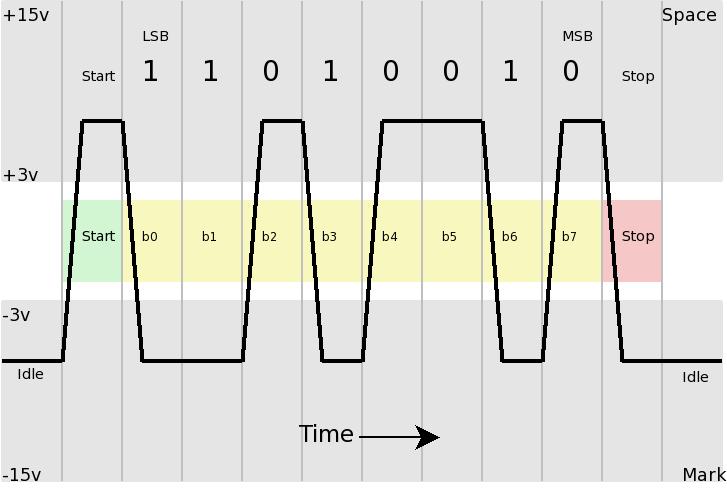
***Lab 4 – Communications on Embedded Systems***

## Learning Objectives

The purpose of this lab is to explore the various communications protocols available on modern microcontrollers. You’ve been using many protocols in this class already, but here we will expose them more explicitly. A communications protocol specifies the physical connection (wiring and signal levels) between two computers, as well as the timing (if applicable), and structure of the data packet. Below is a data packet from an RS232 data packet. RS232 (aka, a ‘serial port’) is one of the oldest serial protocols still in use, but the inconvenient physical specifications (+3 to +15V and -3 to -15V) usually mean we need a special converter or adapter cable to interface it to modern systems. Many micros have built-in UARTs to handle the conversion.



### Part 1 – USB

The Universal Serial Bus is by no means the simplest serial communications protocol, but it is ubiquitous, and easy to implement on most micros. For the Pi and Arduino, it is quite simple.

* 1. Connect the USB micro cable (red cable in this image) from your Arduino to one of the USB ports on your Pi. [The Pi will power the Arduino through the USB power pin.]



* 1. Type ls /dev/ttyUSB\* into the Pi terminal. If you get a response, your connection is good. This should tell you what number your USB connection to the Arduino is assigned (usually USB0).
  2. Download the Arduino and Python code from the Canvas site (both .zip files). Load and run the code on your Arduino. [You can hot swap the USB cables to download code to the Arduino. When you plug it back into the Pi, it will restart and run the code you last sent. You can see the RX pin on the Arduino flash as it sends the message from the code.]
  3. Use a probe to look at Arduino pins 0 and 1. Try to capture a data packet on your scope. It’s tricky to do, but there are special instruments (‘serial analyzers’) that do exactly that, for people who deal with the nitty-gritty details of communications protocols.
  4. Load and open the python code on the Pi. [Use SSH or VNC to send the code to the Pi.] Change the line in the seria\_test.py (note, no ‘l’ in the filename) code from /dev/ttyACM0 to whatever your Pi said in 1.2, probably /dev/ttyUSB0. Save the code. Restart your Arduino with the ***reset*** button. Run the Python code from the terminal window and watch the result.
  5. Explain the text in the terminal window. What is going on? What is the pattern? Why are there sometimes letters?

Text in the terminal window is printing one line being sent from the Arduino converted from hexadecimal to an integer, then printing one line as a hexadecimal. Since the Arduino is simply counting up from 1, and the hexadecimal output is read first but printed second, even numbers are printed as integers followed by odd numbers being printed in hexadecimal.

### Part 2 – Inter-Device Communication

There are many options, but SPI and I2C are the most common protocols for microcontrollers to communicate between each other, displays and some, more advanced sensors.

* 1. Read through the basics of SPI and I2C on the [SparkFun tutorial](https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial) here. Choose one and implement it on your Pi. Note that you do not have a 7-segment display or the specified ADC in your kit, but we do have a critical device that can use either SPI or I2C – the ***Pixy*** camera! You could also do this with another micro (Arduino, Teensy, etc.) It is your choice, but the Pi is probably the most useful for your project. [Yes, the Pixy can connect via USB, and it is advisable to first connect via USB, but use one of the other protocols as well. If you aren’t using a Pixy, you can interface to a different device, but I recommend at least trying it out.]
  2. Once your chosen protocol is up and running, connect it to your Pixy and start obtaining data from the Pixy on your Pi.

##### Signoff

I have witnessed team \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_’s Raspberry Pi communicate via SPI / I2C (circle one) with the Pixy camera.

Witness \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Notes on implementing i2c:

Either the [Sparkfun](https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial?_ga=2.222202392.1488586328.1519936939-1651301836.1474044382) or [Adafruit](https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c) tutorial should get you to the point where you can find your Pixy on the bus.

Once your address shows up, I recommend using smbus to receive data.

To get smbus:

1. sudo apt-get install -y python-smbus
2. sudo apt-get install -y i2c-tools

Here are [handy commands](http://wiki.erazor-zone.de/wiki:linux:python:smbus:doc), and some test code that parses the [Pixy packet](http://cmucam.org/projects/cmucam5/wiki/Porting_Guide) is on the canvas site.

### Part 3 – Pi and Arduino as Primary and Secondary

As we learned in Lab 3, the RPi is not very good at handling PWM signals, and is generally a little clunky when it comes to GPIO (general purpose input and output). Your robot will need a RPi (or some other SBC) to handle the vision processing, but you will probably also want to have an Arduino to handle simple digital sensors and to PWM your motors to appropriate speeds. This means that they will need to talk to each other, or at least the Pi to send commands to the Arduino to execute. [This setup has traditionally been dubbed ‘master/slave’ in the computer networking world. Obviously that terminology might cause offense, so an effort is being made to replace it with ‘primary/secondary.’ While we support the change, you should know that you will see many references to the older terminology.]

You can design this however your team thinks is most beneficial, but my recommendation is to create a system that does something similar to this:

* Pick a physical protocol to allow communications between the Pi and the Arduino. I2C and SPI are feasible. You could also run wires from the devices’ UART communications pins (RX and TX), but then you have to get more involved at the device level to make them communicate. USB is simplest, but using your USB ports means they aren’t available for other things, e.g., power and uploading code, as we learned in part one of this lab. You could also look at using Ethernet. The Pi has Ethernet ports, and Arduino Ethernet ‘shields’ are available for $14-20. You may change your mind after you complete the lab, and use something else for your project, but it may save you time in the long run to implement your final choice now.
* Establish a simple set of commands and the ability to send values along with them, from the Pi to the Arduino. For example “<MOT-CWO|50>” could tell the Arduino to run the motor in one direction at 50% duty cycle. “<MOT-CCW|25> would tell it to turn the motor the other direction at 25% duty cycle. You want to keep the command packet as simple and uniform as possible. The ‘O’ in the first command isn’t really necessary, but it makes that command the same length as the second one, which will be helpful for the Arduino to parse the command. The angle brackets and ‘|’ symbol are also not strictly necessary, but they make it easier for the Arduino to parse, and determine a value from a command. [Here](https://github.com/tttapa/Projects/blob/master/Arduino/Serial/Input%20Parsing/Command-array-of-values/Command-array-of-values.ino) is some sample Arduino code that may help parse the commands as they come in.
* I recommend using the motor driver from the Arduino kit to run one of the geared DC motors, also from the kit.
  1. Once your Arduino is running your motor as the Pi commands, get the signoff below.

##### Signoff

I have witnessed team \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_’s Raspberry Pi send commands to an Arduino, and the Arduino run a motor at varying speeds and directions.

Witness \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_