Communication Part 1: Communication between Computers

Up to this point in class, we have worked projects involving just our micro-controller---the micro-controller ran the show. But in many applications, it's more common to employ a couple of computers, each specializing in a different part of the project. In any application where one computer talks to another, the most common and easiest method to use is *serial communication*.

In this exercise, we are going to explore serial communication between your desktop computer and the micro-controller on your Arduino board. While there are many ways to create this communication link, our approach will be one that provides relatively quick and easy gratification. We will use *Processing* and its Serial communication libraries to create graphical interfaces to the Arduino. But, before we begin that effort, let's learn a little bit about serial communication.

Serial Communication

In serial communication, you send digital pulses (high or low voltages) one after another between computers. The computers involved have to agree on the process of sending and the meaning of these signals; the parameters of this agreement form what is called a *protocol*. In total, a *protocol* can be broken down into 5 *layers* of agreement:

Physical Layer: How the inputs and outputs of each device are connected to each other. How many connections do you need?

For serial communication, you typically use a 9-pin serial cable. Serial cables appear to have a lot of wires, but you only use three of them: one wire to send pulses (TX), one to receive pulses (RX), and a third as a common ground.

Electrical Layer: What voltage levels you send to represent the bits in your data?

Micro-controllers typically use the *TTL* protocol where pulses are either 5 volts or 0 volts. Unfortunately this is not the same arrangement used in the typical serial protocol, *RS-232*, but, of course, there are ways around this.

Logical Layer: Does an increase in voltage represent a 1 or a 0?

With *true* logic, a positive pulse indicates a 1, whereas, in *inverted* logic, it indicates a 0. Again, TTL and RS-232 differ on this point, with TTL using true logic and RS-232 using inverted.

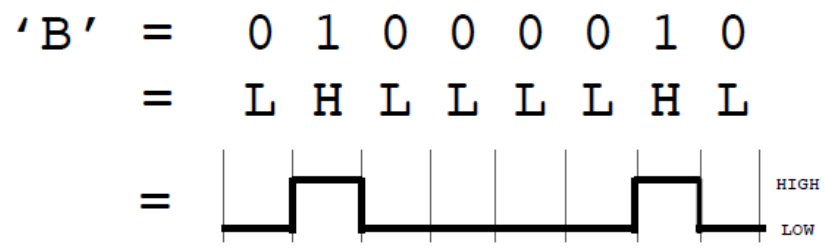
Data Layer: What is the timing of the bits? What is the *packet size*, in other words, are the bits read in groups of 8, 9 or maybe 10 or more? Are there bits at the beginning and end of each group to punctuate and delineate each group?

Serial data is passed byte by byte from one device to another, that's why you frequently see 255 as the max value in a range on the Arduino (e.g., the max pulse width). In TTL, data is sent at 9600 bits per second, each byte contains eight bits preceded by a start bit and followed by a stop bit.

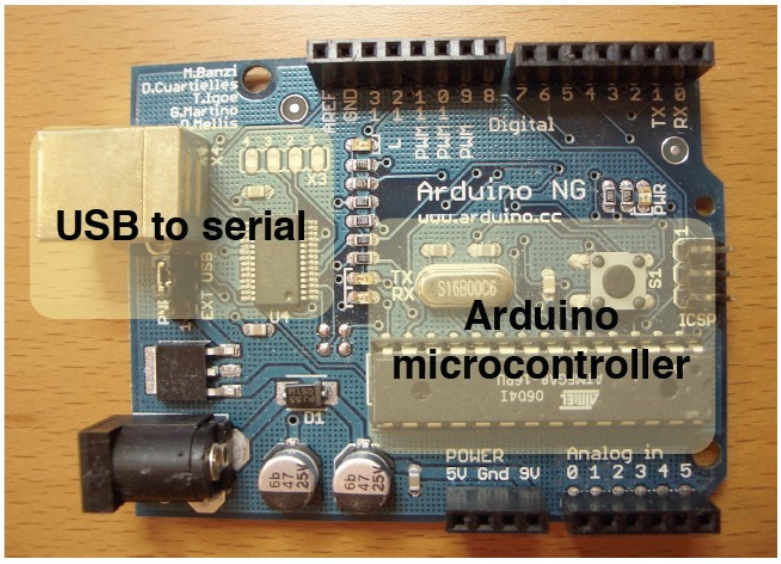
Application Layer: How are the groups of bits arranged into messages? What is the order in which messages have to be exchanged?

In the application layer of our setup, you send one byte from the PC to the Arduino and process it and the Arduino sends one byte back to the PC.

The picture below depicts the TTL encoding, minus the start and stop bits, of ASCII B.

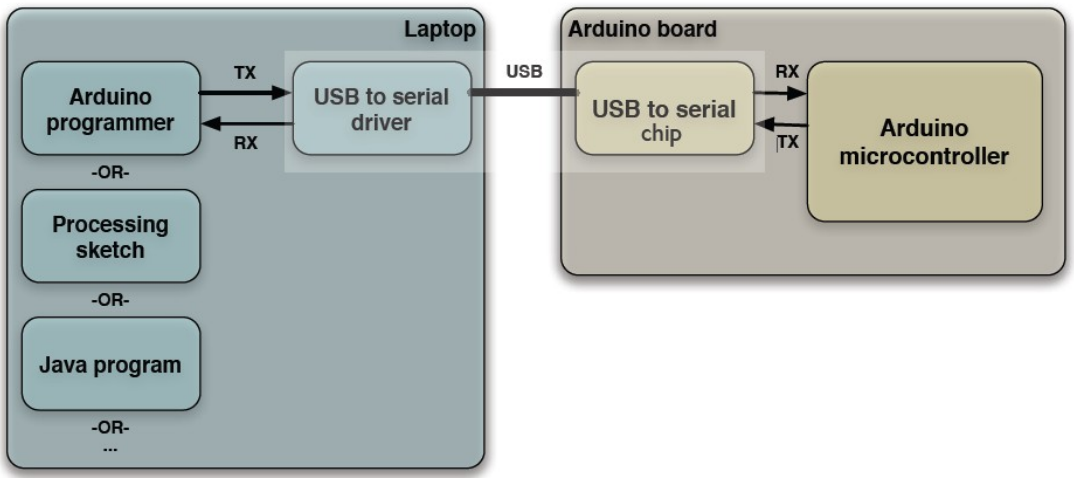


But wait, that's not really all there is in our setup. The 5-volt and 0-volt pulses from the Arduino don't really go directly to the PC over a serial cable; we don't use a serial cable to connect the 2 computers. Before leaving the Arduino board, they go to a serial-to-USB chip on the board that communicates using TTL serial on one side and USB on the other. We'll save the discussion of the USB protocol for another day.



All of the transactions that we've discussed above are transparent to you. The USB chip on your Arduino, presents itself to the operating system of your desktop computer as a

serial port and sends data through the USB connection at the rate you choose (e.g., 9600 bits per second). The desktop's USB controller only passes you the bytes you need.



We could use an oscilloscope see the data that is transmitted, but, again, we'll save that for another day.

Processing

The program receiving data from the Arduino can be written in any language that knows about serial ports, examples include: C/C++, Perl, PHP, Java, Max/MSP, Python, Visual Basic, and Processing. We will use processing because it is fun and easy.

Processing is a multimedia programming environment based on Java. It was made for designers, artists, and the like who want to create multimedia programs without knowing the gory details. Processing can made things happen like opening a network connection, connecting to an external device through a serial port, or controlling a camera though FireWire. It's also free and open source.

Programs in Processing are called *sketches* and the program (i.e., the sketch) plus the data it needs are saved in a folder with the same name as the sketch. (The same is true for Arduino programs.) The programming environment for Arduino is a derivative of the Processing programming environment so you will notice many similarities. Let's try it out!

The Processing development environment has already been installed on the computers in this room; you must use your Linux account to access it. To write your first program, do the following:

1. Login into your Linux account and open a Terminal Window. Start the Processing IDE by typing "processing" (do not include the quotes) at the prompt in your terminal window. It will probably take a long time to start the first time you try it.

2. If prompted, choose a location for the default folder for storing your Processing programs, I mean *sketches*.

3. Write a program in the Processing language.

Of course step 3 is the hard one so let's start with a few examples; use the Examples provided with the IDE. Select **File > Examples** in the IDE **(**does that sound familiar?), and select a few programs to look at. Run them, and, using the **Find in Reference** option on the **Help** menu, try to understand some of the built-in Processing methods. Processing can be thought of as a virtual machine built on top of java. The virtual machine consists of the IDE and the many built-in methods and libraries that comprise the Processing language. It is very similar to the Arduino programming language in some ways. One similarity is the IDE and another is the program structure. All processing programs have two main methods: setup() and draw(); these methods correspond exact to setup() and loop() in the Arduino programming language.

Try your hand at establishing a basic understanding of Processing before moving on. Ask your instructor, if you need help finding answers to your questions.

Let Arduino Talk

Let's start simple. Tether your Arduino to your desktop using the USB cable and run the program below on the Arduino. Be sure to turn on the Arduino's serial monitor while the program is running.

```
int val = 'h';

void setup() {
  Serial.begin(9600);
  Serial.println(val);
  Serial.println(val, DEC);  // print as an ASCII-encoded decimal
  Serial.println(val, HEX);  // print as an ASCII-encoded hexadecimal
  Serial.println(val, BIN);  // print as an ASCII-encoded binary
  Serial.println(val, BYTE); // print as a raw byte value

}
void loop () {
}
```

Notice that the *Serial* class supports serial communication; we've seen that before. Use the on-line reference to find out more about the methods available in the Serial class.

What does the output of the program above tell you about the default interpretation of each byte transmitted via serial communication?

Let's also try reading values from the serial port. Run the following program on the Arduino and enter values using the IDE serial monitor's input window. Make sure that you **set the baud rate in the serial monitor** to correspond to that used in the program.

```
/*
 * Serial Read Basic
 * -----------------
 * Blink the pin 13 LED if an 'H' is received over the serial port
 *
 *
 */

int ledPin = 13;   // select the pin for the LED
int val = 0;       // variable to store the data from the serial port

void setup() {
  pinMode(ledPin,OUTPUT);    // declare the LED's pin as output
  Serial.begin(19200);        // connect to the serial port
}

void loop () {
  // Serial.available() is a way to see if there's serial data
  // without pausing your code
  if( Serial.available() ) {
    val = Serial.read();      // read the serial port
    if( val == 'H' ) {        // if it's an 'H', blink the light
      digitalWrite(ledPin, HIGH);
      delay(1000);
      digitalWrite(ledPin, LOW);
    }
  }
}
```

Were you able to cause the built-in LED to blink by entering 'H'? You can also add an external LED to pin 13 so it's easier to see. Pin 13 has a built-in resistor, so you can place the long leg of the LED in pin 13 and the other leg in ground. Make sure that you understand the purpose of the method call, Serial.available().

Adding Processing

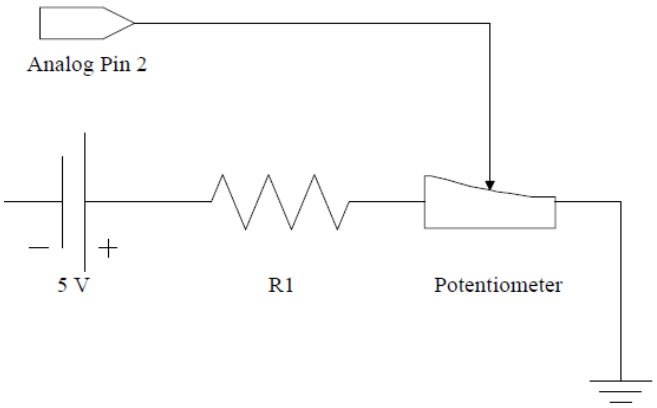
Now let's try something a little more interesting. We'll write one program for the Arduino and one for the desktop computer in Processing. In the Arduino program, we'll read a sensor or some input device and send the value read as a byte via serial protocol (well, sorta) to the desktop computer. In a Processing sketch, we'll read the byte and do something with it.

Let's look at the steps needed for serial communication in Processing; there are 4 of them and they are shown in red in the Processing program below.

```
/*
 * Simple Read
 *
 * Read data from the serial port and change the color of a circle
 * when a switch connected to an Arduino board is pressed and released.
 *
 */

import processing.serial.*;  // STEP 1: import the serial library

// STEP 2: set the portname; use the USB portname selected in the
// Arduino interface
String portname = "/dev/tty.usbserial-A4001qa8";  // or "COM8"

Serial port;  // STEP 3: Create the Serial object from the Serial class

int val=100;  // Data received from the serial port, with initial value

void setup()
{
  size(400, 400);
  colorMode(HSB, 255);
  ellipseMode(CENTER);   // draw from center out
  noStroke();
  frameRate(30);
  smooth();
  background(99);

  // STEP 4: Open the port the Arduino is connected to
  port = new Serial(this, portname, 19200);
}

void draw()
{
  if (port.available() > 0) {  // If data is available,
    val = port.read();         // read it and store it in val
  }
  // Draw the shape
  fill(val,255,255);  // we're in HSB mode, so first value is hue
  ellipse(width/2, height/2, 250,250);
}
```

We will use the program above in our next setup so you can load it into the Processing IDE. BE SURE TO CHANGE THE PORTNAME to be the one your Arduino is connected to. Please spend the time to look at and read about the methods available to you in the Processing serial library.

Meanwhile back at the Arduino, create the following configuration on your breadboard. In the schematic below, the 5V battery symbol represents the 5V pin on your Arduino, R1 can be any resistor, and the middle leg of the potentiometer is connected to analog pin 2.



After creating the configuration above, run the following program on the Arduino. At the same time, run the *SimpleRead* Processing sketch (specified above) on your desktop computer.

```
/*
 * PotSend
 *  Send analog values from pots or similar over the serial port
 *
 * Also see:
 * http://www.arduino.cc/en/Tutorial/Graph
 */

int potPin = 2;
int val;

void setup() {
  Serial.begin(19200);
  pinMode(potPin, INPUT);
}

void loop() {
  val = analogRead(potPin);
  val = val/4;
  Serial.print(val,BYTE);
  delay(100);
}
```

IMPORTANT NOTE: Only one program can use the serial port at a time so turn off the Arduino's Serial Monitor when connecting via Processing and vice-versa.

Did it work---could you control the hue of the circle on your desktop monitor by changing the resistance of the potentiometer on the Arduino? If not, ask your instructor for help. Look at the programs again and answer the questions below before proceeding.

1. Why was the value read on pin 2 divided by 4 before it was sent over the serial port?
2. Why is the value returned by the method call, port.available(), compared to 0 before attempting to read from the port?
3. What is the baud rate of the exchange between these computers?

Let Processing Talk

Let's have Processing talk instead of just listen. Put an LED on pin 13 of your Arduino (just as you did in **Making Arduino Talk** above). The first program below should be run on the Arduino, and the second should be run in the Processing IDE. There are a couple of new features in these programs:

1. The data received by the Arduino is buffered in a variable length String. This is probably not necessary in the current configuration but it may be useful in the future.
2. The Processing code prints a list of all serial ports available. The program then connects to the last port in the list. That may not be the correct port---be sure to check the listing of ports and change the designated connection port, if necessary.

Do you understand what should happen when you run the programs below? Be sure to position your mouse over the Processing graphics window (called the "canvas") when you press the mouse. Ask your instructor for help if things are not working as you think they should.

Arduino Program

```
int LED = 13;
int numberIn = 0;
char myString[100];

void setup() {
  Serial.begin(9600);
  pinMode(LED,OUTPUT);
}

void loop() {
  readSerialString(myString);
  numberIn = atoi(myString);
  if (numberIn >= 1) {
    digitalWrite(LED, HIGH);   // sets the LED on
  } else {
    digitalWrite(LED, LOW);    // sets the LED off
  }
}
```

```
void readSerialString(char *strArray) {
  int i = 0;
  if (!Serial.available()) {
    return;
  }
  while (Serial.available()) {
    strArray[i] = Serial.read();
    i++;
  }
  strArray[i]=0;
}
```

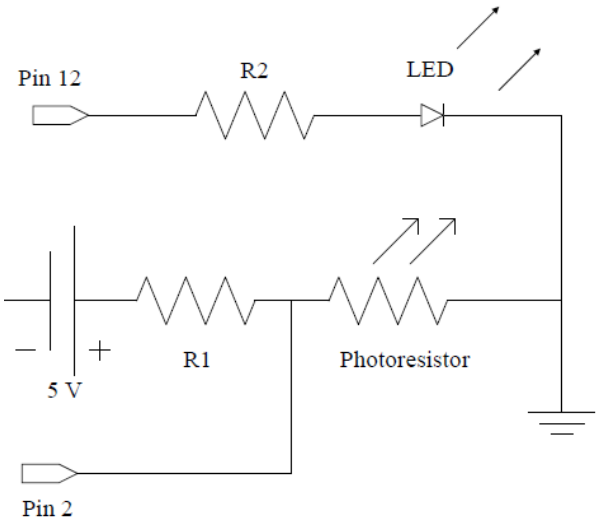
Processing Program

```
import processing.serial.*;

Serial port;  // Create object from Serial class
String valOut;  // Data transmitted on the serial port

void setup()
{
  println(Serial.list());  // only necessary on first connection
  String arduinoPort=Serial.list()[1];
  // Open the port the board is connected to
  port = new Serial(this, arduinoPort, 9600);
}

void draw() {
  println("press a mouse button and I'll blink");
  if(mousePressed) {
    valOut = "1";
  }
  else {
    valOut = "0";
  }
  port.write(valOut);
}
```

Holding a Conversation

Ok, your final challenge is to let both the Arduino and Processing talk. Does that sound like trouble, well it could be.

Create the following configuration on your breadboard. Place the LED very close to the photoresistor so that the LED can influence the resistance of the photoresistor.



The objective is to have Processing control the LED (just as it did in the last program), while the resistance of the photoresistor controls the hue of a circle in the Processing canvas (just as the potentiometer did in **Adding Processing**). In other words, you have to create new programs for both Processing and the Arduino using the previous examples as guides. (Note that we are using pin 12 rather than pin 13 for the LED.) Be prepared to demo your creation!