Exercise 1: Intro to Arduino*

EENG350: Systems Exploration, Engineering, and Design Laboratory

Tyrone Vincent and Hisham Sager

Department of Electrical Engineering
Colorado School of Mines

Spring 2022

1 Introduction

The purpose of this exercise is to get familiar with an Arduino board, its hardware features and how to interact with it using a PC/laptop. These basic exercises will set up the foundation to be able build a much more complex system later.

Recall that a micro-controller is a small computer with a processor, memory and programmable input/output peripherals. You have been provided with an Arduino Uno Board that uses an ATmega328 microcontroller. Bookmark the pages with the technical specifications of the board as well as the datasheet of the microcontroller. Arduinos don't run a full operating system, but simply execute written code as their firmware interprets it. The Arduino board will be used to interface with sensors, motors and other devices.

Exercise Objectives:

- Use the Arduino IDE to download a program and receive messages via the serial port
- Write a program that utilizes analog and digital I/O
- Write a program that utilizes interrupts to read a rotary encoder
- Find information about Arduino programming and the ATmega328 using on-line resources

1.1 Approach to the assignment

This handout (and the ones that follow) may be structured in a different way than a typical lab course. The intent is to provide you with resources that you can use to either solve a problem, or gain proficiency on a tool that you may use later to solve a problem. While there are some demonstration exercises and other deliverables listed below, if you approach this assignment with the goal of merely completing "what needs to be turned in", you will probably find that later in the semester, your proficiency on the material covered in this handout is poor, and you will end up wasting more time later than you saved by "just figuring out what I need to get this assignment done". Your goal for this assignment should be to gain proficiency. On the other hand, there is a lot of material, and it can quickly become overwhelming. As part of the course, you can develop some skills in finding and learning new material when you do not have an instructor to lay it all out for you. The ability to do this is one of the hallmarks of life-long learning.

We will discuss our approaches to learning new software in an upcoming discussion assignment. In the mean time, here is one way to approach it.

- Build up some fundamentals. Go through some introductory tutorials and write some simple programs.
- Work on a project - decide on a task you want to accomplish. Think about what do you need to know in order to accomplish it. Scan through documentation and look for key words and key examples.

- Build up expertise. If you get your project working, go back and research the functions or concepts you used in more detail. Try different implementations - is there a different way to accomplish the same thing? Learn about other methods, data structures, or advanced concepts.
- Pace yourself! Set a schedule for learning. It is hard to overstate how much better your retention will be if you work 1 hour a day for a week rather than 7 hours on one day.

2 The Arduino Software and Tutorials

To be able to do some basic projects, start with some simple programs (also called sketches) and upload them on the board. The Arduino Software (IDE) is used for this purpose. The Arduino language is based on C and C++. All C and C++ constructs work with the Arduino. This software has been installed on all machines in BB 305. It can be installed personal laptops if preferred. It runs on Windows, Mac OS X, and Linux. A step by step guide for Windows, Mac OS X and Linux can be found on Arduino's Website. After installation refer to the Arduino Development Environment guide to understand its purpose and what it helps you do.

The following links walk you through some basic tutorials to get you started.

1. Bare Minimum
2. Blink an LED (The "Hello World" of microcontrollers)
3. Digital Read Serial
4. Analog Read Serial
5. Button
6. Analog Input

3 Sensors and Interrupts

Interrupts are an efficient way of detecting such asynchronous events so the main code doesn't have to waste time keeping an eye out for these events and can keep doing what its doing. In the following, we will examine a position sensor called an encoder, and how to efficiently read an encoder using interrupts

3.1 Encoders

Encoders can be used to determine the amount of rotation of a shaft - for example the amount that the wheels of a robot turn. You will be using motors that have excremental encoders already attached. There are several places on the web to get an introduction to encoders, including the following

- `https://www.pc-control.co.uk/incremental_encoders.htm`
- `http://www.arduinoos.com/2010/06/rotary-encoders/`
- `https://en.wikipedia.org/wiki/Rotary_encoder#Incremental_rotary_encoder`

From these sites you should understand there are two channels that must be read to determine both the amount of rotation and the direction of rotation. If you are not clear on the operation, please pause here and ask a TA for some clarification.

3.2 Reading Encoders

Incremental encoders are also commonly used for manual input dials (for example, for a radio). In your Arduino kit there is a mechanical encoder of this type. The encoders have 5 leads. The two on one side are connected when the knob is pushed (i.e. radio on or off). The three on the other side are for the encoder. The middle lead is ground, while the other two leads are the A and B channels, which are alternately connected to ground as the knob is rotated. If your encoder has been attached to a small PCB Board, then these 5 leads will be accessed through a header, with the A

and B channels marked CLK and DT, and ground marked as GND. Grab an encoder, and connect the ground on the encoder to ground on your Arduino, and the A/CLK and B/DT pins to two other digital input pints. Some of the PCB boards have an internal pull-up resistor and some don't, so you will have to experiment using the boards one of two ways:

- When you program the Arduino, set the pin mode for digital inputs to have a pullup resistor. When this is done these pins will be high when the lead is not connected to ground, and low when they are connected to ground. Write an Arduino program that does nothing but read these two pins and print the result to the screen. Examine how the digital pin readings change as the knob is rotated.
- If this does not work, connect 5 V to the Vcc pin (in this case, pull-up resistors on the Arduino are not needed). Use the same program to check the pins on the Arduino

If you have any difficulty with this, use an oscilloscope to monitoring the A and B channels, triggered off of channel A.

One common issue with these mechanical switch type encoders is when the mechanical contacts bounce, causing quick oscillations between 1 and 0. You can mitigate this by adding some capacitors between the digital pins and ground (use R as the pullup resistor value, and choose 1/RC to have a time constant of between .1 and 1ms), or by adding some code in your interrupt routine that checks when the last interrupt was, and ignoring the interrupt if it happened too soon (say less than 50 ms or so).

Once you have verified that the channels are being read correctly, modify your program so that it counts up when the knob is turned clockwise and counts down when the knob is turned counterclockwise. Think about the transition table that you will need to keep track of. There are four possible states that the A and B channels can be in, so when a change happens, there are 12 possible transitions (some of which should not occur if the encoder is operating correctly). Here is a partially filled in table with the transitions. Complete the table, and then write the code to correctly keep track of the encoder rotation. (Note that if A and B are switched, clockwise and counter-clockwise are also switched. You will need to determine for your encoder which channel is "A" and which is "B")

Last State		Current State		Result
A	B	A	B	
0	0	0	1	count 1 clockwise
0	0	1	0	count 1 counter-clockwise
0	0	1	1	shouldn't happen
0	1	0	0	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	

3.3 Interrupts

One problem with the code you just wrote in the previous section is that the encoder pins are read once each time through the loop, which can be inefficient, and if there is a lot of other code in the loop you could miss transitions. A more efficient implementation would use interrupts. Note that on the Uno, interrupts can only be used with digital input lines 2 and 3. When the microcontroller gets interrupted, the code will stop what its doing and execute an Interrupt Service Routine (ISR) instead. Once the ISR completes execution, it can go back to doing what its doing. There are several types of interrupts. You can use interrupts that trigger based on an external pin. (Note which of your Arduino pins can be used.) You can also use interrupts that trigger based on an internal timer. For a further details, (and a full list of available interrupt events) check out Nick Gammon's notes. Implement the code from this blog page that reads a push button using interrupts, and use the rest of the blog to help you understand how each part of this program works.

While one implementation would have both encoder channels connected to interrupt pins, since we only have two pins, and we will eventually want to keep track of the rotation of two wheels on a robot, you will need implement a design were one of the channels is on an encoder pin, while the other is on a regular pin. This implementation works like this:

1. Suppose you have an interrupt handler monitoring changes in channel A. When called, the handler will read both channel A and channel B (recording the results in global variables). If A and B are the same state, you have rotated **two** counts in one direction, but if they are different, you have rotated **two** counts in the other direction. The count is also saved in a global variable. Use the table above to verify which direction is which.
2. You have a separate function that will supply the current position when called. When this function is called, it will read both channel A and channel B to get the current state. It will used the saved values in global variables as the last state. Using your table above, you can make a final adjustment to return the accurate count.

You will want to demonstrate the operation of both of these functions. In one case, you can have a print statement in the interrupt handler that gives the current count. You won't need anything in your main loop. This should print only as the encoder rotates, and it should update by twos. In the other case, implement a main loop that prints the current position every second, calling your function to obtain the current position. You should be able to increment the encoder by ones in this case.

4 Demonstration Exercises

The following are some simple exercises for you to demonstrate your abilities to program the Arduino. These exercises use elements from the tutorials listed above. Use serial.print functions to keep track of what is happening in your code and debugging.

1. Exercise 1: Hook up 4 LEDs to four digital pins. Use arrays to turn on each one in order until all of them have lit up and extinguish them in order until every one of them is off. Use two pushbuttons to act as "gas" and "brake". The "gas" button should speed up the blinking rate of the LED, and the "brake" button should slow it down.
2. Exercise 2: Demonstrate your encoder-reading program both with and without interrupts.

When you have finished an exercise, call over TA a to get the exercise graded. You should be able to obtain the correct results, and also go through the code and answer questions about it.

5 Documentation

1. Create a document that contains your completed encoder transition table, and answers the following questions. Upload this document to assignment link for the Intro to Arduino and Pi on Canvas,
   (a) How many digital I/O pins do you have access to on the Uno? How many of these pins can provide pulse width modulated (PWM) analog output? Provide an examples where you might use the PWM functionality of those pins.
   (b) How many analog I/O pins do you have access to on the Uno?
   (c) What is the recommended supply voltage for the Arduino?
   (d) How many external interrupts are available on the Uno, and on which pins?
   (e) What are the two functions that will always be a part of an Arduino sketch? Explain the purpose of each of these functions.
   (f) What is the purpose of the delay function in an Arduino sketch? What is the main drawback of using the delay function to control timing?
   (g) What are the alternatives to using the delay function?
   (h) What is a pull-up resistor, why is it used, and how can it be implemented internally on the Arduino?

2. Save your well commented and easily readable sketches for the demonstration exercises, and upload a copy of each to the assignment link on Canvas.

This documentation must be uploaded by the date given in the syllabus. In addition, don't for get to upload your reflection log to Canvas.