EE109 – Fall 2016: Introduction to Embedded Systems

Lab 9

Rotary Encoders

Introduction

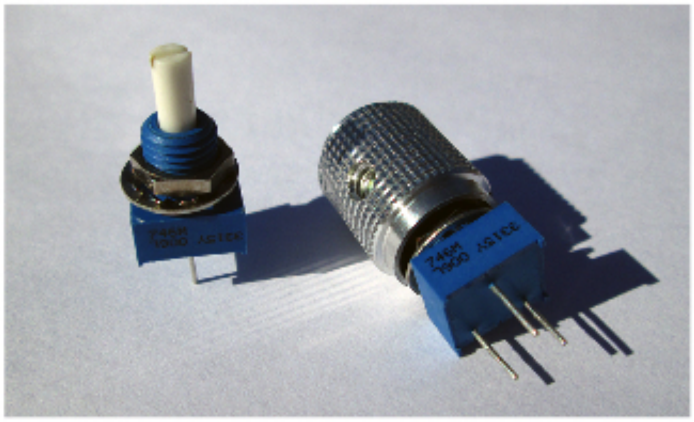
In this lab exercise you will learn how rotary encoders work. The outputs of the encoder will be connected to your Arduino and turning the encoder will cause the Arduino to change a numerical value up or down and then display the value on an LCD display. In the second part of the lab, the performance of the program willbe improved by using interrupts to monitor the inputs from the encoder.

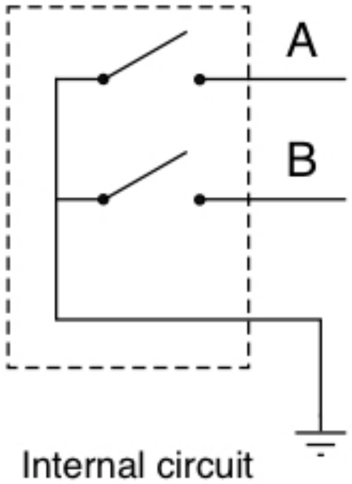
The Rotary Encoder (Background Covered in Class)

A rotary encoder is used to determine the angular position of something that rotates. For example, you are building a weather monitoring system and one of the devices providing input to the system is a weather vane that rotates to point in the direction the wind is blowing. The weather vane could be attached to a rotary encoder and the system would read the inputs from the encoder to see what angle the vane is pointing. Rotary encoders are also commonly used in devices that have a knob on them that the user needs to turn to adjust something.

Rotary encoders come in two types: absolute encoders and incremental encoders. An absolute encoder has outputs to provide a binary number for each angular position of the shaft ($0000 = 0^\circ$, $0001 = 22.5^\circ$, $0010 = 45^\circ$, etc.) The number of bits the encoder provides determines the resolution of the angle that can encoded. For example a 10-bit absolute encoder can resolve angles to $360/1024 = 0.35^\circ$.

Incremental encoders, also called quadrature encoders, are also available in a variety of resolutions (16, 64, 256, etc.~per revolution) but they only provides information about whether the shaft has been turned clockwise or counter-clockwise from the previous position, nothing about the actual position of the shaft. As the shaft of one of these encoders is turned, it opens and closes two switches to generate two binary signals.

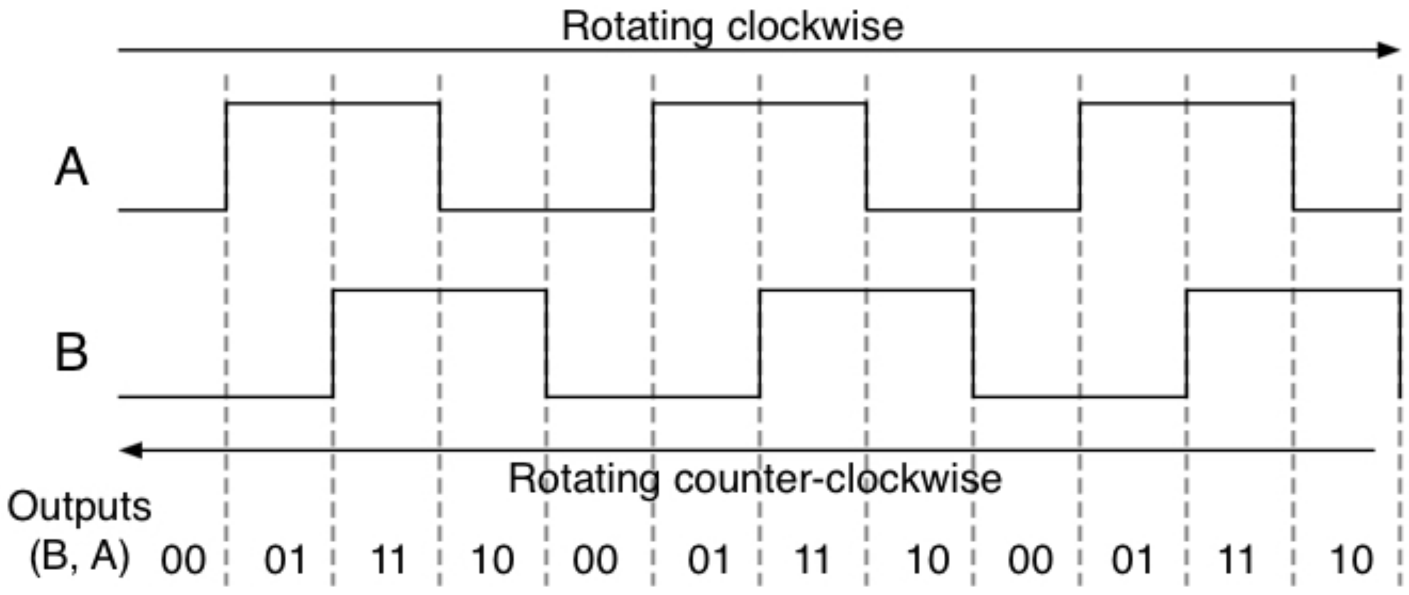

Incremental rotary encoders



Internal circuit of the encoder

The two switches can be used (when pull-up resistors are installed) to produce two signals that both look like a 50% duty cycle square wave (see below) but they are 90° out of phase with each other, which gives the appearance when drawn that one of the square waves is lagging behind the other by one quarter of a period.

To see how this can work, examine the diagram below and move along the waveforms from left to right as they would be generated when the control is being rotated clockwise. If we consider the two signals as the outputs of state machine where B is the MSB and A is the LSB, then the output code goes through the sequence 00, 01, 11, 10, 00, 01, etc. as the control is rotated.

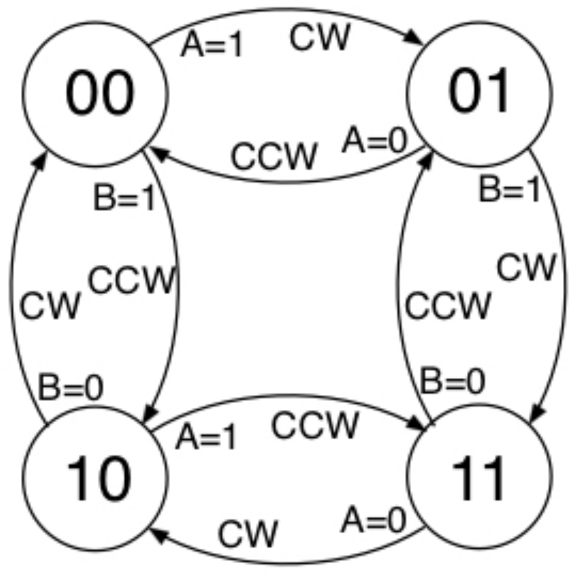


Output of the pins of an encoder while being rotated

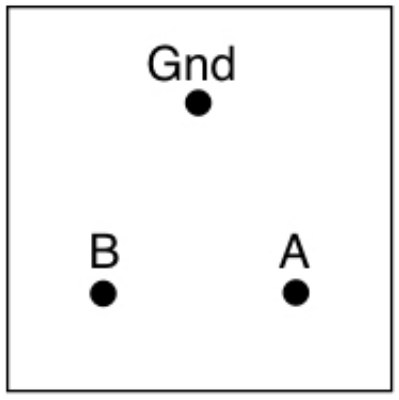
Now try moving along the waveforms from right to left (control being rotated counter-clockwise) and we can see that the output code goes through the sequence 00, 10, 11, 01, 00, etc. as the control is rotated. Note that the sequence is different depending on whether the control is being rotated clockwise or counter-clockwise. In fact every one of the transitions from one two-bit code to another in the clockwise direction is different from the ones that appear when rotating counter-clockwise. As a result whenever there is a code change the direction of rotation can be determined by knowing the old code and the new code.

The above sequences of numbers is called a Gray code and has the property that as you go from one number in the sequence to the next, in either direction, only one bit changes at every transition. Gray codes, usually with many more than two bits in them, are also used in absolute rotary encoders used to encode the position information in electromechanical devices such as scanners, printers, manufacturing equipment and many others. The property that only one bits changes at a time helps to eliminate errors in sensing the device's position. If a normal binary code was used there would be numerous places in the sequence where multiple bits would have to change simultaneously. With mechanical equipment, it's impossible to ensure that all the bits would change at exactly the same time and this can lead to errors as to the position or status of the device.

Using the two-bit Gray code number as a state variable we can construct the state diagram below for a state machine that describes the output of the encoder. Each of the four states corresponds to one of the four sets of output values the control can generate. From each state there are two transitions to an adjacent state, one for each of the code bits that might change. One transition is for clockwise rotation and the other is for counter-clockwise rotation. The direction of rotation is marked on the transition line.



State diagram for a rotary encoder



Encoder pins (bottom view)

The encoders provided for the lab exercise go through 64 states in the process of one revolution of the knob. The pins on the bottom are arranged with two pins (A and B) on one side, and the pin that goes to ground is on the opposite side. It's not too important which pin is A or B since swapping them only makes the state machine count in the opposite direction for a given direction of rotation.

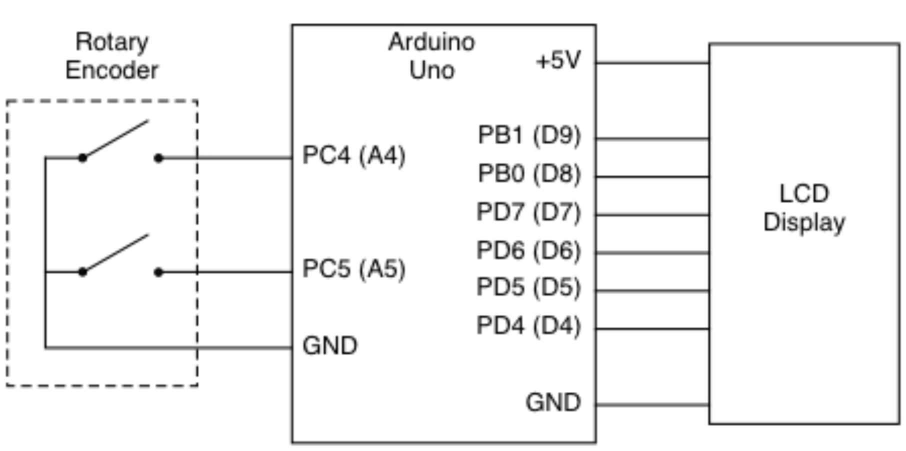
Task 1 - The Circuit

Install the rotary encoder on the breadboard being careful to make sure that all three pins are in different sets of the five hole connections strips. Make the connections between the encoder and the Arduino as shown in schematic diagram below.

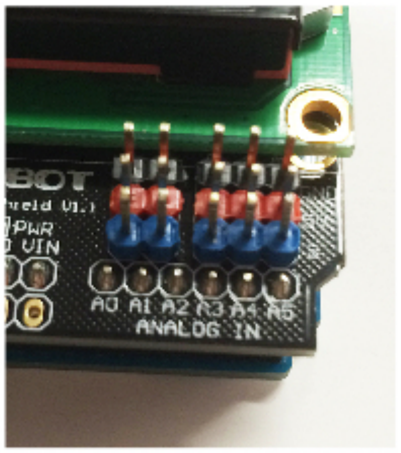
Two I/O port bits will be required for reading the encoder output into the Arduino. With the LCD attached to the Uno, several of the I/O ports bits are used by the LCD. Port D, bits 4-7, Port B, bits 0-2, and Port C, bit 0 are used by the LCD. In addition Port D, bits 0 and 1 should not be used since they are used by the USB interface for programming. To standardize things, everyone should use the A4 and A5 bits (Port C, bits 4 and 5).

Unfortunately once the LCD is mounted on the Arduino, you no longer have access to the connectors on the Arduino to make connections to the various I/O ports pins as you were able to do in previous labs. However the ones that are not being used by the LCD can still be accessed by using the pins on the top of the LCD board. In the bottom right corner of the LCD are three row of five pins with different colored bases. The top row (black base) are all ground connections. The second row (red base) are all $+5V$ connections. The bottom row (blue base) are connections to I/O ports A1 through A5 (PC1 through PC5). The pins for A4 and A5 are the ones just above the

A4 and A5 labels on the board.



Circuit diagram for Lab 9



Input pins on the LCD shield

From the instructor get a pack of male-female jumper wires. The female end can be pushed down onto a pin on the LCD shield and the male end can then be inserted into one of the breadboard holes. Use these jumpers to make the three connections needed between the breadboard and the Arduino as shown in the schematic.

Using the Encoder With Polling

In this lab exercise, the encoder will be used to control the incrementing and decrementing of a signed variable and the value of that variable will be displayed on the LCD shield. The variable starts with a value of zero and then increments or decrements by one for each state change of the encoder outputs. Turning the encoder in one direction increments the count, turning in the opposite direction decrements it. Since there are 64 encoder state changes for each revolution, the count should go up or down 64 each time the encoder is turned one revolution.

We will write the code for the first part of this lab in class together. This version of the program uses polling to watch the two input bits (Port C, bits 4 and 5) connected the encoder outputs. The basic structure of the program is as follows.

- Enable the pull-up resistors for the A4 and A5 input bits (Port C, bits 4 and 5).
- Allocate an `int` variable to store the count value that the encoder will increment and decrement. This needs to be a signed variable so we can handle positive and negative count values. Your program should start by initializing the count variable to zero and displaying this on the LCD.
- Read the two encoder bits to determine the **initial current state** of the encoder (and our initial position in the state machine.)
- Start an infinite loop that does the following:

1. Read the two input bits.
2. Based on the **current state** and the **input values**, find the new state and the new count value. This is all determined from the state diagram.
3. If the state has changed, update the count on the LCD and assign the new state value to the being the current state.

Task 2 - Confirm Encoder Operation

The first task is to use two channels of the oscilloscope to confirm that the encoder outputs are acting correctly as the knob is turned.

- From the class web site, download the file `lab9.c` and put it in a `lab9` folder.
- Add a copy of Makefile, lcd.c and lcd.h from the stopwatch lab to the lab9 folder.
- Modify the Makefile to work for the Lab 9 files.

The oscilloscope can be used to confirm that the encoder output bits are working properly. All that's required from your software for this test is for it to enable the pull up resistors on the encoder input ports, PC4 and PC5.

- Make sure your program contains the line to enable the PC4 and PC5 pull-ups.
  ```
  PORTC |= (1 << PC4) | (1 << PC5);
  ```
- Compile the program and download it into your Arduino..
- Turn on the scope and connect two scope probes to channels 1 and 2 of the scope.
- If it's not lit, press the "2" button below the channel 2 vertical adjustment knob so it lights up. Two traces, one yellow and one green, should show up on the screen.
- Set both channels for 2 volts/division and position one trace near the bottom of the screen and the other about in the middle.
- Set the horizontal scale to 10msec/division.
- Take three wires and plug them into the same three connection blocks that the encoder is plugged in to. These should be in three adjacent blocks with the center one being the ground connection and the two on the sides being for the encoder outputs. You may have to move the encoder and the jumpers to the Arduino around to make space for the wires.
- Connect the two scope probes to the wires coming from the connection blocks with the jumpers going to the A4 and A5 terminals on the Arduino. It doesn't make any difference which probe is connected to which wire. Connect the wire from the middle block to one of the ground clips from both probes. The other clip can be left unconnected.

Try rotating the encoder slowly and see if both traces on the scope go up and down as it's rotated. The scope is showing the states of two encoder output bits. If you rotate slowly you should be able to see it go through the 2-bit Gray code sequence as discussed in class: 00 → 01 → 11 → 10 →

00 `....` The main thing to look for is to confirm that both outputs are changing as the encoder is rotated and that the signal levels cover most of the range from ground to 5 volts.

Task 3 - Confirm Operation of the Encoder Program

Try out the operation of the `lab9.c` program with the encoder. You should be able to rotate the knob back and forth and see the numbers change on the display.

- Check that the numbers change smoothly as the encoder is rotated without skipping or repeating any values.
- Confirm that the numbers can go both positive and negative.
- These encoders go through 64 states per revolution. Confirm that rotating the knob one full revolution in either direction makes the count change by 64.

Task 4 - The Problem With Polling

Using polling to read the encoder inputs works reasonably well but it can fail if the inputs change too rapidly. To see this, try a simple experiment.

1. Turn the knob on your rotary encoder so the black line is visible and lined up with something like a corner of the encoder or some other feature.
2. Use the reset button on the LCD shield or the Uno to restart the program and zero the count value.
3. Spin the control knob through one full revolution as quickly as possible.
4. Note the count value on the LCD.

If the code was keeping up with the encoder outputs, the count value should have gone to 64 (or -64). However the count is probably less than that meaning that the program missed some of the encoder state transitions. This was most likely caused by the program being stuck in a delay routine in the LCD code when one or more transitions occurred. One way to prevent this from happening is to make the state changes trigger an interrupt. Regardless of what the program is doing at the time, the interrupt will cause execution to jump to the interrupt service routine to handle the state change. The program can service the interrupt while executing a delay routine and prevent encoder state transitions from being lost.

To implement this we will use the Pin Change Interrupt capability of the microcontroller. All of the I/O pins in the three ports (B, C and D) are capable of generating an interrupt if the pin does a $0 \rightarrow 1$ or $1 \rightarrow 0$ transition. While individual pins in a port can trigger an interrupt, there is only one interrupt vector (and associated Interrupt Service Routine) for each port.

- Changes on Port B pins invoke PCINT0 → `ISR(PCINT0_vect) { ... }`
- Changes on Port C pins invoke PCINT1 → `ISR(PCINT1_vect) { ... }`

- Changes on Port D pins invoke PCINT2 → `ISR(PCINT2_vect) { ... }`

Pin Change Interrupts for a port are enabled by setting the `PCIE0`, `PCIE1` or `PCIE2` bits in the `PCICR` register. In addition there is an 8-bit mask register (`PCMSK0`, `PCMSK1` and `PCMSK2`) associated with each port that determines which pins can generate an interrupt. If a bit in the mask register is set to a one, then a $0 \rightarrow 1$ or $1 \rightarrow 0$ transition on the corresponding bit in the I/O port will cause a Pin Change Interrupt. If a bit in the mask register is a zero, then changes on the corresponding I/O port bit will not cause an interrupt.

The Pin Change Interrupts are ideal for our application of watching for state changes on the two bits from the encoder. Once the Pin Change Interrupts are enabled on the two encoder bits, any change of state will cause the interrupt to occur. The ISR can include the code that determines what state change happened and whether to increment or decrement the count value.

Task 5 - Modify the Program to Use Interrupts

Before modifying your program to use interrupts, make a copy of your `lab9.c` just in case you have to go back to it for some reason. To change it to use interrupts, make the following modifications to the program.

- As with any program that uses interrupts add the following line somewhere near the start of the program.

  ```
  #include <avr/interrupt.h>
  ```
- To get the interrupts working
  - Set the `PCIE1` bit in the `PCICR` register to enable the pin change interrupts on Port C (PCINT1).
  - Set the bits in the `PCMSK1` mask register to enable interrupts for the PC4 and PC5 I/O lines. For PC4 and PC5, these are bits `PCINT12` and `PCINT13`.
  - Enable global interrupts.
- Add the " `ISR(PCINT1_vect)` " interrupt service routine and into this routine you should move the code from the main program loop that checks the state of the input bits and adjusts the count value accordingly. Don't move the code to display the count on the LCD to the ISR. We don't want the microcontroller tied up writing to the LCD while interrupts are disabled in the ISR.
- The main loop of the program should now just check to see if the count value has changed and if so, update the displayed value on the LCD.

Any variable that needs to be accessed from both the main program and then ISR must be declared as a global variable at the top of the program (not in the main routine). In addition these variables should be declared with the "volatile" keyword to tell the compiler that their contents can change outside the main stream of code execution.

After the the changes have been incorporated into the program, try running it and confirm that rotating the encoder still makes the count value go up and down. Once that is working, then try the experiment of rotating the control quickly and see if the ISR-based code does a better job of keeping up with the state changes. Ideally you should be able to spin the knob one revolution and see the count go up or down by 64 which shows that it is not losing any transitions.

Results

Once you have the assignments working demonstrate them to one of the instructors. Turn in a copy of the `lab9.c` source code through the link on the class web site.