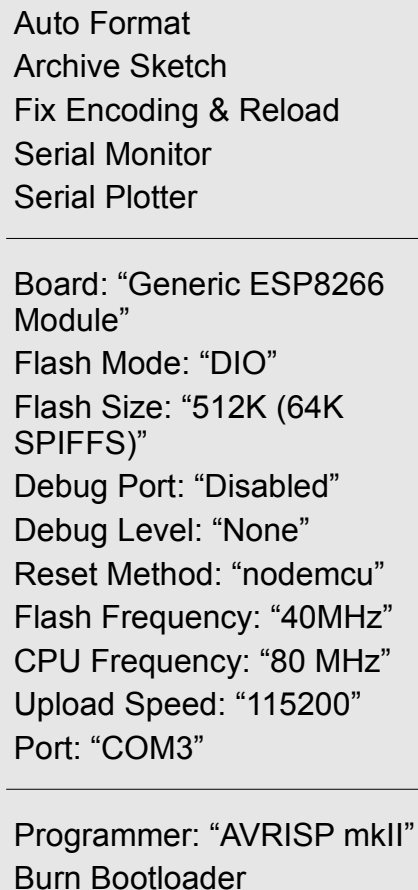
Select the “**Tools**” menu again and then “**Board:**”.

Select “**Generic ESP8266 Module**”

Go back to “**Tools**” menu and select “**Reset Method**”. Select “**nodemcu**”

Nothing else needs to be changed. This completes setting up for the processor board.

To the right is a representation of the Tools Menu. All selections should look like it. Only the Port: selection may be different. The menu may change as newer versions of the software become available. The same setting needs to be set.



Auto Format
Archive Sketch
Fix Encoding & Reload
Serial Monitor
Serial Plotter

Board: “Generic ESP8266 Module”
Flash Mode: “DIO”
Flash Size: “512K (64K SPIFFS)”
Debug Port: “Disabled”
Debug Level: “None”
Reset Method: “nodemcu”
Flash Frequency: “40MHz”
CPU Frequency: “80 MHz”
Upload Speed: “115200”
Port: “COM3”

Programmer: “AVRISP mkII”
Burn Bootloader

Selecting COM Port

The Arduino software communicates with the processor board for uploading code and interaction via a COM port with Windows and a /dev/tty.usbserial-Daxxxxxx with MAC OSX. Under Linux, the COM port is /dev/ttyUSBx where x is a number.

For Linux, the device driver is already part of the operating system.

MAC OSX should include a driver as part of the operating system.

The next two pages explain how to load the device driver for Windows and MAC OS X. Administrative privileges are required for installation. Under Windows, you must run the installation software as an administrator. Under MAC OS X, you will be asked for your login password.

Plug the Slate into the computers USB port.

Windows COM Port

Plug the SLATE into the USB port. If the Serial Port menu does not show any COM ports try the following: Go to <http://www.ftdichip.com/Drivers/VCP.htm>. In the table on the website, locate the row specifying **Windows**. Click on the link **setup executable** and download the software. Right click on the icon labeled **CDM v2.12.00.....** and select **Run as Administrator** in the menu that pops up. Follow the instructions to install. You have to run the installation program as an administrator or the driver will not install. Go back to Arduino and select the COM Port it gives you. Most likely, the COM port will be COM3. If there is more than one COM port, choose the higher number.

Macintosh OS X USB Driver Installation

Go to <http://www.ftdichip.com/Drivers/VCP.htm>. Select VCP driver (Virtual COM Port) for Mac 64 bit.

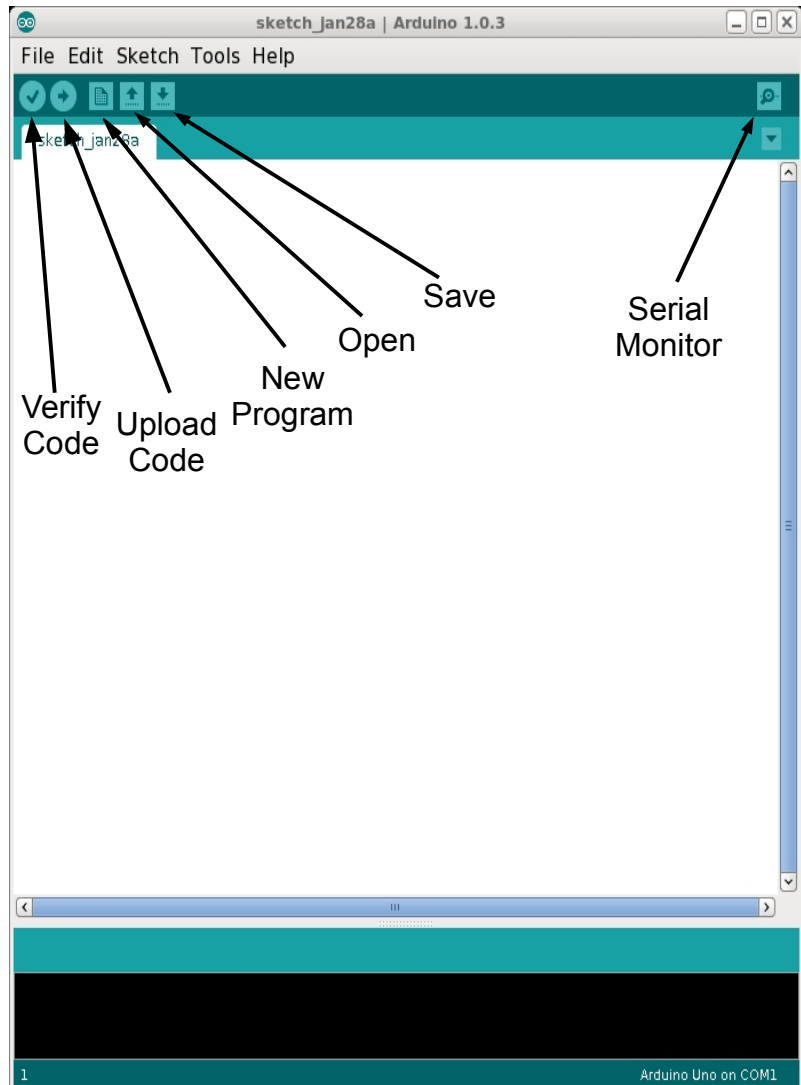
Download and double click on the file. A window will open showing two packages. Double click the package for the OS version you have. Follow instructions for the installation. You do need admin privileges to install. Go back to Arduino and select the Serial Port: /dev/tty.usbserial-Daxxxxxx The xxxxxx will be some combination of letters and numbers

Using Arduino

This is the arduino software. The software will let you enter programs and upload the code to the processor board. The large white area is where the code is entered. The black area below is where error messages will be displayed such as when there is an error in the code or the software cannot upload code for some reason.

The buttons below the menu have different functions. The first is called **Verify Code** and will compile the code and check for errors but not upload the code. The next button will do the same as the first but also upload the code. New Program button opens a new copy of the program allowing you to start writing another program. Open and Save are for opening and saving the code you have written.

Serial Monitor button opens a new window allowing you to interact with the processor. The Serial Monitor window allows the processor to display information and you to send information. This will be used quite a bit in this section.



First Program to Test

Enter the program in the editor on the right. **Do not copy and paste from the pdf file.** It doesn't work. The compiler is case sensitive so pay attention to capitalized letters. Don't forget to include a semi-colon after each statement as shown.

Plug the processor board into the USB port. Click on the upload Code button to compile and upload the program. When the status message at the bottom of the window says done uploading, click on the serial monitor button. The Serial Monitor window pops up with the message being displayed.

Experiment by changing the message. Save your program. Pick a file name.

```
void setup()
{
  Serial.begin(115200);
}

void loop()
{
  Serial.print("Hello World");
}
```



Serial Monitor Window

What are Functions

A function is basically a set of instructions grouped together. A function is created to perform a specific task.

The set of instructions for a function are bounded by the curly brackets as seen to the right.

The `setup()` function is used to initialize the processor board, variables, and devices.

Objects can include functions. **Serial** is an object with several functions. **begin()** in `Serial.begin()` is a function. **print()** is another function that is part of the **Serial** object.

You will notice that some lines end with a semi-colon. This is used to identify the end of an instruction. An instruction can be an equation or function call.

When you create a function such as `setup()`, you do not need a semi-colon. When you call a function then a semicolon is needed at the end of the function.

The program is made up of two functions. `setup()` function is run at reset, power up or after code upload only once. It is used to initialize all the needed interfaces and any parameters. `loop()` function is run after the `setup()` function and is repeatedly run hence the name loop.

This program configures the serial interface to send messages at 115200 bits per second.

The message is "Hello World" and is repeatedly displayed.

```
void setup()
{
  Serial.begin(115200);
}

void loop()
{
  Serial.println("Hello World");
}
```

`Serial.begin()` is a function that initializes the serial interface and sets the bit rate.

`Serial.println()` sends the specified message over the serial interface and move the cursor to down one line.

`delay(500)` is a command to stop the program for 500 milliseconds.

What is in the Software

In the `setup()` function, it executes the function `Serial.begin(115200)`; This function initializes the UART which is connected to the USB port to allow for communications.

In the `loop()` function, it executes the function `Serial.print("Hello world")`; This function send the text in quotes to the UART. This is displayed in the Serial Monitor window.

The other function is called `delay()`. This function stops the program for a specified period of time. The unit is in milliseconds. The code to the write displays the text every half second.

```
void setup()
{
  Serial.begin(115200);
}

void loop()
{
  Serial.println("Hello World");
  delay(500);
}
```

What are Bits and Bytes

Bits are a unit of information in the computer. It has two states:

State 0 is logic level 0 also known as low and represented in the processor as 0 volts. State 1 is a logic level 1 also known as high and represented in the processor as a greater than 2 volts.

Bytes are a group of 8 bits. This allows values greater than 1 to be processed. There is a weighting scheme so that numbers can be represented using a byte. The least significant bit has a weight of zero. 2 to the power of 0 is 1. The most significant bit has a weight of seven. 2 to the power of 7 is 128. Add up all the weights of the bits that are set to logic 1 to get the value of the byte.

Byte



Example:

10111001 equals $2^7 + 2^5 + 2^4 + 2^3 + 2^0 = 185$

$128 + 32 + 16 + 8 + 1 = 185$

Try a couple binary numbers

01100000

10010101

What are Bits and Bytes

- Example
 - 10111001 equals $2^7 + 2^5 + 2^4 + 2^3 + 2^0 = 185$
 - $128 + 32 + 16 + 8 + 1 = 185$
- Try a couple binary numbers
 - 01100000
 - 10010101

Byte



Controlling Hardware

Controlling an LED

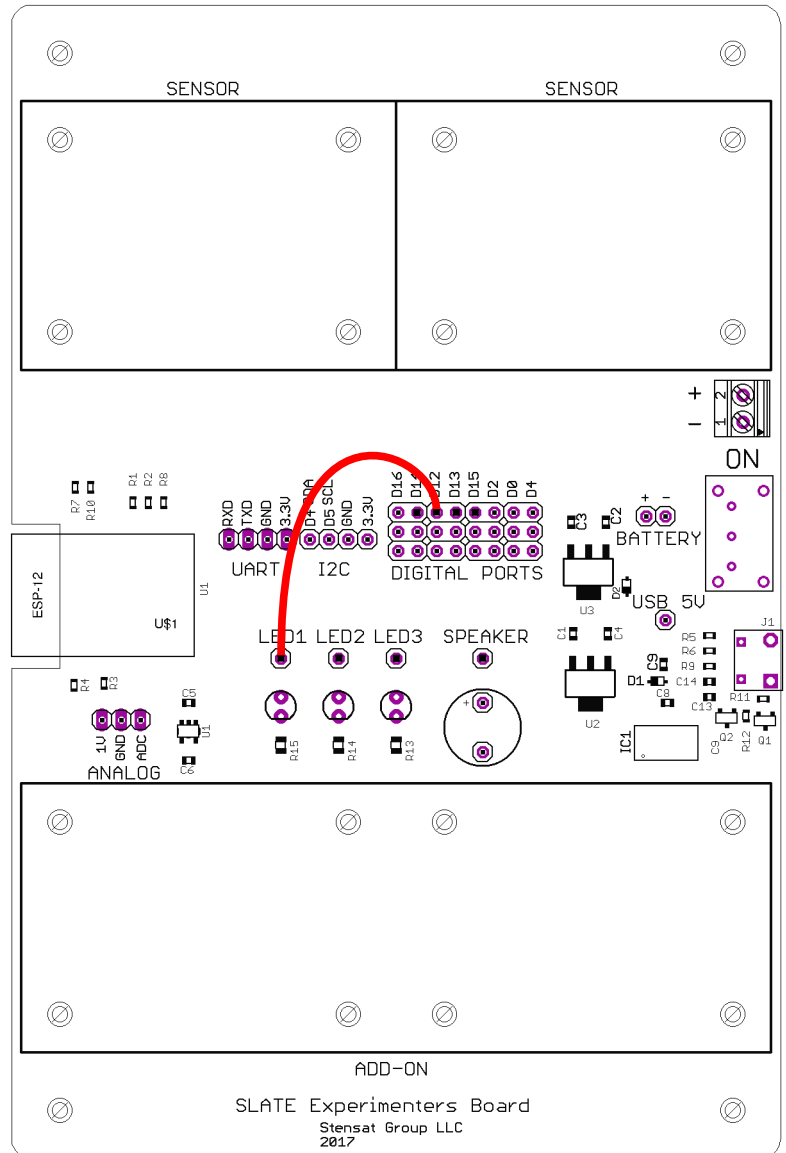
Take a jumper wire and insert one end into pin **D12**. Insert the other end into pin **LED1** as shown in the picture.

Open a new program and enter the code to the right. Click on the **Upload Code** button. You will be asked to save the program to a file. Pick a name like **blinky** and save. The software will then compile and upload the code.

The LED should start blinking.

```
void setup()
{
  pinMode(12, OUTPUT);
}

void loop()
{
  digitalWrite(12, HIGH);
  delay(500);
  digitalWrite(12, LOW);
  delay(500);
}
```



How the Code Works

In the `setup()` function, the function `pinMode(12,OUTPUT)` is used to configure the digital pin 12 to be an output so it can control the LED. You always need to configure the digital Pin to be an output when using it to control something like the LED.

In the `loop()` function, digital pin 12 is set high which causes the pin to generate 5 volts. The LED turns on. The `delay()` function halts the program for 500 milliseconds. The next `digitalwrite()` command sets digital pin 12 to 0 volts turning off the LED.

You can think of the digital pin as a light switch. It can be turned on and off. The computer language uses **HIGH** for the **on** state and **LOW** for the **off** state.

Experiment and change the delay settings to blink at different rates. How fast can you make the LED blink before you cannot see if blink?

What is an LED?

LED stands for Light Emitting Diode. It is a semiconductor device that generates light when an electric current passes through it. Current can only travel through the LED in one direction. In the wrong direction, the LED does not light up.

LEDs need the to current flowing through it limited otherwise it would pull a lot of current on its own and burn up. A resistor is used to reduce the current flowing through the LED. A resistor is a device that limits current flowing through a circuit. A circuit diagram to the right shows the LED connected to 5 volts.

The LED has an anode and a cathode. When the anode is at a higher voltage than the cathode, the LED will conduct current and light up. Do the reverse and the LED does not light up.

```
void setup()
{
    pinMode(12, OUTPUT);
}

void loop()
{
    digitalWrite(12, HIGH);
    delay(500);
    digitalWrite(12, LOW);
    delay(500);
}
```

Two LEDs

Using the same **blinky** program, you will add a second LED. Connect D13 to LED 2, the yellow LED. Modify your **blinky** program to include the extra code in bold shown to the right.

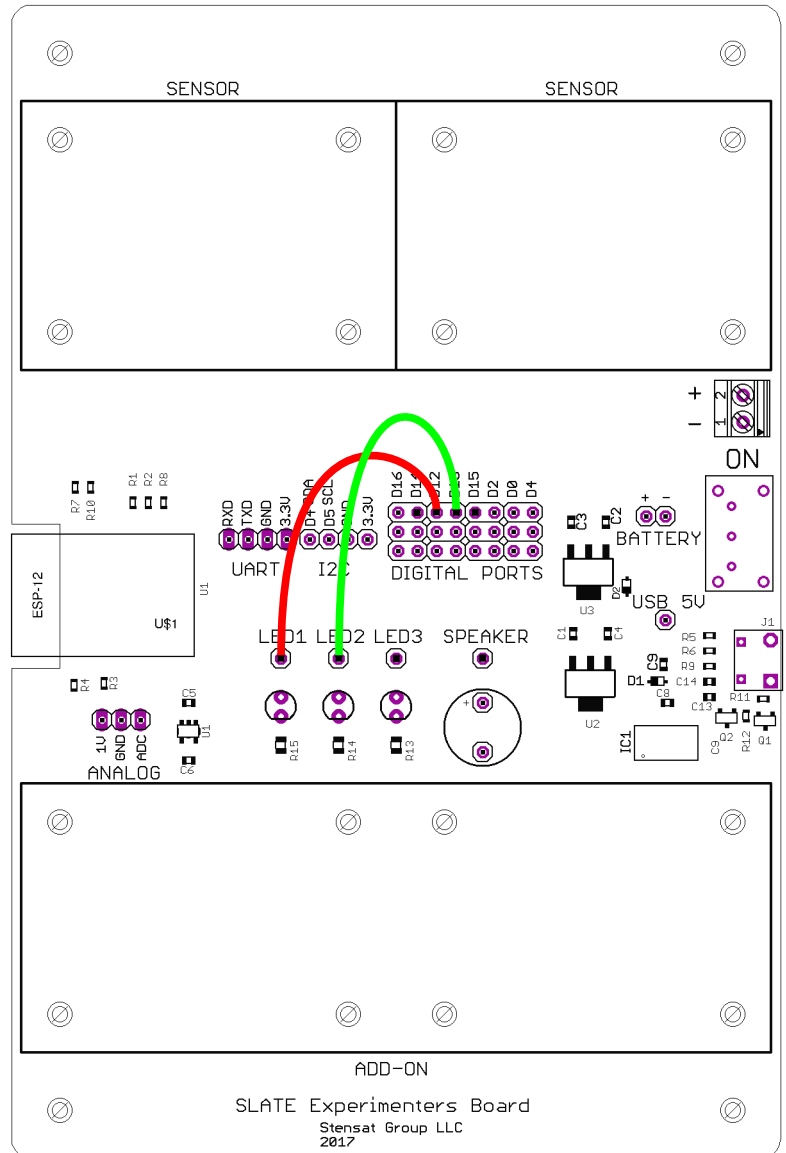
Click on the **Upload Code** button. Once the program uploads, you should have the red and yellow LEDs blinking back and forth.

On your own, add the third LED. Pick a digital pin and connect it to the green LED.

Change the program to blink each LED in sequence.

```
void setup()
{
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);
}

void loop()
{
  digitalWrite(12,HIGH);
  digitalWrite(13,LOW);
  delay(500);
  digitalWrite(12,LOW);
  digitalWrite(13,HIGH);
  delay(500);
}
```

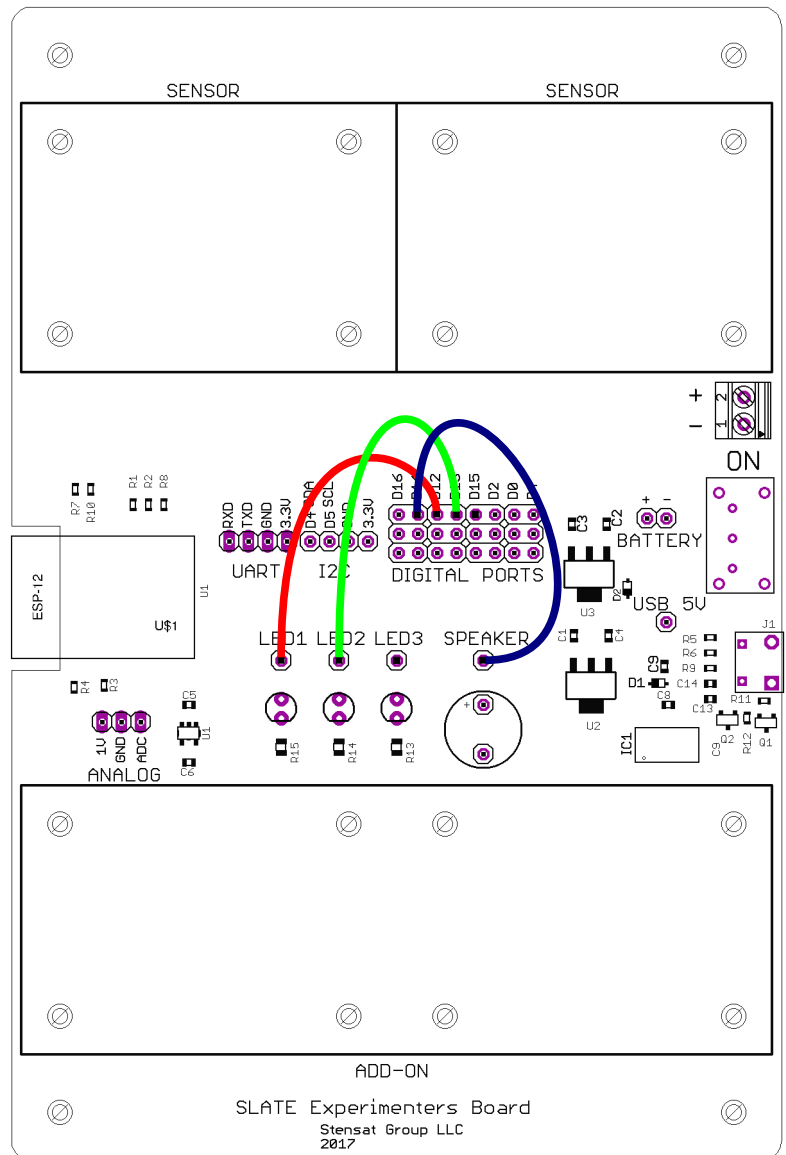


Making Sound

A speaker is a device that can make sounds. Connect digital pin **D14** to the **SPEAKER** pin. Start a new program and enter the code below. To make a sound, you use the **tone()** function. The tone function takes two numbers, the digital pin number and the tone frequency. Try the program below and change the tone frequency. Pick a number between 40 and 5000.

```
void setup()
{
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  pinMode(14, OUTPUT);
}

void loop()
{
  digitalWrite(12, HIGH);
  digitalWrite(13, LOW);
  tone(14, 2000);
  delay(500);
  digitalWrite(12, LOW);
  digitalWrite(13, HIGH);
  tone(14, 1500);
  delay(500);
}
```



Musical Notes

Musical notes have specific frequencies. Since the `tone()` function uses frequency to set the pitch, you will need to translate notes to frequencies.

A C note has a frequency of 261 Hz. A D note has a frequency of 294 Hz.

In the code shown, there are these **#define** statements that are used to assign a value to a capital letter. These are not variables so they do not take up space. These are constants. Instead of entering numbers for tones, you can just enter the musical note. It makes the code more understandable. Try out the program. Add more notes and different delays.

```
#define C 161
#define D 294
#define E 329
#define F 349
#define G 392
#define A 440
#define B 493
#define C2 523

void setup() {
  pinMode(14, OUTPUT);
}

void loop() {
  tone(14, C);
  delay(500);
  tone(14, C2);
  delay(500);
}
```

Play Music

For this program, you will learn about data arrays and **for()** loops. A data array is a group of variables with the same name but uses a number to select which variable is used. Variable **note** is an array. It is indicated by the square brackets after the name. This bracket is used to specify how many elements or variables are in the array. For this program, there are 7 elements. You also see that the elements are set with the notes in the curly brackets. There can be no more than 7 values put into the array.

There is also the **dur** array. This array is used to determine how long the note plays. 4 is for quarter note, 8 is for 1/8 note. A whole note is 1.

The **for()** loop lets your program loop through a specific number of time. There are three parts to the **for()** loop. The first part sets the starting condition. It declares a variable **i** and sets it to 0. The second part tests the value of **i** and if it is not less than **p** which is 7, the program exits the for loop. The last part increments **i** by 1 each time it loops. All code between the curly brackets after **for()** is executed until **i** is not less than **p**.

Notice the **noTone()** function. This turns off the tone being generated. A short delay is added. This allows same notes playing in a row to be distinct.

Try the program out and see how it plays. Try other music. You can change the size of the array. Remember to change the value of **p**.

```
#define C 161
#define D 294
#define E 329
#define F 349
#define G 392
#define A 440
#define B 493
#define C2 523
#define p 7 // number of notes

int note[p] = { C,C,G,G,A,A,G};
int dur[p] = {4,4,4,4,4,4,1};
void setup() {
  pinMode(14,OUTPUT);
}

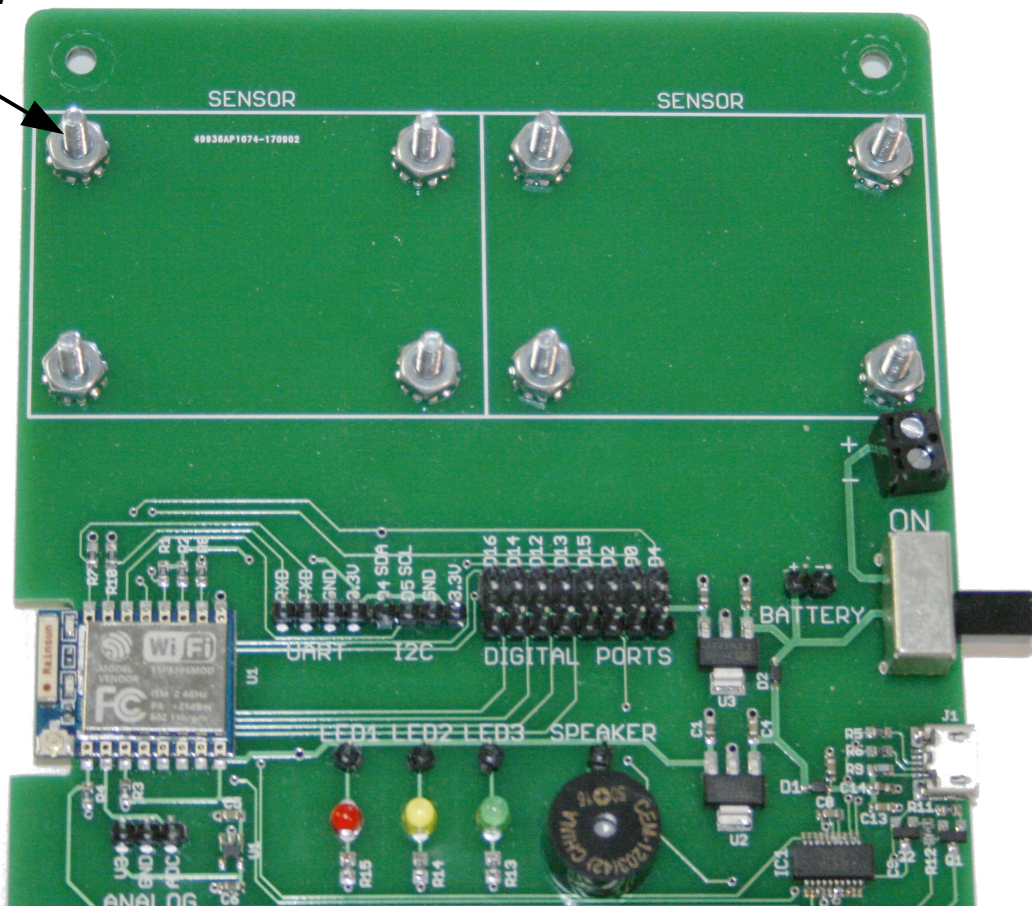
void loop() {
  for(int i=0;i<p;i=i+1) {
    int t = 1000/dur[i];
    tone(14,note[i]);
    delay(t);
    noTone(14);
    delay(20);
  }
}
```

Sensors

Mounting Sensors

There are two locations marked SENSOR for mounting sensor modules. They require screws and nuts to be installed for mounting sensor modules. Eight screws and nuts are required. Insert the screw through the hole from underneath the board and secure with a nut on the top side. Repeat for the remaining seven holes.

1/2 inch screw
and nut



Analog Sensors

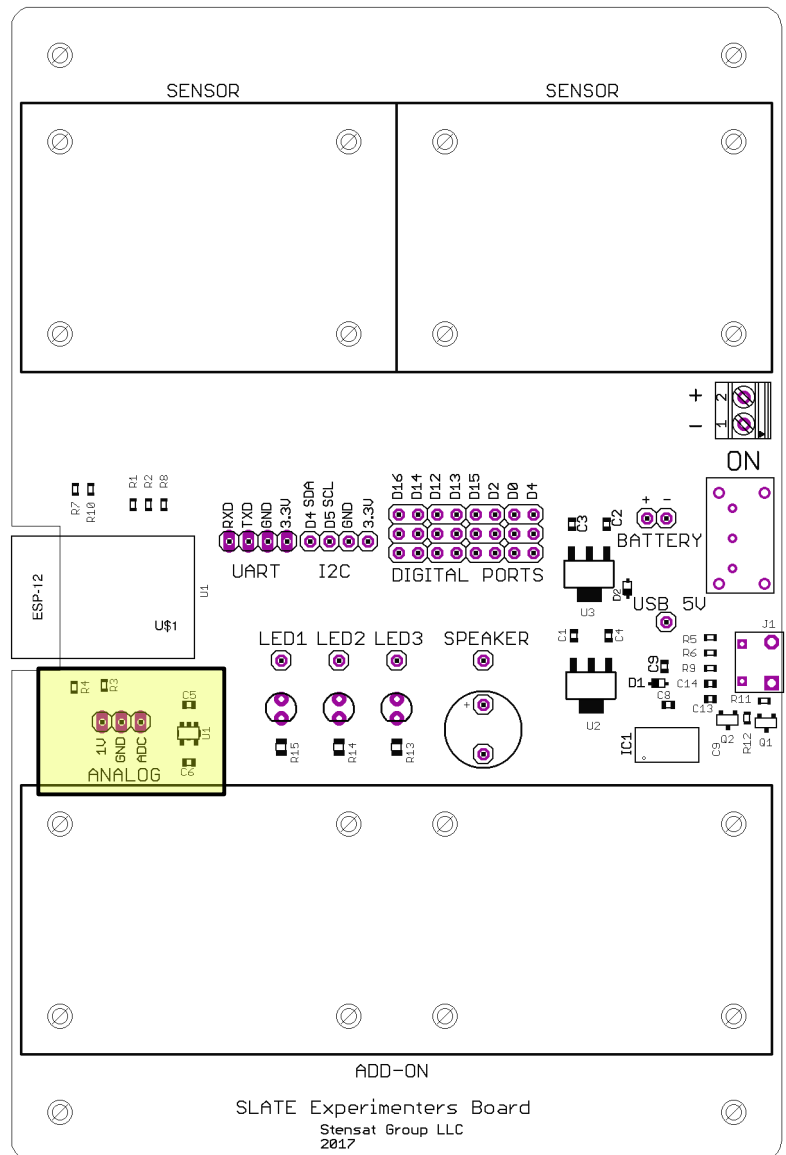
This section introduces analog sensors, a light sensor and a thermal sensor. These two sensors are variable resistors that change resistance based on what they measure.

The SLATE ESP has an analog-to-digital converter, ADC for short, that converts a voltage to a value. The SLATE ESP ADC has a voltage range of 0 to 1 volt. It is a 10 bit ADC which means it generates a number with a range from 0 to 1023. 0 volts results in a value of 0. 1 volt will generate a value of 1023. There is a linear relationship between the voltage and the value generated by the ADC. The voltage can be calculated using the equation below:

$$V = \text{ADC} / 1023.0$$

If 0.5 volts is applied to the ADC, the value generated will be 511..

The Experimenters board has one ADC input as highlighted in the image. A 1 volt reference is included at pin **V3** for any sensor circuit that needs it such as the light and thermal sensor. Pin **GND** is the 0 volt reference. Pin **ADC** is the input.

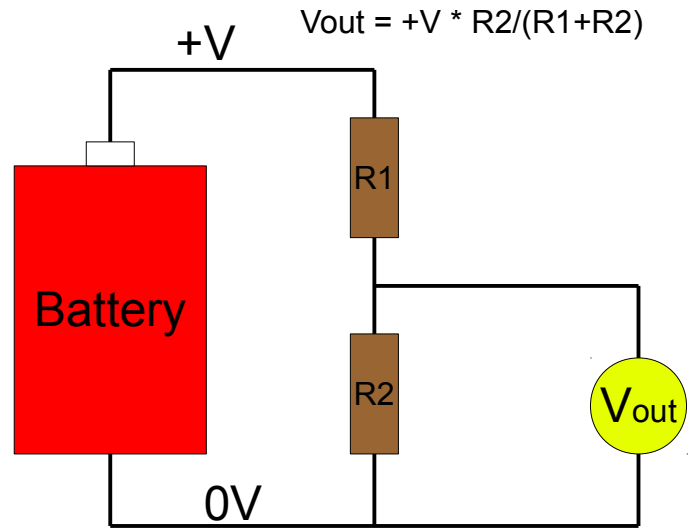


Analog Sensors

The light and thermal sensors are resistive type sensors. Their resistance changes based on what they measure. The ADC cannot measure resistance, only voltage. A voltage divider can generate a voltage based on the resistance of the sensor. The only problem is that the full range of the ADC may not be used since the resistor divider will not allow a full voltage swing from 0 to 1 volt.

The thermal sensor is called a thermistor. It's resistance changes with temperature. Their nominal resistance is specified at 25C. This particular sensor is at 10 Kohms at 25C. The table below shows the resistance at different temperatures.

The thermistor is resistor R1 as shown in the circuit. R2 is a 10 Kohm resistor. At 25C, the thermistor will be 10 Kohm. This divides the 1 volt to 0.5 volts. The ADC will generate 511. As the temperature goes higher, the resistance goes lower and the voltage to the ADC increases. As the temperature goes lower, the resistance increases and the voltage decreases.



Voltage Divider Circuit

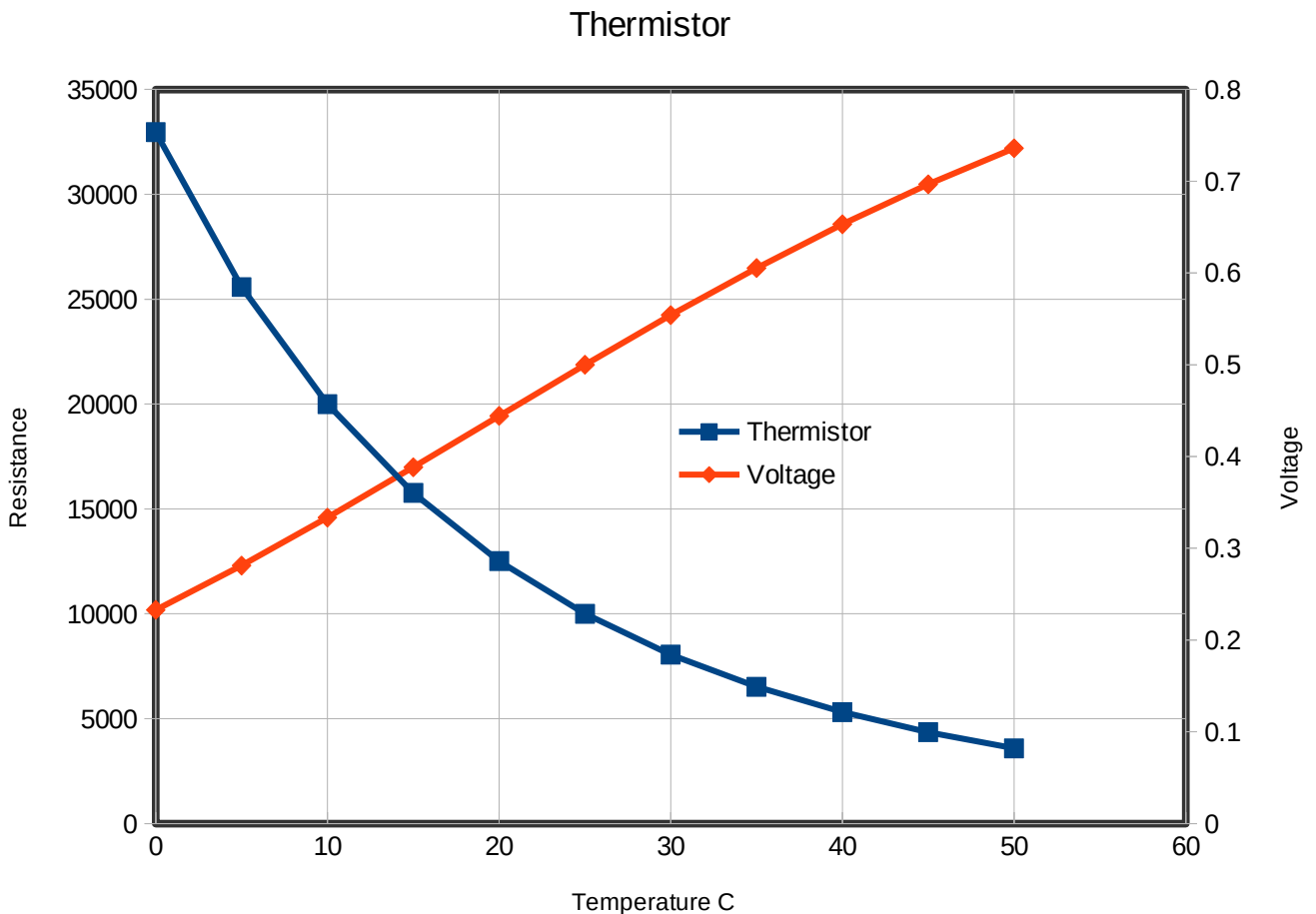
Temperature C	Resistance Ohms
0	32960
5	25580
10	20000
15	15760
20	12510
25	10000
30	8048
35	6518
40	5312
45	4354
50	3588

Thermistor Resistance Table

Thermistor

The plot of resistance versus temperature shows the thermistor does not have a linear response to temperature. The thermistor table can be entered into a spreadsheet and plotted using the scatter plot. The voltage can also be added and shown in the plot. This shows the relationship of voltage out of the voltage divider and resistance versus temperature.

The voltage plot almost looks linear and if a rough temperature is only needed, then a linear equation can be generated to calculate the temperature based on voltage.

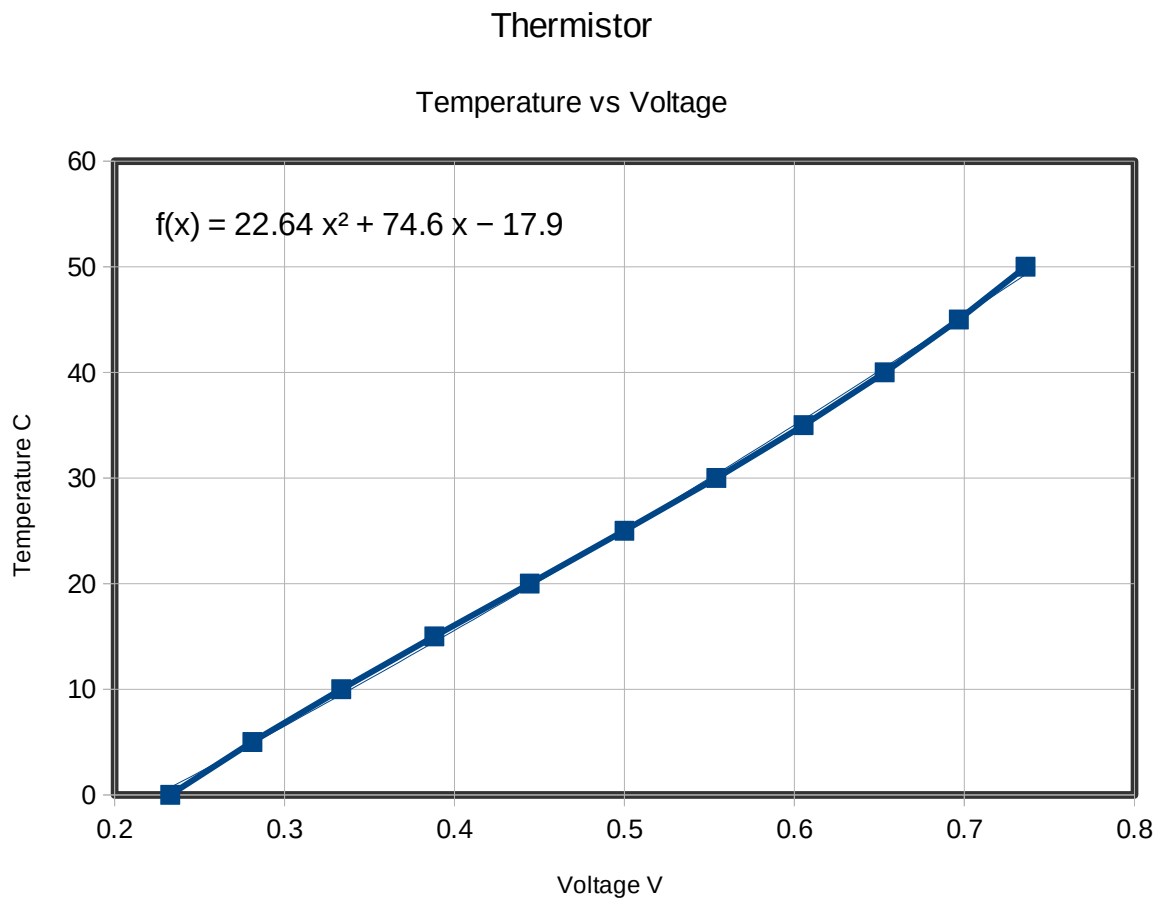


Thermistor

For better temperature accuracy, the voltage and temperature can be plotted in the spreadsheet with the temperature on the Y axis. The spreadsheet can generate a polynomial that can approximate the curve in the plot. Below is the equation generated.

$$\text{Temperature} = 22.64 * \text{voltage} * \text{voltage} + 74.6 * \text{voltage} - 17.9$$

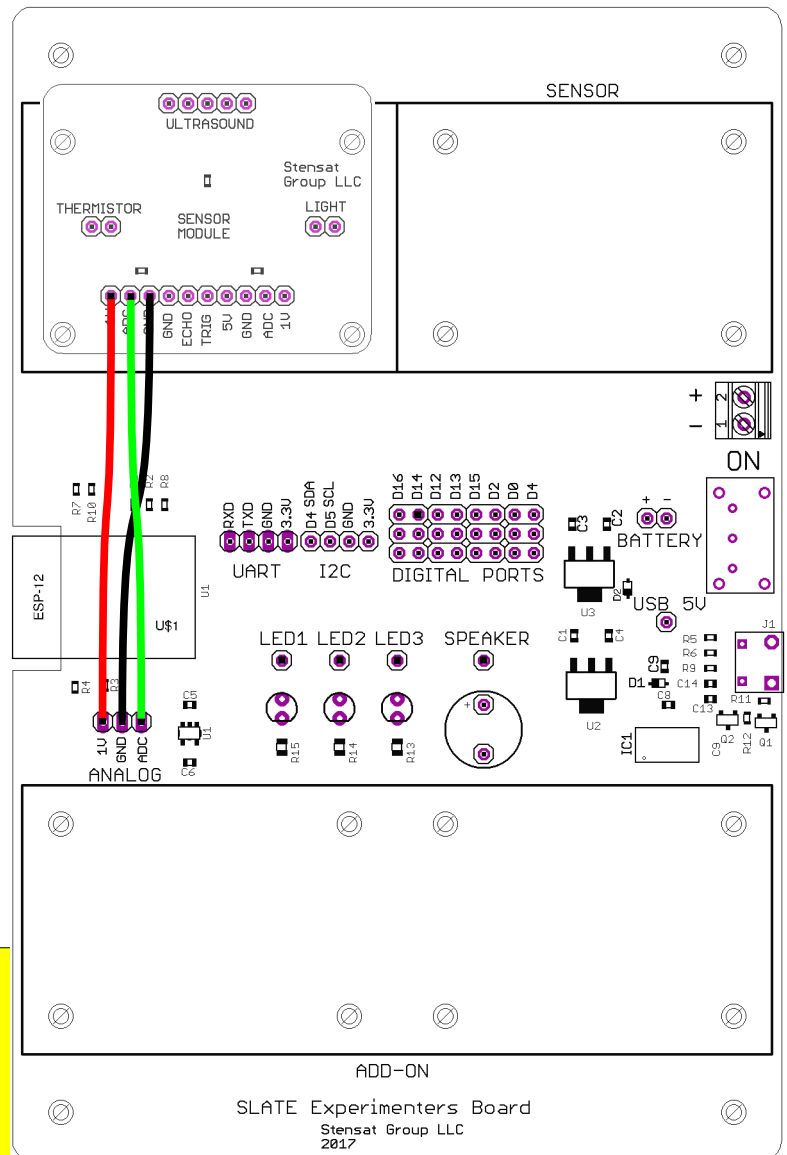
This equation will be used in the program on the next page.



Thermistor

Insert the Sensor module onto the screws in the left sensor area. Secure with #4 nuts. Use three jumper wires to connect the thermistor to the ADC.

Enter the code below and upload and run it. The result will be the air temperature. Place a finger on the sensor and see if the temperature rises.



```
void setup()
{
  Serial.begin(115200);
}

void loop()
{
  int a;
  float volts;
  a = analogRead(0);
  volts = (float)a/1023.0;
  float t = (22.64*volts*volts) + (74.6 * volts) - 17.9;
  Serial.println(t,2);
  delay(200);
}
```

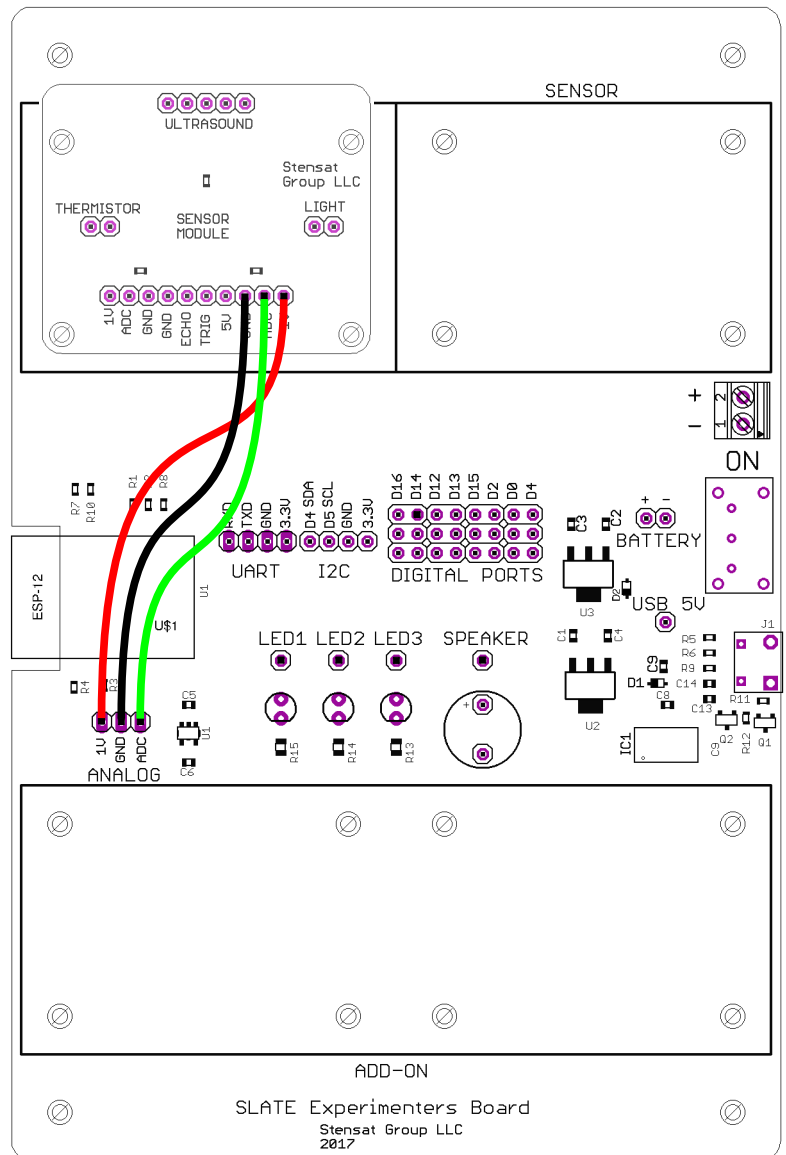
Light Sensor

The light sensor is a light sensitive resistor. It's resistance decreases as the light intensity increases. At 100 lux, the resistance is 5 Kohms. In the dark, the resistance can reach up to 20 megaohm. It is connected similarly as the thermistor taking the place of R1 in the voltage divider circuit. Voltage will increase as the light intensity increases and the voltage decreases as light intensity decreases. There is no calibration data for the sensor. It is used as general light intensity detector and generally used to detect darkness to turn lights on.

Enter the code in a new program and try it out. Use a flash light to shine light on the sensor and see how the voltage changes.

```
void setup()
{
  Serial.begin(115200);
}

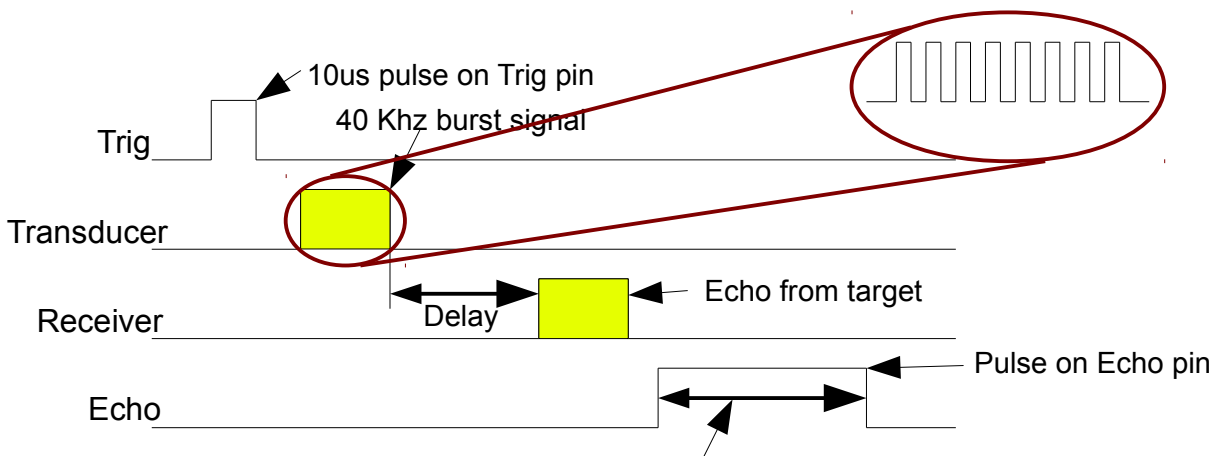
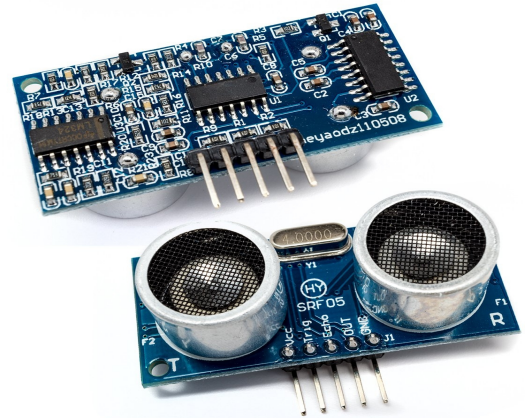
void loop()
{
  int a;
  float volts;
  a = analogRead(0);
  Serial.println(a);
  volts = (float)a/1023.0;
  Serial.println(volts,2);
  delay(200);
}
```



Ultrasonic Sensor

The ultrasonic sensor uses bursts of sound to measure distance. The sensor transmits a short 40 KHz tone and then measures the time it takes the tone to be reflected back.

The distance measurement is started by the Trig pin being pulsed high for at least 10 us. The sensor will then transmit a very short 40 KHz tone and wait for the echo to be detected. The sensor calculates the delay time and generates a pulse with the width proportional to the delay. Distance can be calculated by measuring the pulse width in microseconds and dividing by 58 to get centimeter values.



$$\text{distance} = \text{pulse width (us)} / 58$$

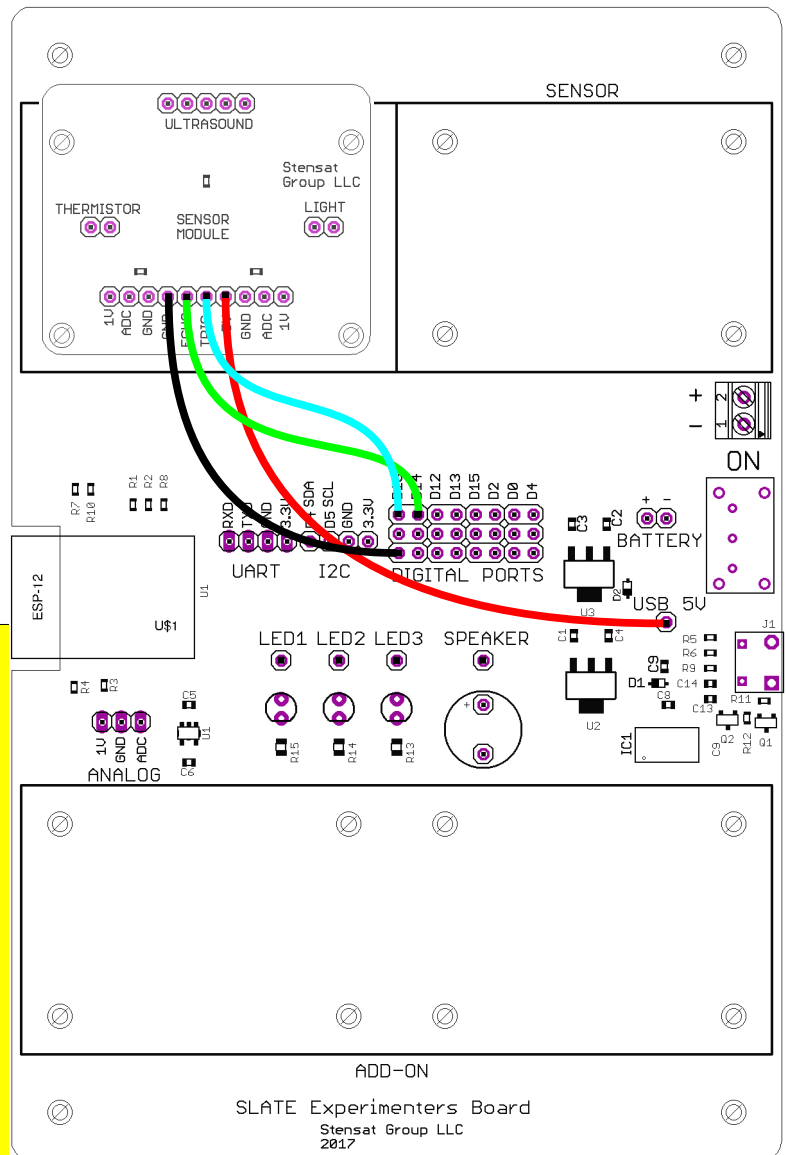
Ultrasonic Range Sensor

The picture shows where to install the jumper wires to connect the ultrasonic range sensor. **GND** is connected to the **GND** pin on the digital port. **5V** is connected to the USB 5V pin. Trigger is connected to **D16**. Echo is connected to **D14**.

Enter the code below in a new program and run it. Use a solid object or hand and move it to and from the sensor. A ruler can be used to verify the accuracy of the sensor. It can measure down to 3 centimeters.

```
void setup()
{
  Serial.begin(115200);
  pinMode(14, INPUT);
  pinMode(16, OUTPUT);
}

void loop()
{
  unsigned long distance;
  digitalWrite(16, LOW);
  delayMicroseconds(2);
  digitalWrite(16, HIGH);
  delayMicroseconds(10);
  digitalWrite(16, LOW);
  distance = pulseIn(14, HIGH);
  distance = distance/58;
  Serial.println(distance);
  delay(500);
}
```



Creating a Function

Using the ultrasonic range sensor program, it will be modified to become a standalone function. A function is a group of instructions with a name that can be called from a program. Functions are useful for where there is an operation that is used in multiple places in a program. This helps eliminate the need to rewrite the same code in different parts of the program. It also allows the function to be used in different programs.

Select **Save As...** under the **File** menu. Give the program the name **ultrasound_f**. Next delete the function **setup()** that is highlighted. Also delete the last two function calls at the end of the **loop()** function.

As shown in the lower right, change the **void loop()** to the name of the function. Insert the return command at the end of the new function.

Select **Save** from the **File** menu.

```
void setup()
{
  Serial.begin(115200);
  pinMode(14, INPUT);
  pinMode(16, OUTPUT);
}

void loop()
{
  unsigned long distance;
  digitalWrite(16, LOW);
  delayMicroseconds(2);
  digitalWrite(16, HIGH);
  delayMicroseconds(10);
  digitalWrite(16, LOW);
  distance = pulseIn(14, HIGH);
  distance = distance/58;
  Serial.println(distance);
  delay(500);
}
```

```
unsigned long ultrasound()
{
  unsigned long distance;
  digitalWrite(16, LOW);
  delayMicroseconds(2);
  digitalWrite(16, HIGH);
  delayMicroseconds(10);
  digitalWrite(16, LOW);
  distance = pulseIn(14, HIGH);
  distance = distance/58;
  return(distance);
}
```

Creating a Separate Function File

Create a new program. Enter the program shown. Save it with the name ranging. You will notice the tab for the program is now named ranging.

Under the Sketch menu, select Add File... Locate and select ultrasound_f file. You have to go to the ultrasound_f folder. A new tab will appear with the ultrasound function. Compile and upload it.

```
void setup()
{
    Serial.begin(115200);
    pinMode(14, INPUT);
    pinMode(16, OUTPUT);
}

void loop()
{
    unsigned long distance;
    distance = ultrasound();
    Serial.println(distance/58);
    delay(500);
}
```



Conditional Programming

Conditional Programming

Conditional programming is how programs make decisions. The simplest form is the **if()** statement. The **if()** statement requires an argument. The argument is a comparison that results as true or false. False has a value of zero.

```
if(a == b) {  
    // execute if true  
}  
else {  
    // execute if false  
}
```

The following are valid comparisons

a == b true if a equals b

a > b true if a greater than b

a >= b true if a greater or equal to b

a < b true if a less than b

a <= b true if a less than or equal to b

Conditional Programming

In this example, the program will monitor the light level and turn on the LED when it gets dark enough.

Reuse the wiring of the light sensor as before. Add a jumper to connect **D14** to **LED1**. Enter the code below and run it. The voltage threshold may need to be adjusted depending on the light level in the room. Adjust it so that the LED is off with the ambient light in the room. Then cover the light sensor. The LED should turn on.

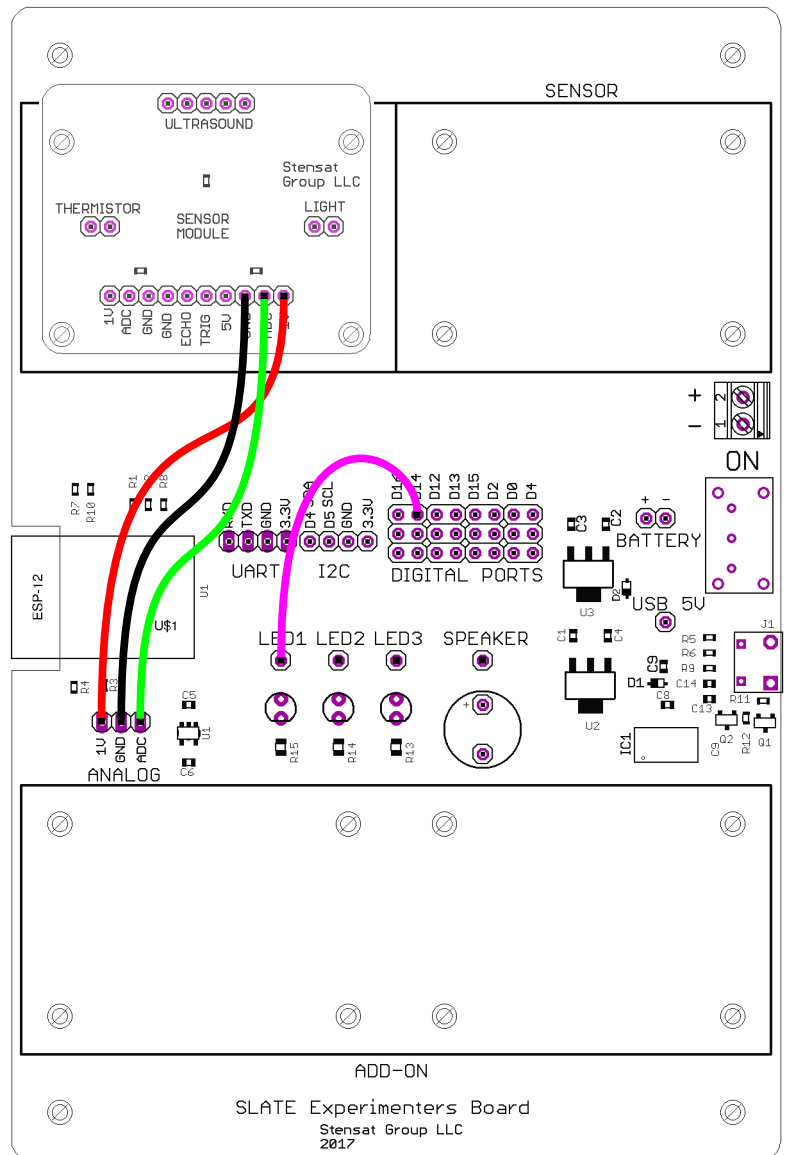
At the top of the code is a **#define** statement. This assigns a constant to a name. This is useful if the constant is used in multiple places. It allows easier changes by only changing the constant in one location. It also helps to use a name that has meaning for its use.

Adjust the **THRESHOLD** constant until the program operates properly.

```
#define THRESHOLD 500

void setup()
{
  Serial.begin(115200);
  pinMode(14, OUTPUT);
}

void loop()
{
  int a;
  a = analogRead(0);
  Serial.println(a);
  if(a < THRESHOLD) {
    digitalWrite(14, HIGH);
  } else {
    digitalWrite(14, LOW);
  }
  delay(200);
}
```



Conditional Test

Reconnect the thermistor to the ADC. Connect all three LEDs to digital pins. Create a program to do the following:

Turn on the green LED when at room temperature

Turn on the yellow LED when it is a bit warmer. Select the threshold.

Turn on the red LED when it is hot. Select the threshold.

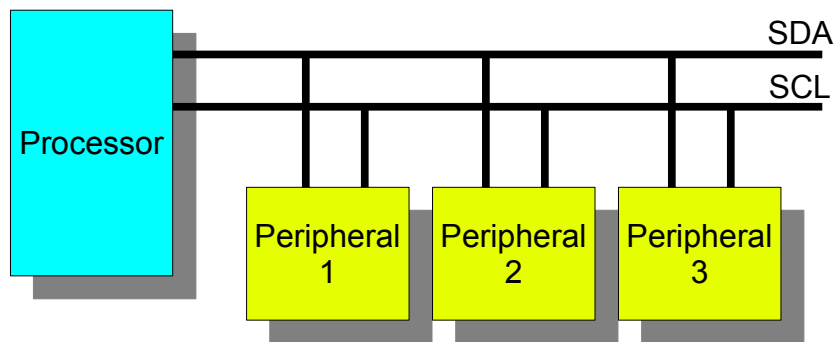
Turn off the red LED when it is below the hot threshold.

Turn off the yellow LED when it is below the warm threshold.

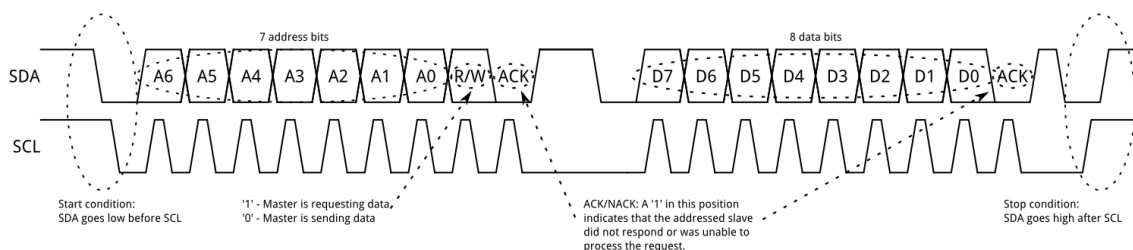
I2C Bus

I2C Bus

I2C stands for Inter-Integrated Circuit. It is a serial type interface requiring only two signals, a clock signal and a data signal. The I2C bus is typically used to interface with sensors and peripheral devices not needing to communicate at high speeds. The standard data rate is 100 Kilobits per second. Multiple devices can be connected to a single I2C bus. The processor is the controller and all the connected devices are peripherals. The processor is also called the master and the peripherals are slaves. Each slave has a unique address.



The clock signal is labeled SCL. This signal is used to control the flow of the data bits. The data signal is called SDA. This carries the data serially. The diagram below shows how a data transfer occurs. The data transfer protocol is for the master to first send out a device address. This is a 7 bit number followed by a bit indicating if the next byte is to be written to a slave or read from a slave. The SCL signal toggles for each bit sent.

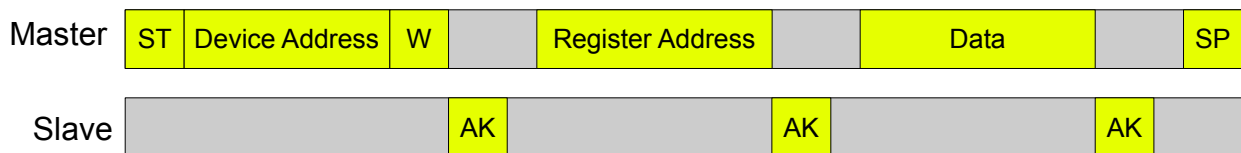


I2C Sequence

Every device on the I2C bus has a unique 7-bit address. The accelerometer address is 0x1C. The I2C operation for writing to a register is:

1. Send Start sequence by keeping SCL high and changing SDA from high to low (ST)
2. Send the device address
3. Send the register address
4. Send the stop sequence by changing SDA from low to high while SCL is high first. (SP)

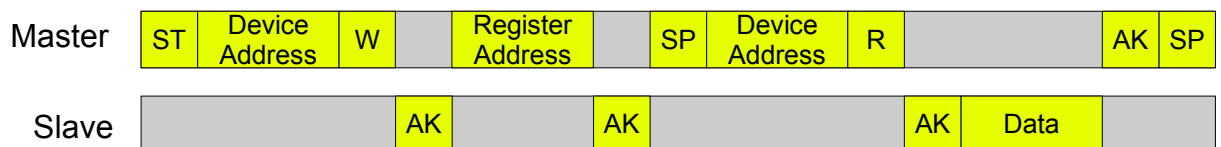
All Bytes sent are acknowledge by the slave.



Every device on the I2C bus has a unique 7-bit address. The accelerometer address is 0x1C. The I2C operation for writing to a register is:

1. Send Start sequence by keeping SCL high and changing SDA from high to low (ST)
2. Send the device address
3. Send the register address
4. Send the stop sequence by changing SDA from low to high while SCL is high first. (SP)

All Bytes sent are acknowledge by the slave.



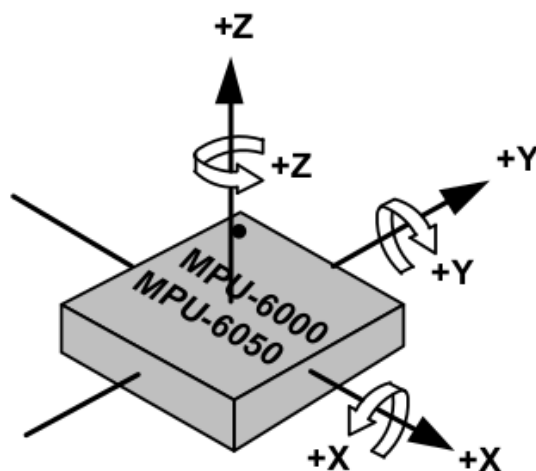
The MPU-6050 is a six degree of freedom Inertial Measurement Unit. It consists of a 3-axis accelerometer and a 3-axis rate gyroscope. The accelerometer has a settable sensitivity range of 2Gs to 16 Gs. The rate gyroscope has a settable sensitivity range of 250 degrees per second to 5000 degrees per second.

The sensor allows you to measure acceleration and rotation in three dimensions. With the combination of the the accelerometers and rate gyros, orientation can be measured in the three axis. In the X and Y axis, the absolute orientation relative to earth's gravity can be measured. The sensor will measure a range of -180 to +180 degrees.

In the Z axis, the accelerometer cannot be used so there is no absolute reference for the orientation around the Z axis. The zero angle orientation around the Z axis is established when the sensor is calibrated. As the sensor is rotated, the angle measurement accumulates. The measurement will not reset when exceeding -360 or +360 degrees. The number of rotations can be calculated by dividing the measured value by 360.

All gyros have drift which needs to be calibrated out. The programs later include a calibration mode. When the program starts, the sensor has to not move at all. The calibration will take about three seconds and will measure the gyro drifts so they can be subtracted from the measurements.

In this lesson, the rate gyro will be used to determine orientation. The library includes functions for calculating the angle of the sensor after it is calibrated. A processing program will be used to graphically demonstrate the orientation of the sensor.



Sensor Axis Diagram

Install the sensor board into the second sensor location as shown. Secure it with nuts. Connect the jumpers as shown. Connect from the pins above the I2C label on the experimenters board.

Connect 3.3V to V+. Connect both GND signals. Connect D5 to SCL. Connect D4 to SDA.

Download the library from www.stenat.org www.stensat.org. This library was modified to work properly on the ESP8266 SLATE board. In the Arduino IDE, select the **Sketch** menu and then select **Include Library**. Select **Add .ZIP file**. Locate the file and select it. It will be added to your library. In the **File** menu, select **Examples** then locate **MPU6050_tockn**. Select **GetAllData**. The program will open in a window.

Before compiling, make the following changes as shown in the next page:

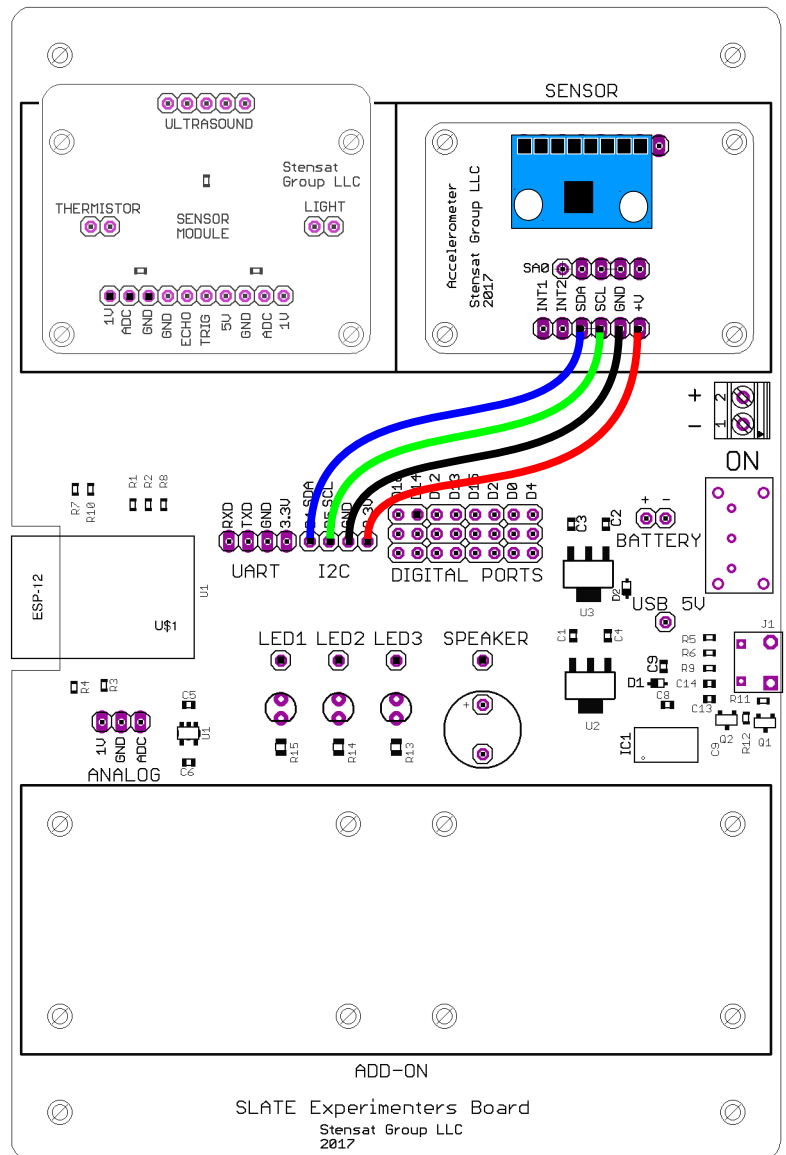
Line 11, **Wire.begin(); => Wire.begin(4,5);**

Line 12, **mpu6050.begin(); ==> mpu6050.begin(ACCEL_2G,GYRO_500);**

Change line 10 baud rate from 9600 to 115200.

Compile and upload the program.

When done uploading, open the Serial monitor and make sure the baud rate is set to 115200. When the program starts, it will spend about 3 seconds calibrating. Make sure the sensor is not moving during this time. It will calibrate right after the code finishes uploading.



GetAllData Program

```
#include <MPU6050_tockn.h>
#include <Wire.h>

MPU6050 mpu6050(Wire);

long timer = 0;

void setup() {
  Serial.begin(115200);
  Wire.begin(4,5);
  mpu6050.begin(ACCEL_2G,GYRO_500);
  mpu6050.calcGyroOffsets(true);
}

void loop() {
  mpu6050.update();

  if(millis() - timer > 1000){

    Serial.println("=====");
    Serial.print("temp : ");Serial.println(mpu6050.getTemp());
    Serial.print("accX : ");Serial.print(mpu6050.getAccX());
    Serial.print("\taccY : ");Serial.print(mpu6050.getAccY());
    Serial.print("\taccZ : ");Serial.println(mpu6050.getAccZ());

    Serial.print("gyroX : ");Serial.print(mpu6050.getGyroX());
    Serial.print("\tgyroY : ");Serial.print(mpu6050.getGyroY());
    Serial.print("\tgyroZ : ");Serial.println(mpu6050.getGyroZ());

    Serial.print("accAngleX : ");Serial.print(mpu6050.getAccAngleX());
    Serial.print("\taccAngleY : ");Serial.println(mpu6050.getAccAngleY());

    Serial.print("gyroAngleX : ");Serial.print(mpu6050.getGyroAngleX());
    Serial.print("\tgyroAngleY : ");Serial.print(mpu6050.getGyroAngleY());
    Serial.print("\tgyroAngleZ : ");Serial.println(mpu6050.getGyroAngleZ());

    Serial.print("angleX : ");Serial.print(mpu6050.getAngleX());
    Serial.print("\tangleY : ");Serial.print(mpu6050.getAngleY());
    Serial.print("\tangleZ : ");Serial.println(mpu6050.getAngleZ());
    Serial.println("=====\n");
    timer = millis();

  }
}
```


In the example program, the sensor library is included at line 2. Line 3 loads the I2C library. Line 5 creates a sensor object. The argument is Wire which tells the library to use the I2C interface. This is done to allow multiple I2C buses to be used. Only one is used here.

The timer variable in line 7 is used to track the time and have the display updated sensor data once a second. Lines 9-14 is the setup function. The serial interface is configured then the I2C interface. Next the sensor is configured with the accelerometer set to 2G range and the gyro set to 500 degrees per second rotation rate range. Line 13 calls a library function to calibrate the gyroscope. The gyroscope has what is called a DC offset or constant offset. This is an error that all sensors have and can be measured with the sensor not moving. The library subtracts the offset from all measurements.

Lines 16 – 46 is the loop function. Line 19 determines if a second has passed. If so, the reset of the code is executed. Line 17 is the function that collects the sensor data. The results are kept in the library variables. Lines 22-40 display the sensor results Notice that the values displayed are function calls. **mpu6050.getTemp()** will return the temperature in Celsius.

mpu6050.getAccx() will return the X-axis accelerometer value in Gs and so on. Notice the values are in floating point and processed from the raw values. Lines 31 and 32 return the sensor angle in the X and Y axis based on the accelerometer.

mpu6050.getAccAngleX() returns an angle in degrees referenced to the Z and X axis.

mpu6050.getAccAngleY() returns the angle in degrees referenced to the Z and Y axis.

mpu6050.getGyroAngleX() returns the angle calculated by the accumulation of the rate gyro around the X axis.

mpu6050.getGyroAngleY() returns the angle calculated by the accumulation of the rate gyro around the Y axis.

mpu6050.getGyroAngleZ() returns the angle calculated by the accumulation of the rate gyro around the Z axis.

mpu6050.getAngleX() provides the angle around the X axis based on the combination of the accelerometer and gyro.

mpu6050.getAngleY() provides the angle around the Y axis based on the combination of the accelerometer and gyro.

mpu6050.getangleZ() provides the angle around the Z axis based on the combination of the accelerometer and gyro.

These three functions provide the best orientation value of the sensor and can be used to indicate the orientation of any device it is connected.

Simpler IMU Program

This program is a simpler version of the example program where only the X,Y,Z angles are sent over the USB port.

Enter this program in the Arduino IDE and upload to the SLATE. Name the program **simpleimu**.

Instead of opening the Serial Monitor, select the menu **Tools** and select **Serial Plotter**. A window will open and plot three lines as data is streaming.

Rotate the board on the table and see how the Z-axis data changes. Rotate more than 360 degrees. Notice the plot goes out of range. Rotate in the opposite direction. The Z-axis plot should return back into the plot area.

Do the same for the other two axis. Notice they only go to +/- 180 degrees. The accelerometer is being used with the gyro to maintain orientation information. The accelerometers are using gravity as a reference to down. For the Z-axis, there is no reference. A magnetometer could be used as a reference for the Z-axis using the earth's magnetic field.

Save this program. It will be used later.

```
#include <MPU6050_tockn.h>
#include <Wire.h>

MPU6050 mpu6050(Wire);
long timer = 0;

void setup() {
  Serial.begin(115200);
  Wire.begin(4,5);
  mpu6050.begin(ACCEL_2G,GYRO_500);
  mpu6050.calcGyroOffsets(true);
}

void loop() {
  mpu6050.update();
  Serial.print(mpu6050.getAngleX());
  Serial.print(",");
  Serial.print(mpu6050.getAngleY());
  Serial.print(",");
  Serial.println(mpu6050.getAngleZ());
  delay(10);
}
```

Matlab Interface

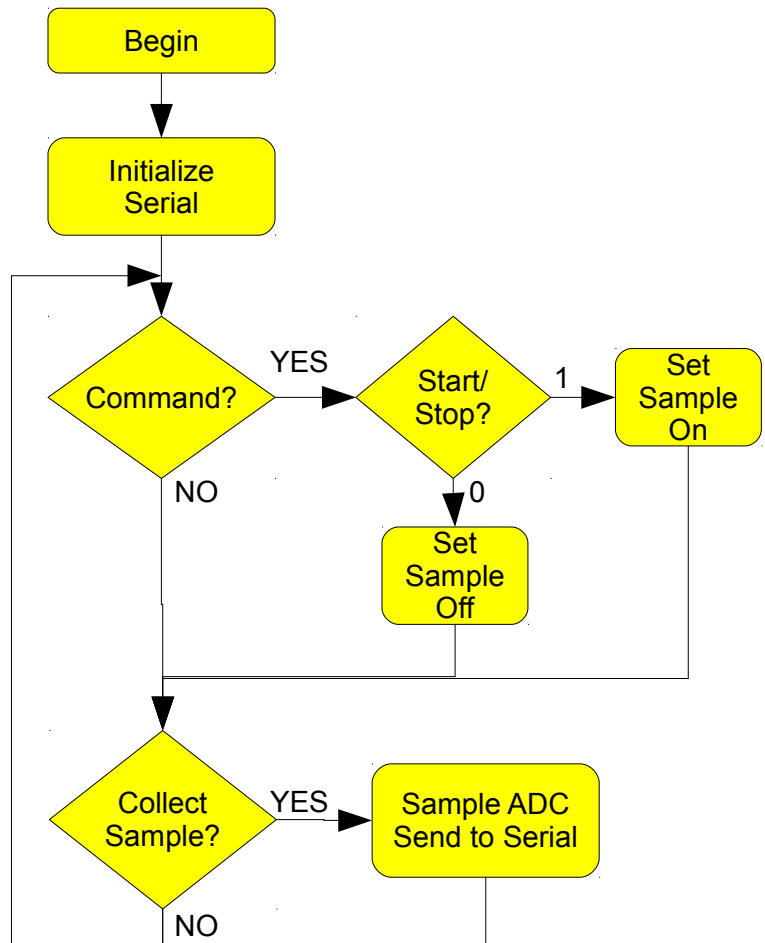
Matlab Interface

Matlab provides functions for interacting with devices. The interface used in the following example is the USB interface which appears to be a COM port. It is the same COM port used for uploading programs.

The example given here will be a simple data acquisition program. The Arduino based processor board will collect data from the ADC and send it over the serial port. The Matlab code will collect the data and plot it.

A simple control command will be included to activate or deactivate the data collection from the ADC.

A single character '1' will be used to start the sampling and '0' will stop the sampling.



The Code

Arduino Code:

Variable **s** is used to turn on and off the data sampling. It is set to zero to be in the off state every time the program runs. In the loop, the the Serial interface is checked to see if any commands were received. The single character command is compared to the two valid options. The state of operation changes only if a valid command is received. If data acquisition is turned on, then the ADC data is collected and sent to the Serial interface. A delay of 1 ms is needed to not overrun the matlab program.

Matlab code:

First, variable **x** is set to an array of 200 elements and cleared to all zeros. Next, the serial interface is declared as variable **s**. Replace COM3 with the serial interface used to program the Arduino based processor board. After the serial interface is declared, the bit rate is set to 115200 bits/second.

fopen() opens the serial interface. This makes the connection. **fprintf()** sends what is between the single quotes to the serial interface. The single character command starts the data collections. The while 1 loop will execute for ever since 1 is considered true. The **for** loop will cycle from 1 to 200 collecting samples from the processor board. After the **for** loop completes, the data is plotted. **drawnow** forces the plot to be displayed and updated. **fscanf()** is what captures the data from the processor board. **'%e'** specifies the format which is a real number that can include a decimal.

Arduino Code

```
int s;

void setup() {
  Serial.begin(115200);
  s = 0;          // start/stop flag
}

void loop() {
  if(Serial.available() > 0) {
    int a = Serial.read();
    if(a == '1') s = 1;
    else if(a == '0') s = 0;
  }
  if(s == 1) {
    int b = analogRead(0);
    Serial.println(b, DEC);
    delay(1);
  }
}
```

Matlab Code

```
x = zeros(200,1);
s = serial('COM3');
s.baudrate = 115200;
fopen(s);
fprintf(s, '1');
while 1
    for b=1:200
        p = fscanf(s, '%e');
        x(b) = p;
    end
    plot(x);
    drawnow;
end
```

Stopping Matlab Program

To stop the code, click on the Pause button. Then click on the Quit Debugging button. In the command window enter `fclose(s)` and press enter. This properly stops the code and closes the serial interface. If this is not done, the program cannot be rerun. If an error indicating the serial interface is not available, restart matlab.

Multiple Variables

Reconnect the IMU and upload the IMU code called **simpleimu**. Three values will be generated per line. The following Matlab code will show how to capture three values.

Three arrays are created to hold the X, Y and Z values from the accelerometer. **p** is assigned values acquired by **fscanf()**. Notice that there are three %e in the function. This tells **fscanf()** to acquire three values. **p** becomes a 3 element array. The x,y,z arrays are filled with the acquired data. Three plots are displayed showing the 200 data points collected for each axis. At 100 samples a second, it will take 2 seconds for the plot to generate and will update in 2 second intervals.

Matlab Code

```
x = zeros(200,1);
y = zeros(200,1);
z = zeros(200,1);
figure;
s = serial('COM1');
s.baudrate = 115200;
fopen(s);

while 1
    for b=1:200
        p = fscanf(s,'%e, %e, %e');
        x(b) = p(1);
        y(b) = p(2);
        z(b) = p(3);
    end
    subplot(3,1,1);
    plot(x);
    subplot(3,1,2);
    plot(y);
    subplot(3,1,3);
    plot(z);
    drawnow;
end
```

Python

Python IDE

Python is an interpreted language. This means that a compiler is not used to generate a executable file. Instead, the python program interprets the written program directly. This lesson assumes basic knowledge of python. You can learn the basics of python at www.learnpython.org

There are a few ways to write python programs. The method shown here is very simple and straight forward. This lesson will introduce you to the pygame library and the socket library that will be used to interact with the SYST101 and SYST395 kits.

Both Windows and Mac OS X will be covered.

For Windows, it is assumed Windows 10 or Windows 7 is being used. Python version 3.8.2 will be used.

Go to www.python.org/downloads/release/python-374/

Scroll down to Files and select **Windows x86-64 executable installer**.

Once downloaded, start the installer program. At the start windows, select **Add Python 3.8 to Path** then click on **Install Now**. Python will now install and show up in the Start menu. Close the window when the installation has completed.

In the start up menu, select **Windows Powershell** and select **Windows Powershell**. Do not select the ISE version.

In the Powershell, type **pip3 install tk** and press **Enter**. This will install the TK library for Python. Once completed, exit **Windows Powershell**. This completes the installation. To start Python, select **Python IDLE** in the **Start** menu.

Python For Mac OS X

It is assumed the latest Mac OS X release is being used. Go to www.python.org/downloads/mac-osx

Locate **python 3.8.2** and select **Mac OS 64-bit installer**.

The installer will be downloaded. Once downloaded, double click on the program and it will go through the installation process. Once installed, the Python 3.8 will be located in the Application folder. Open the Python 3.8 folder and you can double click on IDLE.app to start Python.

To install the TK library, open the terminal application. In the terminal application enter the following:

pip3 install tk and press **Enter**.

The library will be installed.

Python IDLE

The Python IDLE is a python shell. You can enter python commands and they will execute immediately.

Start Python IDLE. At the `>>>` prompt, enter the command:

```
>>>print("Hello world")
Hello world
```

Equations can be entered and Python will generate the answer:

```
>>>5+3
8
```

Python uses indentation to define a block of code. This means all code indented after an if statement, while statement or other conditional statement will be executed.

```
if 10 > 5:
    print("10 is larger")
    print("Second line")
```

After entering the above, press enter again. The commands will execute.

Here is a while loop:

```
while 1:
    print("this is a loop")
```

You can stop the while loop by pressing `<Ctrl>` and `C` keys.

The IDLE will also let you create new programs using an editor. Select menu **File** and **New**. An editor will open and you can enter a python program. Try the program above with the while loop. Save the file. You will be prompted for a name. By convention, use `.py` for the file name extension. Once saved, select menu **Run** and **Run Module**. Press `CTRL` and `C` to stop the program. Now you should have a basic understanding on using the IDLE.

Basic Python

Variables are used to store values or strings. You do not need to declare what type of variable it is. It is automatically determined when a value is assigned. Variables can change types by assignment. Variables must start with a letter. It cannot start with a number. Variables can only be made up of letters, numbers and `_`.

```
a = 5
b = "Fred"
print(a)
print(b)
print(a,b)
```

```
a = 5
print(a)
a = "Fred"
print(a)
```

Math operations are simple in Python. Just use equations.

Notice the `#` sign and the text to the right. This is a comment. Comments are useful to explain what the code is supposed to do.

```
a = 5 + 6           # addition
b = 12.43          # floating point
c = b - 3.14       # subtraction
dog = 45.834       # assignment
cat = dog * b      # multiplication
dd = cat / a       # division
```

Variable and values can be compared with each other. Python provides multiple ways to do comparisons

`==` is equal

`!=` is not equal

`>` greater than

`<` less than

`>=` greater than or equal

`<=` less than or equal

```
a = 45
b = 32
if a > b:
    print("a is greater than b")
if a < b:
    print("a is less than b")
if a == b:
    print("a is equal to b")
if a != b:
    print("a is not equal to b")
```

Enter the code into a program and run it. You can change the values of the variables and see how execution changes.

Rules of Python Coding

When indenting, it is very critical to use the same number of spaces otherwise the code will not be interpreted correctly. Indented code is used to identify code associated with **while**, **for**, **if** and functions. Tabs and spaces are not interchangeable and will cause program errors or incorrect code execution. Using the IDLE editor will help keep the indentation uniform. Problems can occur if you import code from elsewhere and tabs and spaces are not used consistently.

First GUI Program

The Tk or Tkinter library will be used for creating a graphical user interface in the following example programs. Tkinter treats all user interfaces called widgets and graphical components as objects. Widgets includes buttons, pull down menus, radio buttons, text entry, selectors, sliders and canvases. Each widget can be customized in size, color, etc.

The first program will introduce buttons and the canvas. The buttons are clicked on with a mouse and the canvas is an area to render graphics.

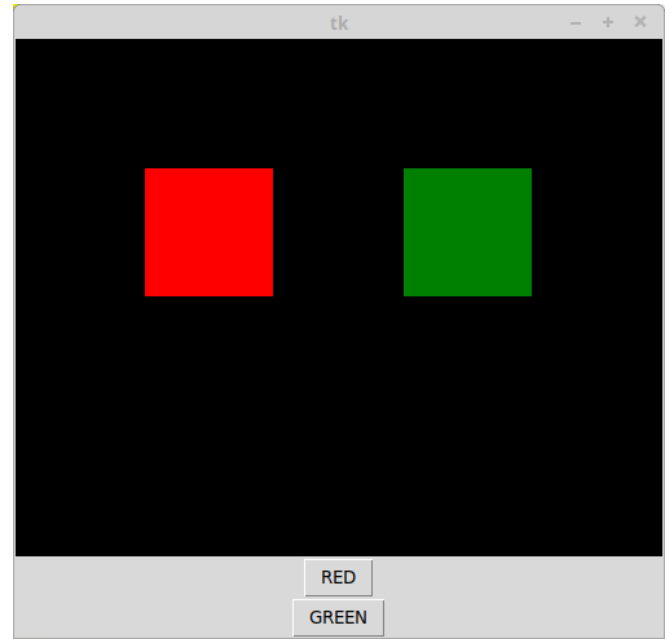
The black region is the canvas and the two buttons are stacked below the canvas.

When clicking on either button, the click action will execute the corresponding function.

The two rectangles rendered in the canvas are also objects. They can be created and assigned to a unique variable. This will allow the rectangles to be modified later. In this example, the color is modified.

Note that the graphic objects are rendered in order of code execution. A graphic object can be on top of another. Modifying the object does not change which is on top.

Widgets are stacked in order of execution from top to bottom using the **pack()** function. There are other window managers to allow more flexible layouts.



TK Library

Tk is a library that allows you to create a graphical user interface. It includes many different widgets such as buttons, menus, text entry, lists and so on. It also provides a canvas widget that allows for rendering graphics.

The code to the right creates a program with two buttons and a canvas to render graphics. When any of the buttons are pressed, a corresponding function is called.

Two rectangles will be created and rendered in the canvas area. When you click on a button, the color of the corresponding rectangle will change color.

At the top, the Tk library also called tkinter is imported. The way it is imported allows the functions to be called more easily.

top = Tk() creates the window for the GUI. The variable **top** relates to the window created. Multiple windows can be created and each will need their own name.

Next, two variables are declared to indicate the color state of the rectangles.

```
from tkinter import *

top = Tk()

toggle_red = 0
toggle_green = 0

c = Canvas(top,bg='black',height=400,width=800)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
    else:
        color = 'gray'
        toggle_red = 0
    c.itemconfig(red_rect,fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
    else:
        color = 'gray'
        toggle_green = 0
    c.itemconfig(green_rect,fill=color)

rrect = c.create_rectangle(100,100,200,200,fill = 'gray')
grect = c.create_rectangle(300,100,400,200,fill = 'gray')
rb = Button(top,text='RED',command=toggle_red_rect)
gb = Button(top,text='GREEN',command=toggle_green_rect)

c.pack()
rb.pack()
gb.pack()
top.mainloop()
```

TK Library

Next, the canvas widget is created with a black background, with 800 pixels in the X direction and 400 pixels in the Y direction. The first argument specifies the window the canvas is to be placed.

Skip the two functions for now.

Two rectangles are created and given variable names **rrect** and **grect**. Graphics rendered are objects and can be later modified. The numbers in the arguments are the top left corner X and Y and the bottom right corner X and Y locations. The **fill** parameter sets the color of the rectangle. Notice the **c.** in front of the create function. This makes the rectangles get rendered in the canvas assigned to variable **c**. You can have more than one canvas.

Two buttons are created. The first argument is the window to place the buttons. The argument **text** sets the text in the button. The argument **command** tells what function to execute when the button is clicked.

```
from tkinter import *

top = Tk()

toggle_red = 0
toggle_green = 0

c = Canvas(top,bg='black',height=400,width=800)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
    else:
        color = 'gray'
        toggle_red = 0
    c.itemconfig(red_rect,fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
    else:
        color = 'gray'
        toggle_green = 0
    c.itemconfig(green_rect,fill=color)

rrect = c.create_rectangle(100,100,200,200,fill = 'gray')
grect = c.create_rectangle(300,100,400,200,fill = 'gray')
rb = Button(top,text='RED',command=toggle_red_rect)
gb = Button(top,text='GREEN',command=toggle_green_rect)

c.pack()
rb.pack()
gb.pack()
top.mainloop()
```


TK Library

At the bottom of the code, there are three lines that tell the window manager to display the widgets. The **pack()** function will pack the widgets in the window in a vertical direction from top to bottom in order of the code.

The last line starts the loop that operates the widgets. The button click will only be detected when **top.mainloop()** is executed. The program stops when you close the window.

Nothing executes after **mainloop()**. This function manages the widgets and calls the related functions when the widgets are activated. This is a type of event based programming where actions occur when events are detected.

```
from tkinter import *

top = Tk()

toggle_red = 0
toggle_green = 0

c = Canvas(top,bg='black',height=400,width=800)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
    else:
        color = 'gray'
        toggle_red = 0
    c.itemconfig(red_rect,fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
    else:
        color = 'gray'
        toggle_green = 0
    c.itemconfig(green_rect,fill=color)

rrect = c.create_rectangle(100,100,200,200,fill = 'gray')
grect = c.create_rectangle(300,100,400,200,fill = 'gray')
rb = Button(top,text='RED',command=toggle_red_rect)
gb = Button(top,text='GREEN',command=toggle_green_rect)

c.pack()
rb.pack()
gb.pack()
top.mainloop()
```

TK Library

Functions in python are declared with the **def** statement. The name of the function and ends in colon. All instructions for the function must be indented as shown.

The statement `global` tells python the variable used is not localized but the same one used in the main program. Python will automatically declare any variables created or referenced inside a function as local. This means you can use the same variable name in the main part of the code and in the function and they will be completely independent. The variable related to Tk are global by default.

The function checks the state of the `toggle_red` or `green` to determine the color of the rectangle. If it is zero, the color is set to red or green and the toggle variable is changed. This allows tracking of the state of the rectangle color. The functions are called only when the button is clicked.

The function **`c.itemconfig()`** lets you change the state of any graphic object you put into the canvas. In this example, the color of the rectangle is changed.

Save the program and call it `buttons.py`. It will be used later.

```
from tkinter import *

top = Tk()

toggle_red = 0
toggle_green = 0

c = Canvas(top,bg='black',height=400,width=800)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
    else:
        color = 'gray'
        toggle_red = 0
    c.itemconfig(red_rect,fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
    else:
        color = 'gray'
        toggle_green = 0
    c.itemconfig(green_rect,fill=color)

rrect = c.create_rectangle(100,100,200,200,fill='gray')
grect = c.create_rectangle(300,100,400,200,fill='gray')
rb = Button(top,text='RED',command=toggle_red_rect)
gb = Button(top,text='GREEN',command=toggle_green_rect)

c.pack()
rb.pack()
gb.pack()
top.mainloop()
```

Tk Canvas

In this example, you will use a mouse to draw on the canvas.

There are two functions in this program, one to paint and the other to clear the screen.

top.title() function Changes the title of the window at the top.

The Canvas is created as before. **w.bind()** is used to capture mouse events when the mouse is over the canvas. The first one is **<B1-Motion>** which is used to call the paint function. When the left button is pressed and the mouse is moved, the paint function will be called repeatedly until motion stops and the button is released. The **w.bind("<Button-3>", clear)** is used to detect the click of the right button. The clear function is called any time the right button is clicked while the mouse is over the canvas.

A new widget in the code is the Label. This is used to place a text label in the window. It is placed below the canvas due to the order of the **pack()** functions.

The **paint()** function renders a small red circle each time it is called at the mouse location. The event passes mouse information into the variable event. The event variable is an object that contains the X and Y position of the mouse.

```
from tkinter import *

canvas_width = 500
canvas_height = 300

def paint( event ):
    color = "#ff0000"
    x1, y1 = ( event.x - 1 ), ( event.y - 1 )
    x2, y2 = ( event.x + 1 ), ( event.y + 1 )
    w.create_oval( x1, y1, x2, y2, fill = color )

def clear(event) :
    w.create_rectangle(0,0,canvas_width,canvas_height,fill='black')

top = Tk()
top.title( "Painting using Ovals" )
w = Canvas(top,
           width=canvas_width,
           height=canvas_height,bg='black')
w.pack()
w.bind( "<B1-Motion>", paint )
w.bind("<Button-3>",clear)

message = Label( top, text = "Press and Drag the mouse to draw" )
message.pack()

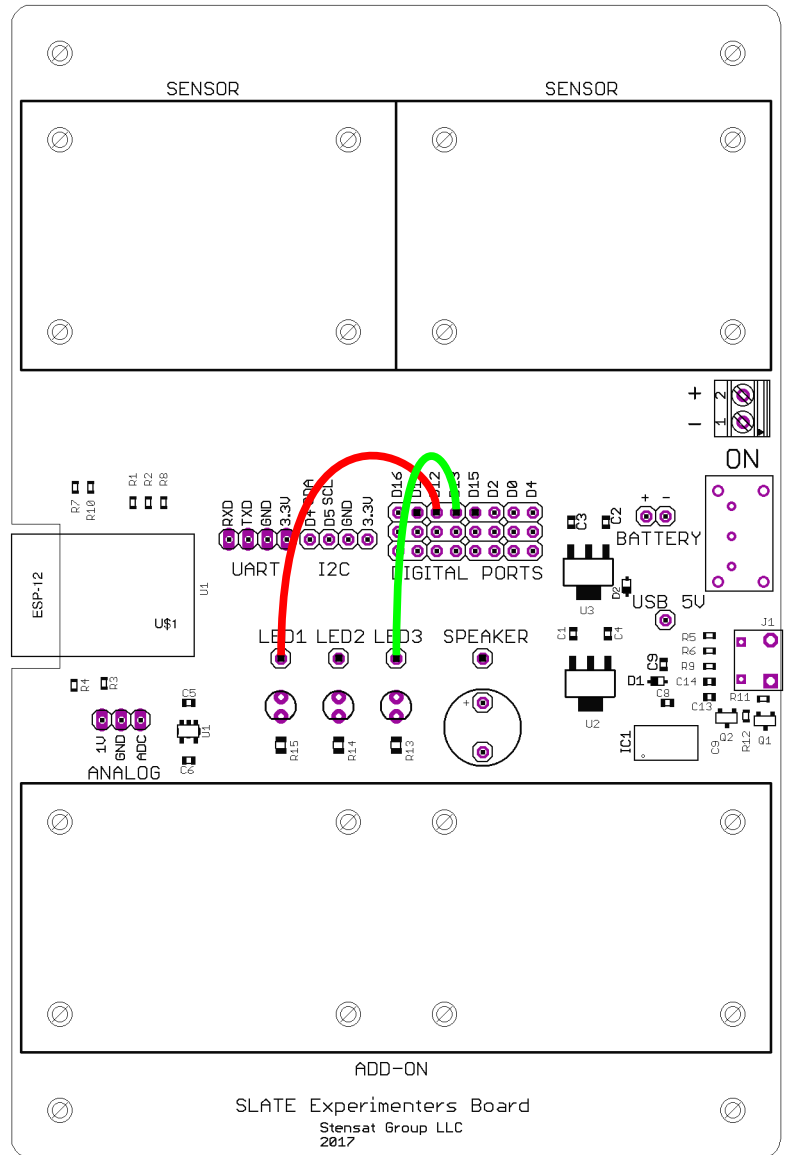
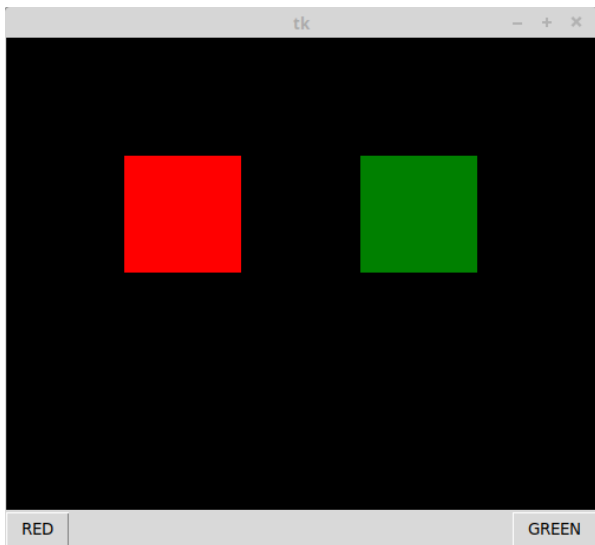
mainloop()
```

The link below provides more information on how to link events. Any object such as buttons and labels can have events attached. Even the top object can have events. Events are not limited to the mouse, keyboard inputs can also be detected.

<https://effbot.org/tkinterbook/tkinter-events-and-bindings.htm>

Graphical Interface

In this section, you will learn how to build a simple graphical user interface to control two LEDs through the USB port. Reuse the two LED circuit from the BASIC Circuits lesson. Two buttons will be created to control the two LEDs. A new program will be created in Processing to command the SLATE and a new program for the SLATE will be created to accept and process the commands.



Graphical Interface

Arduino Program

The Arduino program will look for a single character command from the serial interface over the USB port and process it. It first checks if a command character has been received. Once received, it will read the character and then use the **switch()** function to determine which command was received and turn the appropriate LED on or off.

Upload this to the Experimenters board. Save the program with the file name **RedGreenSerial**.

Open the serial monitor. Type each letter in the serial monitor and press the **Enter** key. The program should respond to the letter selected. Make sure to use caps.

```
void setup() {
  Serial.begin(115200);
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);
}

void loop() {
  if(Serial.available() > 0) {
    int a = Serial.read();
    switch(a) {
      case 'F' : digitalWrite(12,HIGH);
                 break;
      case 'B' : digitalWrite(12,LOW);
                 break;
      case 'L' : digitalWrite(13,HIGH);
                 break;
      case 'R' : digitalWrite(13,LOW);
    }
  }
}
```

Graphical Interface

Python Program

We will reuse the program buttons.py.

In order for the python program to talk to the SLATE board, a new library needs to be added.

Open a terminal or powershell. Enter the command:

pip3 install pyserial

You do need to be connected to the internet for the library to be installed. Once completed, you can start writing the code.

The highlighted lines are where the new serial code will be added.

The serial interface is opened and configured to operate at 115200 baud. Specify the COM port used by the Arduino software.

```
from tkinter import *
import serial

top = Tk()

toggle_red = 0
toggle_green = 0

s = serial.Serial("COM4",115200)

c = Canvas(top,bg='black',height=400,width=500)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
        s.write(b'F')
    else:
        color = 'gray'
        toggle_red = 0
        s.write(b'B')
    c.itemconfig(red_rect,fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
        s.write(b'L')
    else:
        color = 'gray'
        toggle_green = 0
        s.write(B'R')
    c.itemconfig(green_rect,fill=color)

red_rect = c.create_rectangle(100,100,200,200,fill='gray')
green_rect = c.create_rectangle(300,100,400,200,fill =
'gray')
rb = Button(top,text='RED',command=toggle_red_rect)
gb = Button(top,text='GREEN',command=toggle_green_rect)

c.pack()
rb.pack(side=LEFT)
gb.pack(side=RIGHT)
top.mainloop()
```

Graphical Interface

Python Program

s.write() is the function that writes the command byte to the SLATE. Notice the character being sent is preceded with the letter **b**. This converts the character which is automatically in unicode to a byte character.

Python 3 handles all strings as unicode which is 16-bits long. The serial interface cannot support that so the string needs to be converted to an 8-bit character.

Notice the last two **pack()** functions. The argument added will make the two buttons be positioned at the same level with the RED button the left and the GREEN button on the right.

Save the program as **button_serial.py**.

```
from tkinter import *
import serial

top = Tk()

toggle_red = 0
toggle_green = 0

s = serial.Serial("COM4",115200)

c = Canvas(top,bg='black',height=400,width=500)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
        s.write(b'F')
    else:
        color = 'gray'
        toggle_red = 0
        s.write(b'B')
    c.itemconfig(red_rect,fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
        s.write(b'L')
    else:
        color = 'gray'
        toggle_green = 0
        s.write(b'R')
    c.itemconfig(green_rect,fill=color)

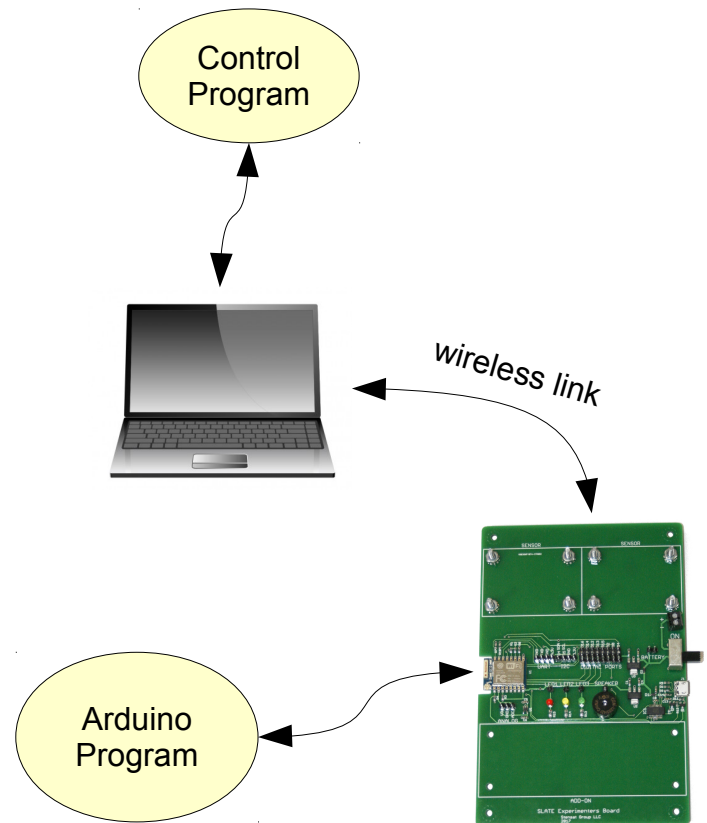
red_rect = c.create_rectangle(100,100,200,200,fill='gray')
green_rect = c.create_rectangle(300,100,400,200,fill='gray')
rb = Button(top,text='RED',command=toggle_red_rect)
gb = Button(top,text='GREEN',command=toggle_green_rect)

c.pack()
rb.pack(side=LEFT)
gb.pack(side=RIGHT)
top.mainloop()
```

WiFi

The WiFi integrated in the processor provides a wireless way to communicate with the processor. The processor will be configured as an access point. This means it becomes a local network where your laptop connects. It is also possible to have a tablet connect to the processor. In this lesson, you will learn how to control digital pins. This will require you to write code on the laptop.

This drawing shows how everything is interconnected. The control program runs on the laptop. The laptop WiFi connects to the processor board WiFi. The control program sends commands over the WiFi to the processor board. The Arduino program interprets the commands and executes them.



What is WiFi

WiFi is a local area wireless computer network. It is also known as wireless local area network. **WiFi** is a standard for allowing computers to interact with each other using radio signals. A **wireless access point** is a device that connects a wireless network to a wired network. It can also provide a local isolated network not connected to the internet or other wired network. Access points usually have a network router and can provide network addresses or IP addresses to any device that connects.

- **SSID** – is a unique identifier for the WiFi network. It can have up to 32 characters and is case sensitive. This allows multiple WiFi access points in the same area without interfering with each other.
- **IP Address** – is the internet protocol address assigned to each device on the network. There are two standards, IP-4 and IP-6. IP-4 is used here. The address consists of four sets of numbers separated by a decimal point. Each number has a range of 0 to 255. Example 192.168.1.10.
- **DHCP** – is Dynamic Host Configuration Protocol. This protocol allows a WiFi router to assign an IP address to any device that connects to the WiFi network. This is done automatically.
- **TCP** – is Transmission Control Protocol. This is one of the main network protocols used by any device on any WiFi network or the internet. The protocol enables two devices to establish a connection to each other and exchange data. The protocol guarantees delivery of data and that the data is delivered in the same order sent. The sender sends a data packet, when the receiver gets the packet, it sends an acknowledgment. If the receiver doesn't receive the packet, the sender will send again after a time out period.
- **UDP** – is user datagram Protocol. This protocol is a stateless protocol. No connection needs to be made and packets received are not acknowledged. The sender just sends a packet to an IP address and port. There is no guarantee the receiver actually received any packets. Data packets can be sent much more quickly because there is no handshaking.

There are two parts to the WiFi operation. Configuration which sets up the module to operate properly. Data operation where the module receives data and can send data. The WiFi module will be configured to operate as an access point. This allows another computer to connect to the module and communicate with the module. More than one WiFi access point can be in the same area and operate independent of each other as long as their SSID are different. In this lesson, the WiFi module will be configured as an access point and allow TCP connections.

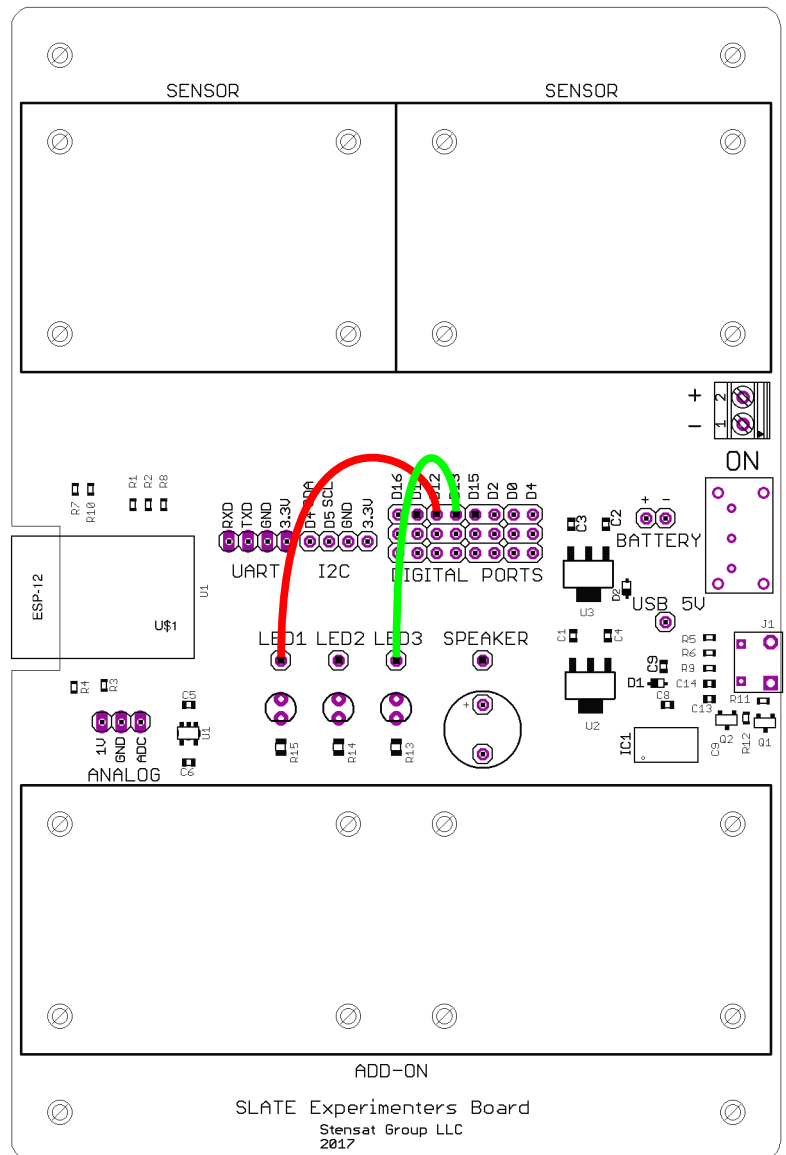
Remote LED Control

In this example, you will control the red and green LEDs using TCP packets. This example will require a program on the processor board and on the laptop. The processor board program will wait for the laptop to connect via WiFi connection and interpret the commands. The laptop program will detect certain keys on the keyboard being pressed and send commands to the processor board.

NOTE:

Some things to remember when uploading code to the SLATE. Each time code is uploaded, any network operation is stopped. After uploading code, you will need to reconnect your laptop WiFi to the SLATE access point. Windows may show the laptop is still connected but it really is not. Disconnect and connect again. Any time you upload code to the SLATE, the access point software stops functioning and Windows will have stale data about the connection.

If you use the menu Include Library to add the WiFi to the code, a whole bunch of include statements will be added. Remove all the ones except what is shown in the code in this document. Some of those include files can cause issues with compiling and generate code that doesn't execute properly.



WiFi Configuration

First thing to do is include the **ESP8266WiFi** library by adding the include statement to the top of the program. Some items need to be declared. A **WiFiClient** object needs to be created. This allows the code to get commands from the laptop and send telemetry.

WiFiServer object needs to be created so the laptop can connect and send data to the Experimenters Kit. This allows the kit to receive connections. When creating the **WiFiServer** object, the network port is selected.

A character array is created for holding the commands sent by the laptop. For now, the first character in the array will be the command.

WiFi.mode() is used to configure the operating mode of the WiFi interface. **WIFI_AP** parameter configures the WiFi interface to operate as an access point where it will have a default address of **192.168.4.1** and assign any device connecting to it a different address.

Last operation is to set up the WiFi as an access point. **WiFi.softAP()** will set up the Experimenters Kit as an access point with the SSID specified. If a password is desired then the format is:

```
WiFi.softAP("ssid", "password");
```

After the access point is configured, the server is started. This implements the ability for clients to connect to the kit.

Arduino Program

```
#include <ESP8266WiFi.h>

WiFiClient client;
WiFiServer server(80);

unsigned char cmd[6];

void setup()
{
  Serial.begin(115200);
  pinMode(14, OUTPUT);
  pinMode(16, OUTPUT);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("nameofboard");
  server.begin();
}

void loop()
{
}
```

Add the **loop()** function to the program if it isn't already included. Upload the program and let it run. On your computer, look up the available wireless networks and see if the one you named appears on the list. It may take a little while since the laptop OS checks for available networks at some interval of seconds. If it appears, try connecting. If you include a password, you should be prompted to enter a password.

Command Processing

A unique byte value is required to differentiate the LEDs. The table below shows the commands for controlling the LEDs. A single letter will represent each action.

In the **loop()** function, two things need to be checked. Has a client connected to the processor? Has a command been received? One big rule about writing code. No infinite loops in the **loop()** function. This will cause the processor to crash. It needs to enter and exit the loop function repeatedly or execute a **delay()** function.

The first thing that is checked is if a client is connected to the processor. **(2)** The object **client** is assigned to a client that has connected. If no client has connected then the **client** object is empty or null. **(3)** The **if()** statement checks if the **client** object is null or not. The result of the **if()** statement is always true if the variable is not empty or null. If a client has connected, **(4)** the statement **Connected** will be displayed on the serial monitor.

(5) A **while()** loop is created to process all commands while the client is connected. As long as the result of **client.connected()** is true, the code inside the **while()** loop will be executed.

Arduino Program

```
1 void loop() {
2   client = server.available();
3   if(client) {
4     Serial.println("Connected");
5     while(client.connected()) {
6       while(!client.available()) {
7         if(!client.connected()) break;
8         delay(1);
9       }
10    char a = client.read();
11    switch(a) {
12      case 'F' : digitalWrite(14,HIGH);
13                break;
14      case 'B' : digitalWrite(14,LOW);
15                break;
16      case 'L' : digitalWrite(16,HIGH);
17                break;
18      case 'R' : digitalWrite(16,LOW);
19                break;
20    }
21  }
22 }
23 }
```

Action	Command
Red LED On	F
Red LED Off	B
Green LED On	L
Green LED Off	S

Command Processing

Line (6) is where the code is looking for any commands sent to the Experimenters Kit. It works the same as **Serial.available()**. The **while()** loop here executes as long as there are no commands being sent. It does two things. First, it checks to make sure a client is still connected otherwise the **while()** loop will get stuck forever. Second, a **delay()** function is executed. This allows the processor to multi-task and handle WiFi operations. If the client disconnects, the break causes the code to exit the the **while()** loop.

After a command has been received, the code exits the while loop and then the command byte is read. **(10)** Reading a byte from the client is the same as reading a byte from the serial interface. **(11 – 20)** The command is then checked in the **switch()** statement. The **switch** statement allows a variable to be compared against a list of values. The values are listed after the **case** statement. If the value matches, the code after the **case** statement is executed.

A **break** statement is needed to exit the switch statement otherwise all code after the matched **case** will be executed.

Arduino Program

```
1 void loop() {
2   client = server.available();
3   if(client) {
4     Serial.println("Connected");
5     while(client.connected()) {
6       while(!client.available()) {
7         if(!client.connected()) break;
8         delay(1);
9       }
10    char a = client.read();
11    switch(a) {
12      case 'F' : digitalWrite(14,HIGH);
13                break;
14      case 'B' : digitalWrite(14,LOW);
15                break;
16      case 'L' : digitalWrite(16,HIGH);
17                break;
18      case 'R' : digitalWrite(16,LOW);
19                break;
20    }
21  }
22 }
23 }
```

Action	Command
Red LED On	F
Red LED Off	B
Green LED On	L
Green LED Off	R

Control Software

Open **button_serial.py** and save it to a new name such as **button_tcp.py**.

Replace the **import serial** with **import socket**. This loads the network socket library.

Replace the line that opens the serial port with the **socket.socket()** function. This function sets of the network connection to be TCP. **SOCK_STREAM** is the parameter that specifies TCP protocol.

Next, establish the connection with **s.connect()**. Notice the IP address and port are a tuple type value.

Replace the **s.write()** functions with **s.sendall()**.

Load the SLATE board with the software and let it start up. Connect to the SLATE access point then run the python program.

Python Program

```
from tkinter import *
import socket

top = Tk()

toggle_red = 0
toggle_green = 0

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('192.168.4.1', 80))
c = Canvas(top, bg='black', height=400, width=500)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
        s.sendall(b'F')
    else:
        color = 'gray'
        toggle_red = 0
        s.sendall(b'B')
    c.itemconfig(red_rect, fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
        s.sendall(b'L')
    else:
        color = 'gray'
        toggle_green = 0
        s.sendall(b'R')
    c.itemconfig(green_rect, fill=color)

red_rect = c.create_rectangle(100,100,200,200, fill='gray')
green_rect = c.create_rectangle(300,100,400,200, fill='gray')
rb = Button(top, text='RED', command=toggle_red_rect)
gb = Button(top, text='GREEN', command=toggle_green_rect)

c.pack()
rb.pack(side=LEFT)
gb.pack(side=RIGHT)
top.mainloop()
```

UDP Connection

Arduino Program

To use the UDP protocol, modify the SLATE program as shown to the right. Another include file is required for UDP. The client has been replaced with a udp object. The cmd array has been increased in size to support possible large packets.

In the setup, the only change is replacing server with udp and specifying the port number.

In the loop, the program looks for a UDP packet. If there is one, the size is returned or zero for no packet. The **if(packetsize)** is true when **packetsize** is not zero. The packet is read and then the first byte of the array is extracted. The rest of the code is not changed.

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

WiFiUDP udp;

unsigned char cmd[256];

void setup()
{
  Serial.begin(115200);
  pinMode(14, OUTPUT);
  pinMode(16, OUTPUT);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("nameofboard");
  udp.begin(80);
}

void loop()
{
  int packetsize = udp.parsePacket();
  if(packetsize) {
    udp.read(cmd, 256);
    int a = cmd[0];
    switch(a) {
      case 'F' :
        digitalWrite(14, HIGH);
        break;
      case 'B' :
        digitalWrite(14, LOW);
        break;
      case 'L' :
        digitalWrite(16, HIGH);
        break;
      case 'R' :
        digitalWrite(16, LOW);
        break;
    }
  }
}
```

UDP Connection

To use UDP, replace `SOCK_STREAM` with `SOCK_DGRAM` to specify the UDP protocol.

Delete `s.connect()` since UDP protocol does not require connecting to a server.

Add a variable `address` and assign it the tuple with the IP address and port number.

Replace the `s.sendall()` functions with `s.sendto()`. The `s.sendto()` function requires the destination address.

Load the SLATE board with the software and let it start up. Connect to the SLATE access point then run the python program. The program should operate the same way just using a different protocol.

```
from tkinter import *
import socket

top = Tk()

toggle_red = 0
toggle_green = 0

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
address = ('192.168.4.1', 80)
c = Canvas(top, bg='black', height=400, width=500)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
        s.sendto(b'F', address)
    else:
        color = 'gray'
        toggle_red = 0
        s.sendto(b'B', address)
    c.itemconfig(red_rect, fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
        s.sendto(b'L', address)
    else:
        color = 'gray'
        toggle_green = 0
        s.sendto(b'R', address)
    c.itemconfig(green_rect, fill=color)

red_rect = c.create_rectangle(100,100,200,200, fill='gray')
green_rect = c.create_rectangle(300,100,400,200, fill='gray')
rb = Button(top, text='RED', command=toggle_red_rect)
gb = Button(top, text='GREEN', command=toggle_green_rect)

c.pack()
rb.pack(side=LEFT)
gb.pack(side=RIGHT)
top.mainloop()
```


Sending Data

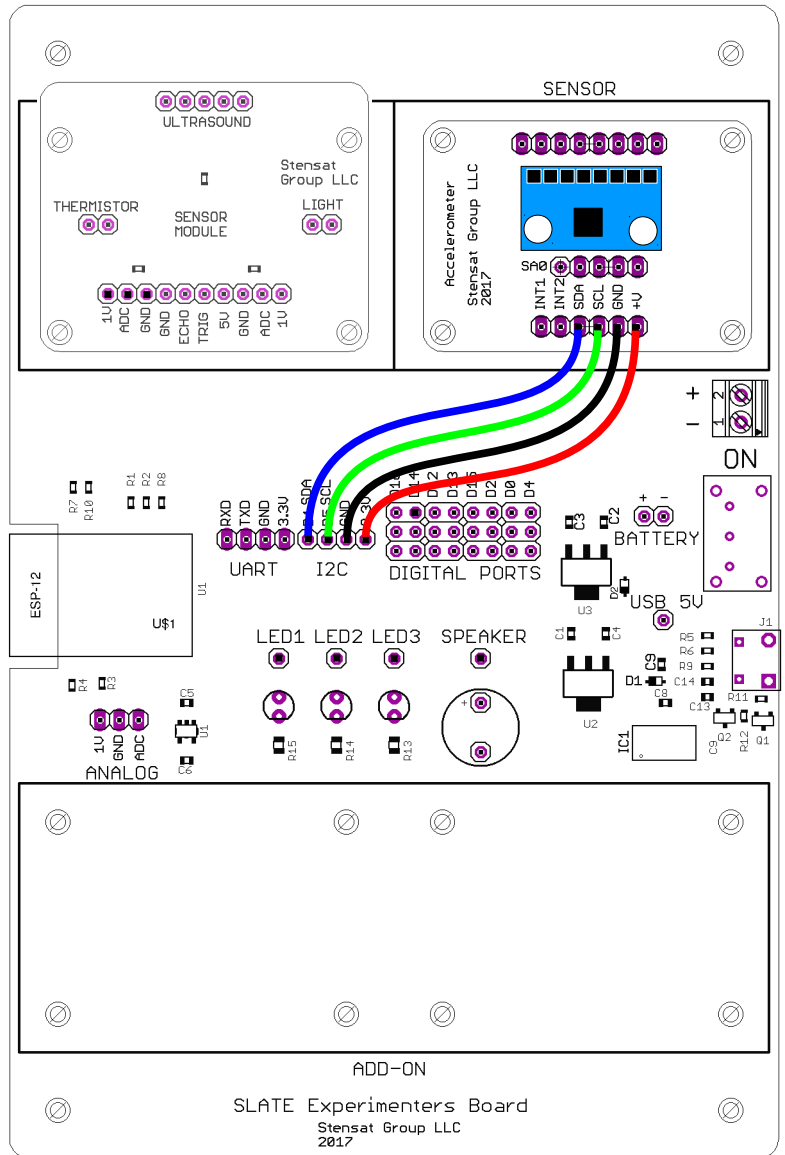
In this section, you will be shown how to send data from the SLATE to the a python program. The python program will receive the data and plot it in real time. To plot data, the matplotlib library will be used. This library work similar to Matlab plotting functions.

The first version will show how to collect data over the serial interface and plot it. The second version will show how to collect data over WiFi and plot it.

Sending Data

The IMU program from earlier will be used for this section. The IMU program generated orientation in degrees for the three axis.

Reconnect the IMU as shown.



Sending Data

The program from the IMU section will be reused as is. The code is shown to the right. The only change in the code is changing the delay to 10 milliseconds instead of 100 ms at the end of the program. The python program can handle higher data rates.

For python, a new library needs to be added. Open a terminal or powershell and execute the programs below.

pip3 install matplotlib

For more information on matplotlib, go to <https://matplotlib.org/>

Arduino Program

```
#include <MPU6050_tockn.h>
#include <Wire.h>

MPU6050 mpu6050(Wire);
long timer = 0;
char buf[64];

void setup() {
  Serial.begin(115200);
  Wire.begin(4,5);
  mpu6050.begin(ACCEL_2G,GYRO_500);
  mpu6050.calcGyroOffsets(true);
}

void loop() {
  mpu6050.update();
  Serial.print(mpu6050.getAngleX());
  Serial.print(",");
  Serial.print(mpu6050.getAngleY());
  Serial.print(",");
  Serial.println(mpu6050.getAngleZ());
  delay(10);
}
```

Sending Data

Python Program

This python program will receive data from the serial interface and plot the X,Y,Z data from the IMU as the data becomes available.

Three libraries are imported. First is the matplotlib library. Notice the **as plt**. This renames the library to **plt** so there is less typing. The second matplotlib library is to support real time updates of the plots.

Line 5 creates a figure. This will be window that contains the plot. Line 6 adds a plot to the figure. There is only one plot.

Line 8 sets the number of data points to be plotted. Line 9 declares a variable to hold the Y scale of the plot. The IMU is operating at 2 G range.

Line 11 fills in the x-axis values from 0 to 199. Line 12, 13, 14 fill arrays with zero of length 200. Each of the arrays will hold the IMU X, Y, Z values.

Line 15 sets the Y scale for the plot using the variable value.

Line 16 opens the serial port.

Line 17 declares a line plot for the X value. Line 18 does the same for the Y value and line 19 does the same for the X value. Each data plot will be plotted in the one plot in the figure. The name of the plot is **ax**.

Lines 20 to 22 set up the plot labels. Line 23 makes the data legend visible.

```
1  import matplotlib.pyplot as plt
2  import matplotlib.animation as animation
3  import serial
4
5  fig = plt.figure()
6  ax = fig.add_subplot(1, 1, 1)
7
8  x_len = 200
9  y_range = [-200, 200]
10
11 xs = list(range(0, 200))
12 xa = [0] * x_len
13 ya = [0] * x_len
14 za = [0] * x_len
15 ax.set_ylim(y_range)
16 s = serial.Serial('COM5',115200)
17 line, = ax.plot(xs, xa,label='X')
18 line2, = ax.plot(xs,ya,label='Y')
19 line3, = ax.plot(xs,za,label='Z')
20 plt.title('IMU')
21 plt.xlabel('Samples')
22 plt.ylabel('Degrees')
23 ax.legend()
24
25 def animate(i, xa,ya,za):
26     a = s.readline()
27     b = a.decode('utf-8')
28     c = b.split(',')
29     if len(c) == 3:
30         xa.append(float(c[0]))
31         ya.append(float(c[1]))
32         za.append(float(c[2]))
33         xa = xa[-x_len:]
34         ya = ya[-x_len:]
35         za = za[-x_len:]
36         line.set_ydata(xa)
37         line2.set_ydata(ya)
38         line3.set_ydata(za)
39         return line,line2,line3,
40     else:
41         return line,line2,line3,
42 ani = animation.FuncAnimation(fig,animate,
43     fargs=(xa,ya,za,),
44     interval=1,
45     blit=True)
46 plt.show() # show the figure
```

Sending Data

Lines 25 through 41 are for the function **animate**. This function is called repeatedly to update the plot. Variables **i,xa,ya,za** are passed to the function. variable **i** is automatically passed and provides a count update. It is not used.

Line 26 reads data from the serial interface. If data is not available, the program halts until it becomes available. The received data is a byte array stored in variable **a**.

Line 27 converts the byte array into a string which uses unicode. Python 3 works with strings so the byte array needs to be converted.

<https://en.wikipedia.org/wiki/UTF-8>

Line 28 splits the received data into individual X, Y, Z values. The data is separated by a space. the `split()` function argument is the character that is used to separate the values.

Line 29 verifies there are three values. Some times the program will start reading the serial interface in the middle of data being sent and not all the data is received. This makes sure all three values have been received so the program will not crash.

Lines 30 to 32 add the data to the arrays. Since the split values are still strings, the strings need to be converted to floating point values.

Lines 33 to 35, trim the arrays back to 200 values removing the oldest value.

```
1  import matplotlib.pyplot as plt
2  import matplotlib.animation as animation
3  import serial
4
5  fig = plt.figure()
6  ax = fig.add_subplot(1, 1, 1)
7
8  x_len = 200
9  y_range = [-200, 200]
10
11 xs = list(range(0, 200))
12 xa = [0] * x_len
13 ya = [0] * x_len
14 za = [0] * x_len
15 ax.set_ylim(y_range)
16 s = serial.Serial('COM5',115200)
17 line, = ax.plot(xs, xa,label='X')
18 line2, = ax.plot(xs,ya,label='Y')
19 line3, = ax.plot(xs,za,label='Z')
20 plt.title('IMU')
21 plt.xlabel('Samples')
22 plt.ylabel('Degrees')
23 ax.legend()
24
25 def animate(i, xa,ya,za):
26     a = s.readline()
27     b = a.decode('utf-8')
28     c = b.split(' ')
29     if len(c) == 3:
30         xa.append(float(c[0]))
31         ya.append(float(c[1]))
32         za.append(float(c[2]))
33         xa = xa[-x_len:]
34         ya = ya[-x_len:]
35         za = za[-x_len:]
36         line.set_ydata(xa)
37         line2.set_ydata(ya)
38         line3.set_ydata(za)
39         return line,line2,line3,
40     else:
41         return line,line2,line3,
42 ani = animation.FuncAnimation(fig,animate,
43     fargs=(xa,ya,za,),
44     interval=1,
45     blit=True)
46 plt.show() # show the figure
```

Sending Data

Lines 36 to 38 update the plot data by reloading the **linex, liney, linez** with the updated arrays.

The updated arrays are returned in line 39.

Line 40 and 41 handle the situation where data was not in the proper format. The non updated **linex, liney, linez** are returned.

Line 42 sets up the real time plotting. The first argument identifies the figure to be updated. The second argument is the function that does update the data. The third argument specifies the data arrays the animate function will use. The **interval** argument specifies how fast to animate the plot. The **blit=True** accelerates the plotting so it can keep up with the data.

interval is set to 1 millisecond. This is faster than the data being generated. This is done so that the program does not lag behind the data. The **readline()** function will control the speed of the plotting based on the rate the data is received.

Line 46 makes the figure with the plot visible.

```
1 import matplotlib.pyplot as plt
2 import matplotlib.animation as animation
3 import serial
4
5 fig = plt.figure()
6 ax = fig.add_subplot(1, 1, 1)
7
8 x_len = 200
9 y_range = [-200, 200]
10
11 xs = list(range(0, 200))
12 xa = [0] * x_len
13 ya = [0] * x_len
14 za = [0] * x_len
15 ax.set_ylim(y_range)
16 s = serial.Serial('COM5', 115200)
17 line, = ax.plot(xs, xa, label='X')
18 line2, = ax.plot(xs, ya, label='Y')
19 line3, = ax.plot(xs, za, label='Z')
20 plt.title('IMU')
21 plt.xlabel('Samples')
22 plt.ylabel('Degrees')
23 ax.legend()
24
25 def animate(i, xa, ya, za):
26     a = s.readline()
27     b = a.decode('utf-8')
28     c = b.split(',')
29     if len(c) == 3:
30         xa.append(float(c[0]))
31         ya.append(float(c[1]))
32         za.append(float(c[2]))
33         xa = xa[-x_len:]
34         ya = ya[-x_len:]
35         za = za[-x_len:]
36         line.set_ydata(xa)
37         line2.set_ydata(ya)
38         line3.set_ydata(za)
39         return line, line2, line3,
40     else:
41         return line, line2, line3,
42 ani = animation.FuncAnimation(fig, animate,
43                               fargs=(xa, ya, za, ),
44                               interval=1,
45                               blit=True)
46 plt.show() # show the figure
```

Sending Data over WiFi

In this section, the accelerometer data will be sent to the python program over WiFi using a UDP network connection. Remember a UDP connection does not require the SLATE to make a connection to a computer. In this example, the python program will operate as a server and listen for UDP packets.

Arduino Code

The setup code will be similar to the original accelerometer code except with the addition of configuring the WiFi.

The **ESP8266WiFi.h** include file is added to the top of the code. The **WiFiUdp.h** include file is also added.

The **udp** object is declared to send and receive UDP packets.

The variable **send_data** is declared and set to zero. This will be used to indicate when a packet is received from the python program.

The **remote** variable is declared as an IPAddress type variable. This will be used to store the IP address of the computer running the python program.

The variable **buf** will hold the data to be sent to the python program as a character array.

In the setup() function, the I2C interface is configured as before and the WiFi is configured. Last, the accelerometer is configured as before.

A UDP port is declared so the SLATE can get a packet from the python program and acquire the IP address to send data.

Arduino Program

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include <MPU6050_tockn.h>
#include <Wire.h>

WiFiUDP udp;

int send_data = 0;
IPAddress remote;
char buf[32];

MPU6050 mpu6050(Wire);

void setup() {
  Wire.begin(4,5);
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP("SSID name");
  udp.begin(10000);
  mpu6050.begin(ACCEL_2G, GYRO_500);
  mpu6050.calcGyroOffsets(true);
}
```


Arduino Code

In the `loop()` function, the `udp` port is checked for any UDP packet from the python program if a packet has not been received before. The contents do not matter for this program. The purpose is to acquire the IP address of the computer the python program is running on. When a packet is received, the **send_data** variable is set to 1 and the IP address is captured.

When the program has captured the IP address and `send_data` is set to 1, the program will then get the accelerometer data and send it to the python program.

The **snprintf()** function format the data into a character array that can be read by a person. The values are converted into an ASCII string. The first parameter is the character array to store the string. The second parameter is the size of the character array. The third parameter specifies the formatting of the string. After that, the rest of the parameters are the variables that are used to put values in the string.

To send a UDP packet, **udp.beginPacket()** is required to specify the IP address and network port. **udp.print()** fills the packet with the contents. Multiple **udp.print()** statements are allowed. The packet is sent when **udp.endPacket()** is executed.

Arduino Program

```
void loop() {
  int reg[6];
  int i;
  if(send_data == 0) {
    int ps = udp.parsePacket();
    if(ps > 0) {
      send_data = 1;
      remote = udp.remoteIP();
    }
  }
  if(send_data == 1) {
    mpu6050.update();
    float gx = mpu6050.getAngleX();
    float gy = mpu6050.getAngleY();
    float gz = mpu6050.getAngleZ();
    snprintf(buf, 32, "%f,%f,%f\n", gx, gy, gz);
    udp.beginPacket(remote, 10000);
    udp.print(buf);
    udp.endPacket();
    delay(10);
  }
}
```

Python Code

The same plotting python program will be modified to use the network socket. The UDP protocol will be used.

Line 3 is modified to import the socket library instead of the serial library.

Line 16 sets up the type of network connection to be UDP. Line 17 establishes a connection to the SLATE. This is done to get the local IP of the device making the connection. There can be more than one network device. Line 18 gets the local IP address of the host computer that the python program is hosted. Line 19 closes the network connection.

Line 20 sets up the UDP network type again. Line 21 tells the program to start listening to the network port on the local IP address.

Line 22 sends a packet to the SLATE. This allows the SLATE to get the IP address of the host computer running the python program.

Python Program

```
1 import matplotlib.pyplot as plt
2 import matplotlib.animation as anim
3 import socket
4
5 fig = plt.figure()
6 ax = fig.add_subplot(1, 1, 1)
7
8 x_len = 200
9 y_range = [-200, 200]
10
11 xs = list(range(0, 200))
12 xa = [0] * x_len
13 ya = [0] * x_len
14 za = [0] * x_len
15 ax.set_ylim(y_range)
16 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
17 s.connect(('192.168.4.1', 80))
18 localip = s.getsockname()[0]
19 s.close()
20 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
21 s.bind((localip, 10000))
22 s.sendto(b'start', ('192.168.4.1', 10000))
```

Python Code

Lines 23, 24 and 25 create plots for each axis of the IMU. The object **ax** refers to the one subplot. Any plot functions to the same object gets plotted on the same subplot.

Lines 26 to 28 set up the labels. Line 29 makes the legend visible.

Lines 31 through 47 are the same as before. The only change is line 32. It is changed to read a UDP packet that has been received. If none has been received, the program waits here. Notice the function returns two values. The first is the packet contents into variable **a**. The second is the IP address and port from the SLATE.

The rest of the program is not changed.

Python Program

```
23 lineX, = ax.plot(xs, xa, label='X')
24 lineY, = ax.plot(xs, ya, label='Y')
25 lineZ, = ax.plot(xs, za, label='Z')
26 plt.title('MPU6050')
27 plt.xlabel('Samples')
28 plt.ylabel('Degrees')
29 ax.legend()
30
31 def animate(i, xa, ya, za):
32     a, d = s.recvfrom(256)
33     b = a.decode('utf-8')
34     c = b.split(',')
35     if len(c) == 3:
36         xa.append(float(c[0]))
37         ya.append(float(c[1]))
38         za.append(float(c[2]))
39         xa = xa[-x_len:]
40         ya = ya[-x_len:]
41         za = za[-x_len:]
42         lineX.set_ydata(xa)
43         lineY.set_ydata(ya)
44         lineZ.set_ydata(za)
45         return lineX, lineY, lineZ,
46     else:
47         return lineX, lineY, lineZ,
48 ani = anim.FuncAnimation(fig, animate,
49                          fargs=(xa, ya, za, ),
50                          interval=1,
51                          blit=True)
52 plt.show() # show the figure
```

WiFi with Matlab

Matlab provides functions for interacting with devices. The interface used in the following example is the USB interface which appears to be a COM port. It is the same COM port used for uploading programs.

The Arduino code from the previous lesson will be used. Reload the code with the WiFi accelerometer program if necessary. You will need to change the **delay()** at the bottom of the program. Set it to 100ms. This is because Matlab is a bit slow and the high data rate will overload Matlab.

```
loop() {
  int reg[6];
  int i;
  client = server.available();
  if(client) {
    Serial.println("Connected");
    while(client.connected()) {
      Wire.beginTransmission(0x1c);
      Wire.write(0x01);
      Wire.endTransmission(false);
      while(Wire.available() < 6) {
        delay(1);
      }
      for(i=0;i<6;i++)
        reg[i] = Wire.read();
      short x = (reg[0] << 8) | reg[1];
      short y = (reg[2] << 8) | reg[3];
      short z = (reg[4] << 8) | reg[5];
      x = x >> 2;
      y = y >> 2;
      z = z >> 2;
      float gx = x / 4095.0;
      float gy = y / 4095.0;
      float gz = z / 4095.0;
      client.print(gx,2);
      client.print(",");
      client.print(gy,2);
      client.print(",");
      client.println(gz,2);
      delay(100);
    }
  }
}
```

Accelerometer Code

WiFi with Matlab

First, three arrays will be created to hold the data for each axis of the accelerometer. A figure will be created to display three plots. Next, variable `t` will be created and be the network object. A tcp-ip connection is being created. The first argument is the Experimenters Kit IP address. The second argument is the port number. The third argument indicates a network connection and the last argument specifies the program operates as a client. `fopen(t)` connects to the Experimenters Kit.

An infinite loop is created with the `while 1`. A `for` loop is used to collect 50 data samples and fill the arrays.

```
xa = zeros(50,1);
ya = zeros(50,1);
za = zeros(50,1);

figure;
t = tcpip('192.168.4.1',80,'NetworkRole','client');
fopen(t);
while 1
    for b=1:50
        p = fscanf(t,'%e %e %e');
        xa(b) = p(1);
        ya(b) = p(2);
        za(b) = p(3);
    end
    subplot(3,1,1);
    plot(xa);
    title('Accel X');
    axis([1,50,-8200,8200]);
    subplot(3,1,2);
    plot(ya);
    title('Accel Y');
    axis([1,50,-8200,8200]);
    subplot(3,1,3);
    plot(za);
    title('Accel Z');
    axis([1,50,-8200,8200]);
    drawnow;
end
```

WiFi with Matlab

After the 50 samples are collected, the data is plotted in three separate plots. `drawnow` is executed to update the display.

To stop the Matlab code, click on the Pause button. Then click on the Quit Debugging button. In the command window enter **`fclose(t)`** and press enter. This properly stops the code and closes the network connection. If this is not done, the program cannot be rerun. If an error indicating the network connection is not available, restart Matlab.

```
xa = zeros(50,1);
ya = zeros(50,1);
za = zeros(50,1);

figure;
t = tcpip('192.168.4.1',80,'NetworkRole','client');
fopen(t);
while 1
    for b=1:50
        p = fscanf(t,'%e %e %e');
        xa(b) = p(1);
        ya(b) = p(2);
        za(b) = p(3);
    end
    subplot(3,1,1);
    plot(xa);
    title('Accel X');
    axis([1,50,-8200,8200]);
    subplot(3,1,2);
    plot(ya);
    title('Accel Y');
    axis([1,50,-8200,8200]);
    subplot(3,1,3);
    plot(za);
    title('Accel Z');
    axis([1,50,-8200,8200]);
    drawnow;
end
```

End
