Microcontroller Lab 5

Reading

- http://www.ladyada.net/learn/arduino/lesson4.html (first part)
- http://www.learnpython.org/ (first few lessons)

In this lab we're going to learn about serial communication between your computer and your Arduino, learn a bit of the Python language and the Tk graphics toolkit, and make an oscilloscope.

Serial Communications

The USB cable from your Arduino to the computer can be used for communication of data as well as programming new sketches into your Arduino. Please read the first part of the Ladyada Tutorial (link).

- ☐ **Lab 5.1 - Serial Communications**

  Write a sketch which initializes the serial port for 9600 baud, prints a "Hello" message, and then outputs a message with a counter once per second after that. Test it using the Arduino serial monitor. The output should look something like this:

```
Hello World!
Seconds since start: 1
Seconds since start: 2
Seconds since start: 3
 ...
```

- ☐ **Lab 5.2 - Oscilloscope v1**

  Connect a function generator (either the built-in one on the breadboard or a separate one) to the A0 pin on your Arduino through a small resistor (1k or so). Ground the Arduino to the generator or breadboard.

  Write a sketch which reads the A0 pin using **analogRead()** and prints the output to the serial port in **loop()**.

  Set the function generator to generate a square wave of about 10Hz swinging between about 1V and 2V. What do you see in the serial monitor?
  (*Hint: unplug the USB cable to get the output to stop so you can look at it).*

  - Look at the waveform on the oscilloscope and sketch it
  - Cut/paste the list of numbers from the serial monitor into a spreadsheet or other plotting program and make a plot. Compare it with the oscilloscope picture. They should look quite similar.

Congratulations! You have made a digital oscilloscope. It's not very easy to use, though.

There are two glaring problems with your first oscilloscope. It sends data continuously rather than in convenient pieces, and the sampling rate is determined by how fast the data can be sent to the computer.

☐ Estimate the sampling rate. The "baud rate" is 9600, which is the number of bits per second sent. Each byte takes 8 bits (plus a start and stop bit) so the rate of sending bytes (characters) is about 1/10 of the baud rate.

☐ **Lab 5.3 - Oscilloscope v2**

Now we're going to make a new version which addresses the problems above.
Write a new sketch which does the following:

a. waits for a character to be received from the computer
b. sends 100 samples, collected at 1 sample per 1ms (use **delay()**).

NOTE: you must collect the samples in an array in one loop, then send them to the computer in a separate loop. Otherwise the collecting will be slowed by the data transmission as in Lab 5.2.

A couple of tips. To wait for one character from the computer, use this code:

```
while( Serial.read() == -1) { }
```

Serial.read() returns -1 if there is no character to read. The loop body is empty because there isn't anything to do while we're waiting.

Test it. Set the signal generator to 100Hz. You should see about 10 cycles in the waveform. It won't be exact because **delay()** is not a very accurate way of controlling the sample rate.

☐ **Lab 5.4 - Oscilloscope v3**

Now we're going to make the sample rate more precise, and adjustable.
Modify your sketch to use the TimerOne library as you did in Lab 2.9 (look back at the notes if you need to). The sample rate will be the parameter to **Timer.Initialize()** in uS. Note that the ADC is fairly slow, so values below about 100uS will begin to give poor results.

Set two constants at the top of the code:

```
const unsigned int sample_interval_us = 100;
const unsigned int record_length = 100;
```

Your sketch should use these to set the sampling interval and number of points.
It should also trigger only when it receives a character from the computer as in Lab 5.3

You will need to figure out a way to make the timer interrupt capture the specified number of samples in response to a trigger and then stop. The best way to do this is to declare a variable to count the number of samples recorded, and an array to store the samples in, like this:

```
int waveform[record_length];
volatile int num_samples;
```

("Volatile" is a modifier required for all variables shared between interrupt routines and the main program).

Then the timer interrupt looks something like this:

```
void timer_func() {
      if( num_samples < record_length)
            waveform[num_samples++] = analogRead( A0);
}
```

The main loop would wait for a character, and then set **num_samples = 0** to start data recording. Sampling is done when `(num_samples == record_length)`.

Test it at various sampling rates and various frequencies of the signal generator.

Python

To display the data from your oscilloscope on the computer you need to write some software for the computer which communicates with the Arduino and draws graphs of waveforms. There are lots of ways to do this, but few of them work easily on the various OS in common use (Windows, MacOS and Linux). A popular choice is the language Python. We're going to learn just enough python to make a simple oscilloscope display, but hopefully this will whet your appetite and encourage you to learn more.

Several important differences between Python and C:

- Variables don't have to be declared in python. First time you assign one, it is created.
- Variables can hold values of different types.
- If a variable contains an integer represented as a string, then you must use int() to convert it before doing arithmetic:

```
>>> a = "123"
>>> a+5
TypeError: cannot concatenate 'str' and 'int' objects...
```

```
>>> int(a)+5
128
```

- Structure is represented by indenting, so for example:

```
if a > 5:
    print “too big”
else:
    print “just right”
```

See http://www.python.org for complete information about Python. Note that we are using Python 2 here.

You should already have Python and pyserial installed...
see Installation Instructions or ask for help.

☐ **Lab 5.5 - Python Introduction - Serial Communication**

Our focus is on two areas: serial communications and graphics.
Load up the sketch from Lab 5.4 if it isn’t already in your Arduino.

Next download or the following python program: simple_serial.py

```
#
# simple python communications demo
# use with Lab 5.4 sketch

import serial
import time

# connect to the Arduino via the serial port
# change the serial port name as required
#  (to i.e. 'COM6' if on windows)
ser = serial.Serial( '/dev/ttyACM0', 9600, timeout=1)

# the Arduino reboots when you do this, so a delay is needed
print "Waiting for Arduino to reset..."
time.sleep( 3.0)

print "Sending trigger..."
ser.write( 't')
for i in range( 100):
     s = ser.readline()
     print s
```

Linux:

Use a text editor (emacs, for example) and save the file when you are done
To run the program, open a terminal and type:

```
$ python simple_serial.py
```

Windows:

You may use a text editor (such as Notepad or Emacs if you have it)
To run the program you can use the Python GUI (IDLE):

Start -> All Programs -> Python 2.7 -> Python GUI (IDLE)

From this environment you can load and edit programs, and run them by hitting F5 (or use the Run menu)

MacOS

You are on your own! Should work like linux.

Run the python program. If it works, you should see:

```
Waiting for Arduino to reset...
Sending trigger...
189

138

170

222

273
```

... 100 numbers

If you get an error message which ends with:

```
serial.serialutil.SerialException: could not open port
```

You are probably using the wrong port name. Fire up the Arduino software and notice the port name which appears in the bottom right corner of the screen.

For Linux (and perhaps MacOS) you may need some help with port permissions if you get an error saying you are not authorized to access the serial port.

☐ **Lab 5.6 - Python Introduction - Graphics**

For graphics display we will use the Tkinter, which is a simple set of python bindings for the Tcl/Tk graphics toolkit. Here is an example program: simple_graphics.py.

It draws a simulated oscilloscope trace.

```
#-------------------------------------------------------------
# simple_graphics.py : graphics demo using Tkinter
#-------------------------------------------------------------

from Tkinter import *            # import all names from Tkinter
import math

root = Tk()                      # create top-level Tk object

da = 4*math.pi/110
a = 0.0

# define a function to draw a simulated scope trace
def draw_trace():

       global a, da              # get access to these global variables
       w.delete(ALL)             # delete everything in the window
       # draw a frame around the canvas
       w.create_rectangle( 50, 50, 350, 350)

       for i in range( 100):            # plot 100 points
       x = 50 + 300.0*(i/100.0)                # calculate x coord
       y = math.sin(a) * 100 + 200             # calculate y coord
       if i > 1:
          w.create_line( x0, y0, x, y)         # draw a line
       x0 = x                                  # save last coords
       y0 = y
       a += da                                 # increment angle

       w.after( 100, draw_trace)  # call function again after 100ms
#--- end of function definition

# create a "Canvas" widget we can draw on as a child of the root object
w = Canvas( root, width=400, height=400)
w.pack()                         # display the canvas

draw_trace()                     # call draw_trace once to get started

mainloop()                       # start the main event loop
# we never get here... the main loop just looks for events
# and handles them.
```

Please note several things about Tkinter:

- Everything which appears on the screen is called a *widget*
- Widgets exist in a hierarchy rooted in a top level widget (**Tk()**)
- After the initial set-up, you must call **mainloop()**, a function which processes

events (mouse clicks, for example) and never returns. Any code which you would like to have executed while your application is running must be coded as a *call-back* function which is attached to an event.

Finally, we want to combine the last two labs to get data from the Arduino and display it on the computer screen.

☐ **Lab 5.8 - Oscilloscope with Display**

Try to combine the python code from the last two labs to make an oscilloscope display. Some hints:

- Add the line of code to initialize the serial port to the beginning of the program
- In the function **draw_trace()** add the part to send a trigger byte to the Arduino and retrieve the data points.

☐ **Lab 5.9 - Improving the performance**

The default clock for the on-board ADC on the Arduino is pretty slow. It is based on the microcontroller clock (16Mhz) divided by a *prescaler* value which is by default 128, so the ADC clock is (16MHz/128 = 125kHz). Since a conversion takes 13 ADC clocks, the conversion time is about 104uS. The ADC is specified to work up to 1Mhz, so we can set the prescale to 16 without much loss of performance.

Refer to the ATMega328P data sheet, starting on p 245. Three bits (ADCPS0..2) control the prescale, so we can set them to 16 according to the table on page 256 with the following code:

```
// set prescale to 16
ADCSRA |= _BV( ADPS2);
ADCSRA &= ~_BV( ADPS1);
ADCSRA &= ~_BV( ADPS0);
```

Add this to your **setup()** function. Notice that the ADC now works well down to a conversion time of about 13uS. Confirm this with the function generator.

☐ **Lab 5.10 - Controls**

It is a nuisance to download a new sketch each time you want to set the sample rate or other settings in your oscilloscope. Using the Python Tkinter toolkit you can create buttons, menus etc to allow the user to interact with the application while it is running. There really isn't time to develop all of this from scratch, but here is an example sketch

and example Python code which implements a few of these ideas. Study the code, and try it out.

Then, try to add some features of your own. How about adjustable vertical scale? How about a level-sensitive trigger?

- Lab 5.10 Arduino
- Lab 5.10 Python