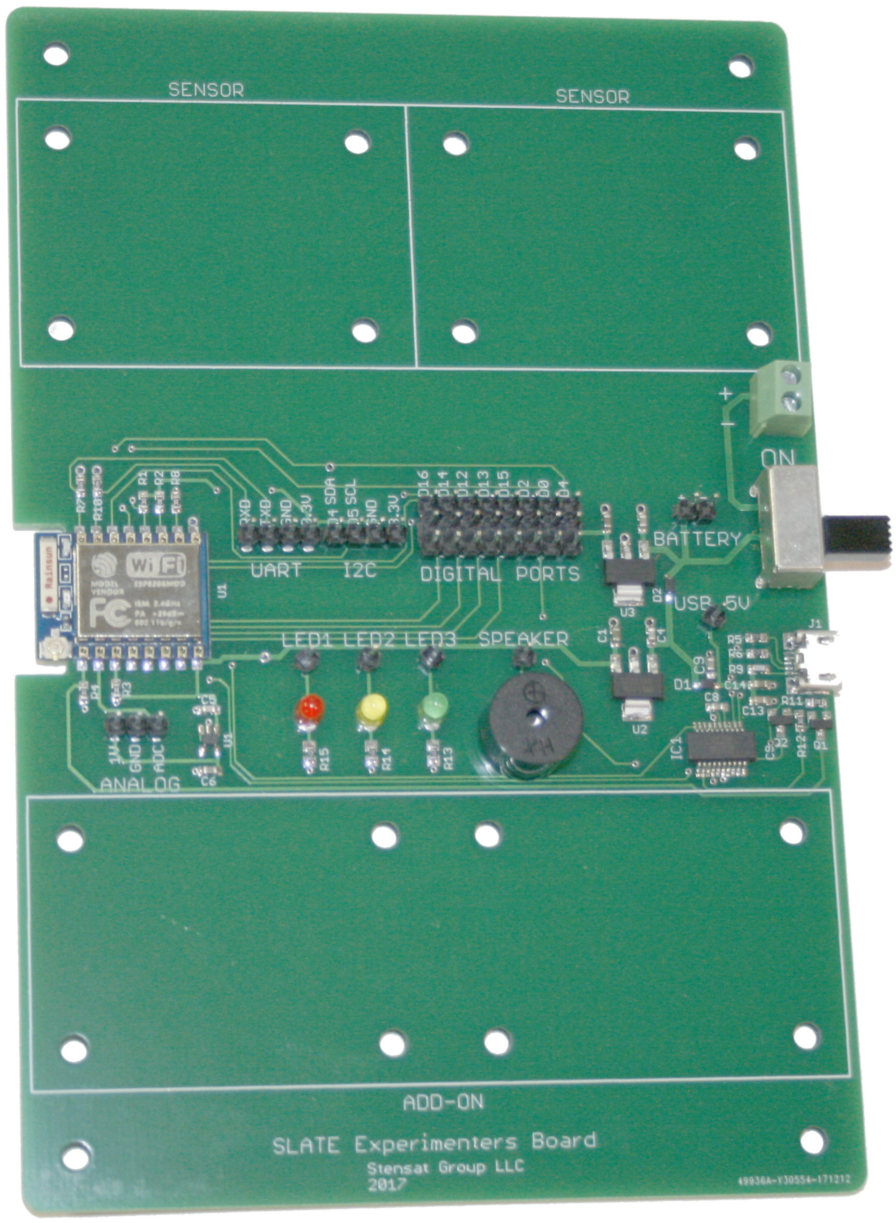
Sten-SLATE ESP Kit

WiFi

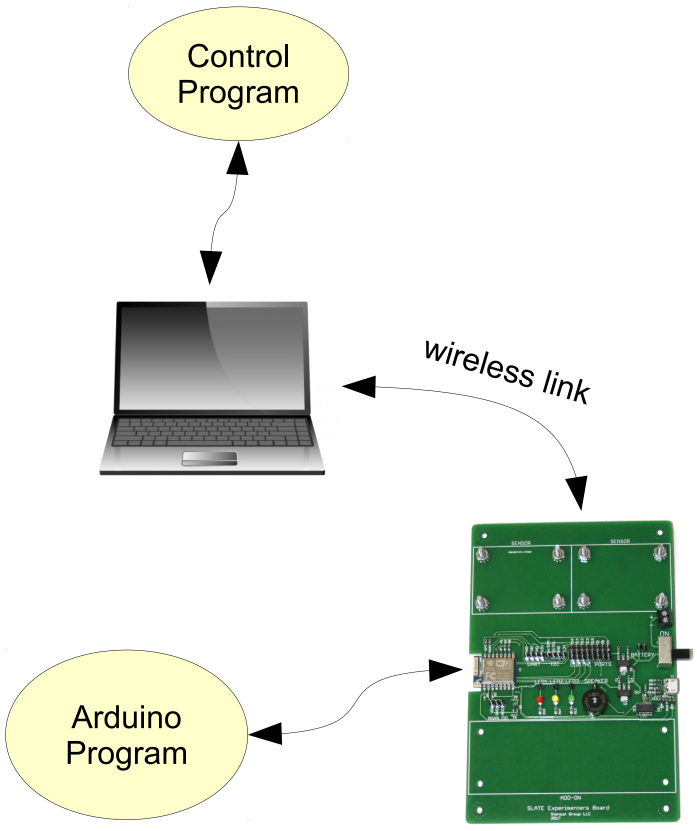


Stensat Group LLC,

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.

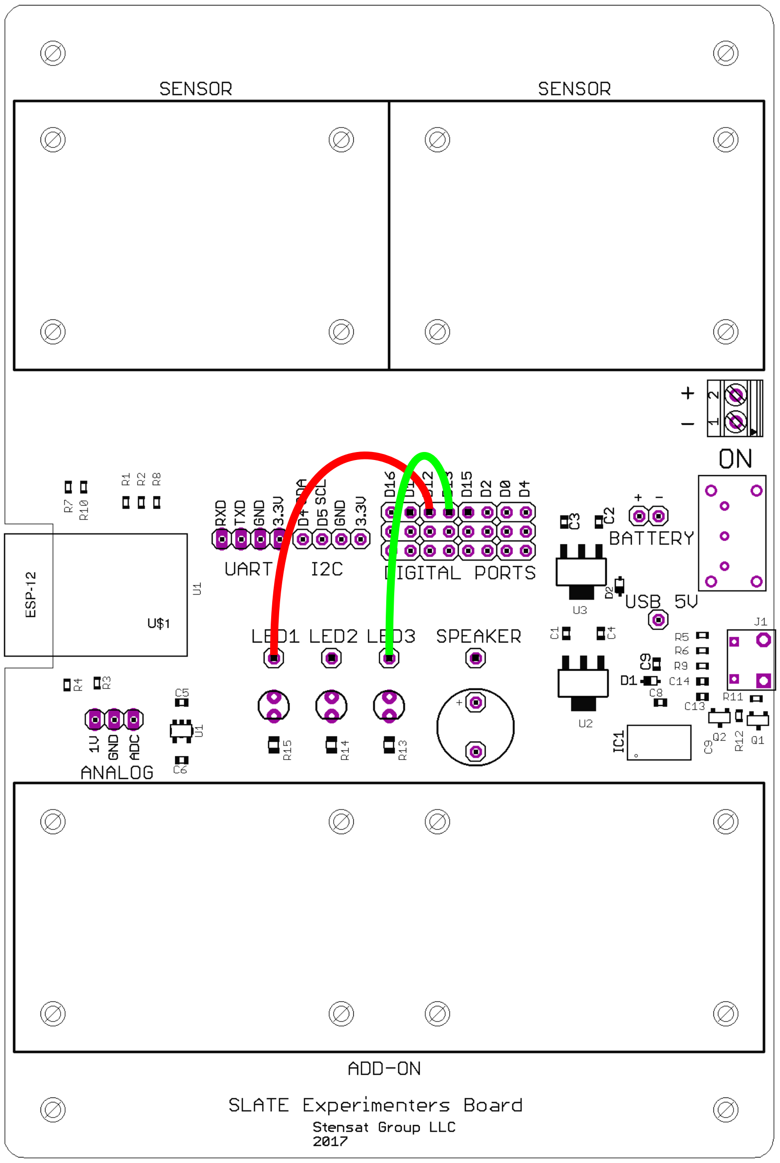
- **TCP** – is Transmission Control Protocol. This is one of the main network protocols used by any device on any WiFi network or the internet. The protocol enables two devices to establish a connection to each other and exchange data. The protocol guarantees delivery of data and that the data is delivered in the same order sent. The sender sends a data packet, when the receivers 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 guaranteere 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 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 tuplet type value.

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

Load the SLATE board with the software and let is 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

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.

Arduino Program

```
#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 is 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()
```