Lab 4 - Asynchronous Serial Communications

Part 1 - Software Loopback

In serial communications one of the important tools we have that allows us to verify the communications channel is working properly is the loopback. A loopback is just what it sounds like; it loops the communications back to where it came from. Loopbacks can occur in many places along a communications path but they are use at the point where there is assumed to be some problem in the path. Loopback can be either software based or hardware based. When implemented in hardware they can simply be a wire that redirects an output signal to an input. When implemented in software they transmit any character received.

In this section we will create a software loopback that loops all incoming data on the targets primary serial port back to from where it came.

Upload the following code to your target board:

```
void setup()
{
  Serial.begin(115200);
}
void loop()
{
  if (Serial.available()>0)       // have I received any data?
  {
    Serial.write(Serial.read()); // read it and send it back?
    delay(250);                  // delay between transmitting characters?
  }
}
```

Now open the serial monitor. Anything you type in to the serial monitor is looped back and displayed in the terminal.

Analysis:

This code has three functions you may not have seen before.

Serial.available()

This function returns the number of characters that have been received that have not yet been read. How it's used here is that if it returns a value greater than zero it will make the "if statement" condition true and will enter and run the code inside the "if statement" braces { }.

Serial.read()

Returns one character from the serial port input buffer.

Serial.write()

This treats the parameter passed to it as a character instead of printing the numerical value in decimal.

Check off

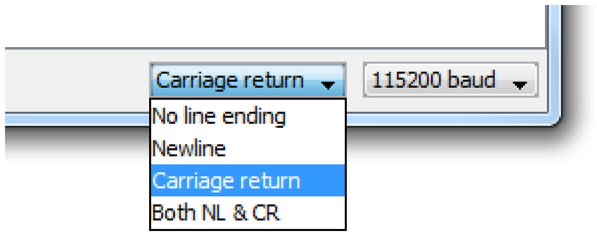
Call the instructor over for check off when you get it working.

Part 2 - Loopback with Value

In this section we modify the software loopback so that it stores the value received in a variable. This allows us to do some analysis on the variable and examine its contents a bit more.

```
void setup()
{
  Serial.begin(115200);
}
void loop()
{
  uint8_t received;
  if (Serial.available()>0)//have I received any data
  {
    received = Serial.read();
    Serial.print("Received: \""); // Backslash used here \"\"\"
    Serial.write(received);
    Serial.print("\" base 10 = ");
    Serial.print(received, DEC);    // DEC constant implies base 10
    Serial.print(", base 16 = ");
    Serial.print(received, HEX);    // HEX constant implies base 16
    Serial.print(", base 8: ");
    Serial.print(received, OCT);    // OCT constant implies base 8
    Serial.print(", base 2: ");
    Serial.println(received, BIN);    // BIN constant implies base 2
  }
}
```

Experiment with different Line endings to see what is sent to your board with each type.



Analysis

The only new concept is this program is the use of the backslash (\) character inside of a literal string to escape the detection of the end of the string so that the " character can be printed.

Check off

Call the instructor over for check off. Be prepared to explain what the difference is between the different line endings and difference between capital and lower case of the same letter in base 16.

Part 3 - Hardware Loopback

In this section we are creating software that passes data through from our primary serial port out to a secondary serial port and vice versa. Upload the following code to your board, use wire to make a hardware loopback between the RX and TX pins for port Serial0 on your Fubarino then open the serial monitor.

```
//Serial  Fubarino SD USB, Fubarino Mini USB
//Serial0 Fubarino SD tx  8 rx  9, Fubarino Mini tx 17 rx 18
//Serial1 Fubarino SD tx 28 rx 29, Fubarino Mini tx 26 rx 25
void setup()
{
  Serial.begin(115200);
  Serial0.begin(115200);
}
void loop()
{
  if (Serial.available()>0)
  {
    Serial0.write(Serial.read());
  }
  if (Serial0.available()>0)
  {
    Serial.write(Serial0.read());
  }
}
```

Everything you type gets sent back by the board to the window again but if you pull the jumper the loop is broken and you don't get anything back.

Analysis:

In this program we are using will be using two serial ports in the same program:

```
  Serial.begin(115200);
  Serial0.begin(115200);
```

From the users perspective (you) these functions are the same. Internally however, the code that actually controls the communications is different. The serial code associated with the USB serial ports is quite different than that of boards that do not have it built in but instead use USB to serial converters like the FTDI chip.

Check off

Call the instructor over for check off when you get it working.

Part 4 - Communication Between Boards

In this section we will connect establish communications between two boards using serial communications. This is your first step into machine-to-machine communications. In this example we will not be causing the machines to do anything special but instead just using them as a relay for signals from on PC to another.

Instructions

1. Get another student to help you.

2. Remove all power from the boards, including USB connections.

3. You are going to need to connect three signals between your board and theirs. Connect TX pin on your board to the RX pin on theirs and the TX pin on theirs to RX pin on yours. Finally connect a ground signal between both boards. The ground signal is needed to establish a common reference voltage since the boards are powered by computers with isolated power supplies (the boards are floating relative to each other).

4. Reconnect power and with the same sketch from above now what you send shows up in their Serial monitor and the other way around.

5. Play around with disconnecting the Serial wires and see which direction of communication is broken.

Check off

Call the instructor over for a check off when you get it working.

Part 5 - Visualizing Serial

In this section we will look at the physical manifestation of serial data on an oscilloscope by probing the transmit line of the UART on your board while continuously transmitting a single ASCII character. There is however an issue with looking at serial data that may not be immediately apparent. Serial data is a simple not periodic signal like a sinusoid or square wave. The lack of pure periodicity makes triggering on the oscilloscope unpredictable because the oscilloscope does not know which edge to trigger on. To resolve this issue we introduce a signal that is purely for triggering. In this case we will use pin 0 for the trigger signal. For your setup you may wish to use a different pin depending on what you have connected to your board.

Load the code onto your board

```
uint16_t TRIGGER = 0;  // Use pin 0 for the o-scope trigger

void setup()
{
  Serial0.begin(115200);
  pinMode(TRIGGER,OUTPUT);
}
void loop()
{
  digitalWrite(TRIGGER,HIGH); // Create a rising edge on trigger pin
  Serial0.write('5'); //'5' is hex value 0x35
  digitalWrite(TRIGGER,LOW); // Create a falling edge on trigger pin
  Serial0.write('3'); //'3' is hex value 0x33
}
```

Connect an oscilloscope channel 1 input to the output of UART transmit pin (Fubarino SD pin 9) and oscilloscope channel 2 input to the trigger pin.

Analysis

We have seen and printed strings of characters using double quotes such as "hello" but in this example we are printing a single character such as '5'. The difference between a string and a single character is how the value is stored in memory. However, this lab lets us understand aspects of using an oscilloscope in ways that elude most users.

Check off

Use edge triggering to switch back and forth between the ASCII value for '5' and '3'. Draw the waveform on a piece of paper (hint, align one bit cm on the oscilloscope graticule so that it is easy to pick out bits that do not change state). Next convert the waveform levels to bit values (0 / 1) then use these bits to convert the binary value on the oscilloscope to a hexadecimal value. Compare the hexadecimal value to the expected ASCII values for the transmitted characters.

Measure the period for transmitting one byte on the oscilloscope including the start and stop bits.

Call the instructor over for a check off, be able to point out the start/stop bit and show where the 3 and 5 are on the scope.

Part 6: Change Transmitted Values

Start with the code from part 5 and change the characters that are sent to 'a' and 'A'. Also change the transmission bit rate from 115200 to 9600 bits per second. Measure the period for transmitting one byte on the oscilloscope including the start and stop bits.

Check off

Call the instructor over for a check off, you must be able to point out the start/stop bit. Be prepared to point out the difference between the two signals 'a' and 'A'.

Part 7 - Serial Receive

In this section we will start writing some programs that scan the serial input to our target board. This will give us the ability to make decisions based on what is being sent to the board over the serial port and take action based on what is received. The actions that can be done are as diverse as the possible programs we can write.

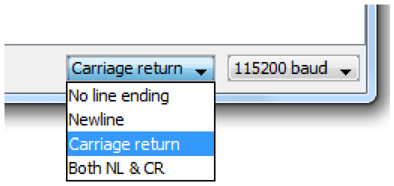
The Arduino IDE and the Wiring core automatically add serial interrupts and a buffer to store input until you call for it in your code. This makes it very easy to create sophisticated programs as we are about to do quite quickly.

Upload the following code to your board.

```
char buff[0x20];  // char string to store our incoming data
uint8_t ctr = 0;       // Store the current position in the buff array
uint8_t ch;            // Store the last character received from serial port

void setup()
{
  Serial.begin(115200);
  pinMode(PIN_LED1, OUTPUT);
}
void loop()
{
  if (Serial.available() > 0) // True if characters in the input buffer
  {
    ch = Serial.read();        //read one character from buffer
    if( ctr < sizeof(buff)) { // True if the ctr is less than buffer size
      buff[ctr++] = ch;        //add it to your buffer and increment ctr
    }
    if (ch == '\r')            // is received character is a '\r'
    {
      buff[ctr-1] = 0;         // replace '\r' with '\0' (string termination)
      ctr = 0;                 //reset the pointer
      Serial.print("Command: "); //print a string and stay on the same line
      Serial.println(buff); // print received followed by a new line
    }                          // end found carriage return
  }                            // end serial was available
}
```

Set the Serial Monitor to end lines with a Carriage Return.
Now anything you type and press Send (or enter) gets returned prefaced with "Command: "



Analysis:

```
ctr++
```

The ++ operator increments a value by one. If the operator precedes the variable then the variable is incremented before evaluated. However, if the variable precedes the operator the variable is evaluated first then it is operated on. The -- operator is similar to the ++ operators and works the same except that is decrements the variable instead of incrementing it. An easy way to see the difference between pre and post increment and decrement operators is with some sample code:

```
uint8_t var = 5;
Serial.println(var++,DEC); //prints 5 var =6
Serial.println(++var,DEC); //prints 7 var =7
Serial.println(var--,DEC); //prints 7 var =6
Serial.println(--var,DEC); //prints 5 var =5
```

```
Serial.println(buff)
```

Uses the Serial print in a way you have not seen yet instead if passing it a string like "Hello" you are giving it a location at which the string is stored, in this case the beginning of the buff array.

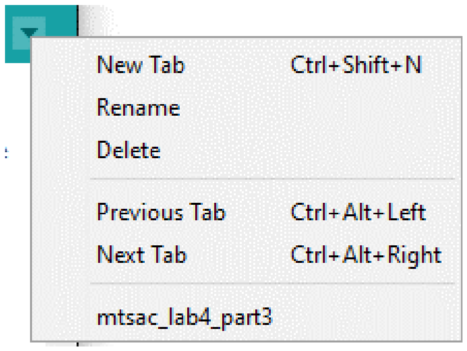
Check off

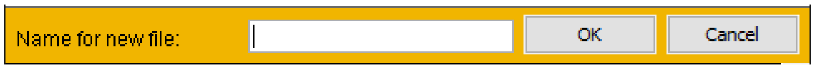
Run the above program. Open the serial and verify its operation by typing in a few strings and examining the output.
Call the instructor over for a check. You should be able to explain the function of the ctr variable.

Part 8 - Serial Receive Function

In this section we will take what was done in the previous section and turn it into a function for the purpose of information hiding. One way to decide on how to make a function is when you have a piece of complex code that is complete. This code, now not needing further refinement, can be placed into a function that is called over and over thus effectively hiding the complexity of the code from the calling point. This hiding of complexity can be viewed from the perspective of driving a car. You needed not look at and study the stroke of the engine in order to utilize its functionality.

In Arduino IDE we can have several files with code in the same sketch at the same time. To add a new piece of code to our sketch click the down arrow icon on the blue bar and select "New Tab" option.



This opens 

Type receiveCode and hit OK

This creates a new file in your project folder that compiles with your other code.

In the new file create the function

```
uint8_t receive_function(char* buff, uint8_t sizevar)
{
  static uint8_t ctr = 0;      // store the current position in the buff array
  uint8_t ch;                  // store the last character received
  if (Serial.available() > 0) // true when characters in serial input buffer
  {
    ch = Serial.read();        // store character from buffer in ch
    if( ctr < sizevar) {       // if the ctr is less than your buffer yet
      buff[ctr++] = ch;        // add it to your buffer
    }
    if (ch == '\r')            // if that character is a carriage return
    {
      buff[ctr-1] = 0;         // replace '\r' with '\0' (string termination)
      ctr = 0;                 // reset the pointer
      Serial.print("Command: "); // print a string and stay on the same line
      Serial.println(buff); // print received followed by a new line
      return 1;
    }
    else
      return 0;
  }                            //end serial was available
  return 0;
}
```

This should look familiar it is the receive code converted to a function that only returns 1 when a command is received otherwise it returns 0.

Now change the main code to the following to utilize the new function.

```
char command[0x20];

void setup()
{
  Serial.begin(115200);
  pinMode(PIN_LED1, OUTPUT);
}
void loop()
{
  receive_function(command,sizeof(command));
}
```

Verify that the program functions the same way as before

Check off

Call the instructor over for a check off.

Part 9 - Serial Command

In this section we add the ability to discriminate between commands sent to the board. We start out by detecting commands for turning on and off the default LED on the motherboard. Additionally, we can detect the word "hello" and print a reply.

```
void loop()
{
  if (receive_function(command,sizeof(command)))
  {
    if (strcmp(command, "hello") == 0)     // Compare received string
                                           // to the string "hello"
    {
      Serial.println("Back at ya!");       // if they match print out reply
    }
    else if (strcmp(command, "on") == 0)        // compare to "on"
    {
      digitalWrite(PIN_LED1, HIGH);        // if they match turn the led on
    }
    else if (strcmp(command, "off") == 0) // compare to "off"
    {
      digitalWrite(PIN_LED1, LOW);         // if they match turn the led off
    }
    else                                   // none of the ifs above were true
    {
      Serial.println("Not Recognized");
    }
  }
}
```

Analysis

```
strcmp ( const char * str1, const char * str2 );
```

The strcmp() compares two strings, if their values match the function returns the value 0, if they do not match the function returns a non-zero value. Here it is being used to compare the string in the command array with commands you want the board to respond to.

Check off

Call the instructor over for a check off when you get it working. Be prepared to demonstrate on, off, hello and bad commands.

Part 10 - Serial Command add one

random(min, max) is a built in function that returns a pseudo-random integer value number between min and max. We can utilize this function to make the "hello" command return pseudo-random responses when the board is queried.

A pseudo-random sequence is hard to predict but it would be the same every time you reset the board. The start position of the sequence can be changed with randomSeed(seed) but if you hard code in the seed value, your code will still return the same series on every reset. You would need to reseed it with a value from an external source (such as an unconnected analog pin) so that it will not have a fixed value every time.

Add a command to your code that responds to "hello" with three different strings depending on the value of a random(min, max) number.

Analysis:

You don't need to use randomSeed(seed) just use random(min, max) for your command.

Check off

Call the instructor over for a check off when you get it working.

Assigned Project (Homework)

Create a "rock paper scissors lizard spock" game between your target board and you the human player.

http://www.youtube.com/watch?v=iapcKVn7DdY

Use the serial port to prompt the user for a choice of "rock, paper, scissors, lizard or spock". The game then should use the random() to make a choose for the computer. Finally, the computer reports the result of who is the winner. (Hint: if you're having trouble implementing "rock paper scissors lizard spock" you can start with a simpler version of "rock paper scissors").

https://en.wikipedia.org/wiki/Rock%E2%80%93paper%E2%80%93scissors

Grading:

Correctly working rock paper scissors lizard spock: full credit 10/10 points

Partially working rock paper scissors lizard spock with known issues: 9/10 points

Correctly working rock, paper scissor: 8/10 points.

Partially working rock paper scissors with known issues: 7/10 points

Nothing working with comments: 5/10 points

Not attempted: 0/10 points

Appendix: random()

The following is derived from this web page but modified for this class:

https://www.arduino.cc/reference/en/language/functions/random-numbers/random/

Description

The random function generates pseudo-random numbers.

Syntax

```
random(max)
random(min, max)
```

Parameters

`min` - lower bound of the random value, inclusive (optional)

`max` - upper bound of the random value, exclusive

Returns

A random number between min and max-1 (`long`).

Example Code

The code generates random numbers and displays them.

```
uint3_t randNumber;

void setup(){
  Serial.begin(9600);

  // if analog input pin 0 is unconnected, random analog
  // noise will cause the call to randomSeed() to generate
  // different seed numbers each time the sketch runs.
  // randomSeed() will then shuffle the random function.
  randomSeed(analogRead(0));
}

void loop() {
  // print a random number from 0 to 299
  randNumber = random(300);
  Serial.println(randNumber);

  // print a random number from 10 to 19
  randNumber = random(10, 20);
  Serial.println(randNumber);

  delay(500);
}
```

Notes and Warnings

If it is important for a sequence of values generated by `random()` to differ, on subsequent executions of a sketch, use `randomSeed()` to initialize the random number generator with a fairly random input, such as `analogRead()` on an unconnected pin.

Conversely, it can occasionally be useful to use pseudo-random sequences that repeat exactly. This can be accomplished by calling `randomSeed()` with a fixed number, before starting the random sequence.

The `max` parameter should be chosen according to the data type of the variable in which the value is stored. In any case, the absolute maximum is bound to the `long` nature of the value generated (32 bit - 2,147,483,647). Setting `max` to a higher value won’t generate an error during compilation, but during sketch execution the numbers generated will not be as expected.