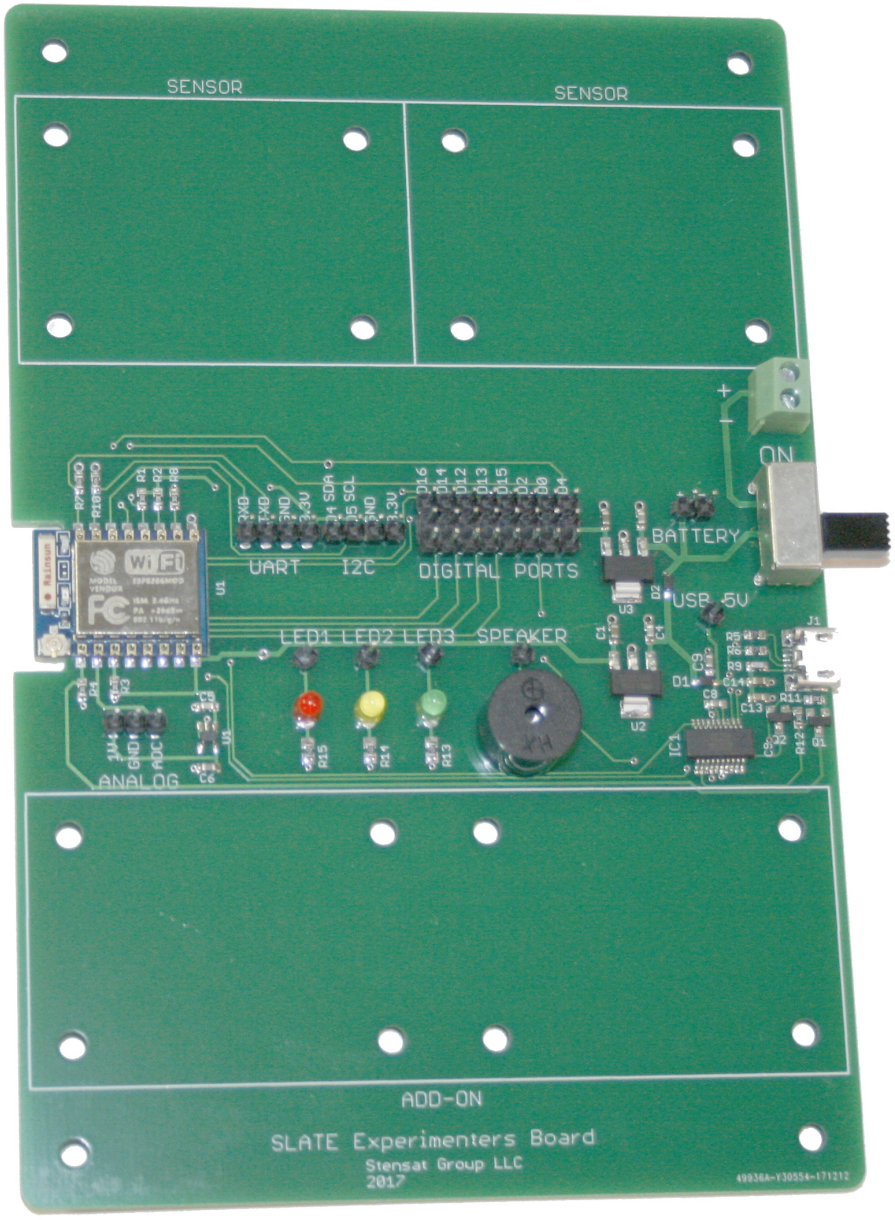
Sten-SLATE ESP Kit

Inputs



Serial Input

The STENSLATE supports different types of inputs, in this lesson, the serial input will be explained. The serial input allows information to be sent to the STENSALTE through the USB port.

In the first example, any byte received through the serial interface is sent back out the serial interface. The program basically echos all bytes received. Compile and upload the program. Give it the name **serial_echo**. Open the Serial Monitor window. In the top text entry, enter a sentence and press the Enter key. What was typed in should appear in the large text area below.

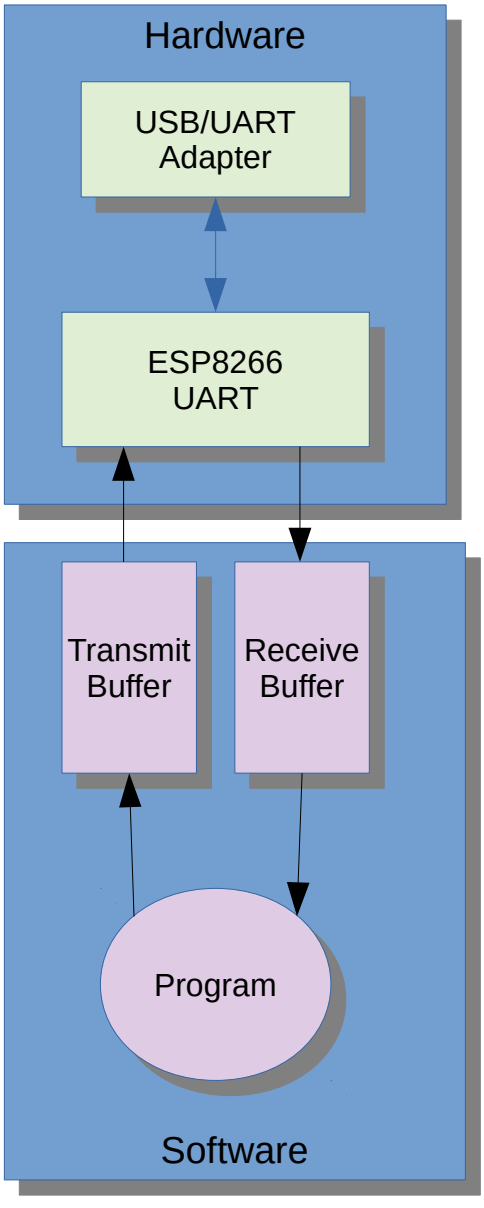
```
void setup() {
  Serial.begin(115200);
}

void loop() {
  if(Serial.available() > 0) {
    int a = Serial.read();
    Serial.write(a);
  }
}
```

Example 1

The **loop()** executes repeatedly. The function **Serial.available()** checks to see if any bytes have been received. It will return the number of bytes received. The function **Serial.read()** returns what byte was received. Only one byte is returned at a time. If multiple bytes are available, the **Serial.read()** function needs to be executed for each of the multiple bytes.

The STENSLATE uses a USB to UART adapter to convert the USB port to a serial interface that the processor uses for communications with the USB port. The Serial library has software that works in the background managing buffers. When bytes are received through the serial interface, they are stored in the receive buffer. Up to 64 bytes can be stored in the receive buffer. The **Serial.available()** function checks to see how many bytes are in the receive buffer and returns the number of bytes in the buffer. If no bytes are available, **Serial.available()** returns zero. **Serial.read()** will return the first byte received in the buffer, if there are no bytes in the buffer, **Serial.read()** will return some random value. **Serial.read()** does not wait for a byte to be received. This is why **Serial.availabl()** is called to check is any bytes have been received.



Serial Input

The next program will show how to receive multiple bytes and store them in an array. This is useful if text is being sent that needs to be parsed to perform an operation.

```
char buf[64];

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

void loop() {
  if(Serial.available() > 0) {
    bzero(buf,64);
    Serial.readBytesUntil('\n',buf,64);
    Serial.println(buf);
  }
}
```

Two new functions are shown. First is **bzero()**. This function clears the **char** array **buf** by setting all the bytes to zero. This is performed because the function **Serial.readBytesUntil()** only fills the array with the bytes received. Old bytes previously received will not get over written if the function receives less bytes than previously received.

Serial.readBytesUntil() function has three arguments. The first argument specifies the terminating byte value. In the example, it is the control code for a line feed. For carriage return, the control code is **'\r'**.

Compile and upload the program. Open the serial monitor. At the bottom to the left of the baud rate setting which should be at 115200, select **Newline**. With this selected, every time you enter text at the top and press enter, the line feed character is added to the end of the text. Enter some text and observe the results. The same text should have appeared immediately. Change the **Newline** setting to **Carriage return**. Enter the text again and notice the delay in the text appearing below. This is due to the **Serial.readBytesUntil()** function timing out. By default, if the terminating byte is not received within a second of the function call, it will return with whatever bytes it has received.

There is a built in timeout for the **Serial.readBytesUntil()** function and the default it 1 second. The time out can be adjusted using the **Serial.setTimeout()** function. The argument is time in milliseconds. The **Serial.setTimeout()** function must be placed after the **Serial.begin()** function. Modify the above code and add the **Serial.setTimeout()** function with different delays and see how it works.

If more than 64 bytes are sent to the serial port at a time, the **Serial.readBytesUntil()** will take in the 64 bytes and return leaving the reset in the receive buffer to be read later. This avoids buffer overruns which can cause the software to crash.

Parsing Byte Array

Now that a byte array can be received, the byte array can be parsed using the **sscanf()** function. The **sscanf()** function requires the **string.h** include file.

```
#include <string.h>
char buf[64];

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

void loop() {
  int a,b;
  if(Serial.available() > 0) {
    bzero(buf,64);
    Serial.readBytesUntil('\n',buf,64);
    sscanf(buf,"%d %d",&a,&b);
    Serial.print(a);
    Serial.print(" ");
    Serial.println(b);
  }
}
```

The first argument for **sscanf()** is the array to be scanned which is buf. The second argument is the format of the byte array which is expected to be a text string. Shown in the code are two integers separated by a space. An integer is indicated by **%d**. The remaining arguments is the list of variables to be filled by **sscanf()** with the values in the text string. Notice the **&** sign in front of the variable name. This is used to pass the location of the variable in memory to the **sscanf()** function. The number of **%d** must match the number of variables listed.

Try the program and enter two integer values separated by a space in the serial monitor window. You should have the values entered displayed below. try separating the two numbers with a coma? The output should not match. This is because, the format in the second argument must be matched. You can use comas or colons or other character to separate the values.

You can even input floating point values by replacing **%d** with **%f**. try it and do not forget to change the variables from **int** to **float**.