Introduction

A rotary shaft encoder is an electromechanical device that can be used to determine angular position of a shaft. Encoders have numerous applications, since angular position can be used to determine position or velocity of a mechanical device connected to a shaft making them ideal for speed control or for precise motor movement. A common application is in motor feedback and motors are commonly sold with encoders attached to the end.

There are two general types of shaft encoders, absolute and relative. Absolute encoders offer the exact angular position of a shaft at any given time, whereas relative encoders can only measure change in angular position relative to an arbitrary datum (such as the moment a switch is activated). In the case of relative encoders, more sensors are required if exact position is desired. This paper documents an absolute shaft encoder, primarily due to ease of integration and cost.

Construction of both sensors is different; however, the general principle is the same. Optical sensors detect patterns on a rotating disk, which are directly attached to a rotating shaft (which can then be connected to the device of interest, such as a motor's shaft). Absolute encoders commonly use optical sensors which read graycodes on the disk (Figure 1). Depending on accuracy, this could be a simple black dot which the sensor detects on each rotation. For slightly more precision, a basic code can be printed on a paper disk using a postscript utility, then glued on to a solid disk (see References).

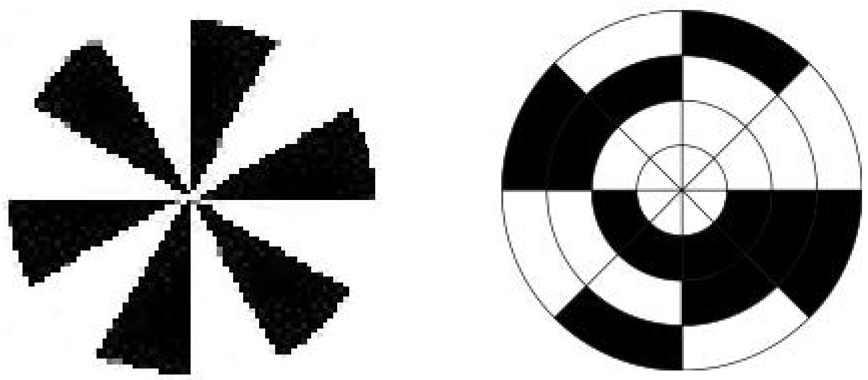

Figure 1: Sample Encoded Disks

This code must then be decoded in hardware or software to determine the corresponding angular position. Relative encoders also use optical detectors and disks; the disks have radial slots which the optical detectors recognizes each time it passes through its field of view. More sensors can be attached to increase precision, and if direction (forward/reverse) is desired, then the design requires two sensors at a phase offset (such encoders are called quadrature encoders).

Although the absolute shaft encoder documented in this paper does not utilize the above techniques, the background is useful if the reader is designing their own encoder (there is an excellent link in the References section to a site describing exactly how to do design your own encoder). The encoder described here, the MA2 miniature absolute magnetic shaft encoder, uses magnetic techniques to provide 0.35° resolution at 1024 positions per revolution. There are two output options, analog and PWM. The analog option outputs an analog voltage between 0V and the input voltage (maximum of 5V) that is proportional to the absolute shaft position. The PWM option outputs a pulse width modulated digital signal nearly 1000 times per second with a duty cycle that is proportional to the absolute shaft position. Both encoders offer the same functionality, so choosing which to purchase is primarily a function of preference and the capabilities of the microcontroller. When ordering the encoder, there are a number of other options including ball bearings and shaft diameter. It is highly recommended the user purchases the ball bearing option and also buys the connector offered by US Digital, as the non-ball bearing option has significant resistance in rotation and the encoder requires the manufacturer’s custom connector. For details, see the US Digital purchase site (References).

Analog Absolute Shaft Encoder

A diagram of the absolute encoder is shown in Figure 2. There are inputs for ground and Vdd (maximum of 5V). The encoder has one output which consists of a voltage, between 0 and Vdd, which indicates the current shaft angular position relative to an arbitrary reference position. There is a jump discontinuity in the voltage output as the shaft completes a revolution and the output drops from Vdd to 0V. Because of this, when the encoder shaft is spun at a constant speed, the output should take the form of a sawtooth wave.

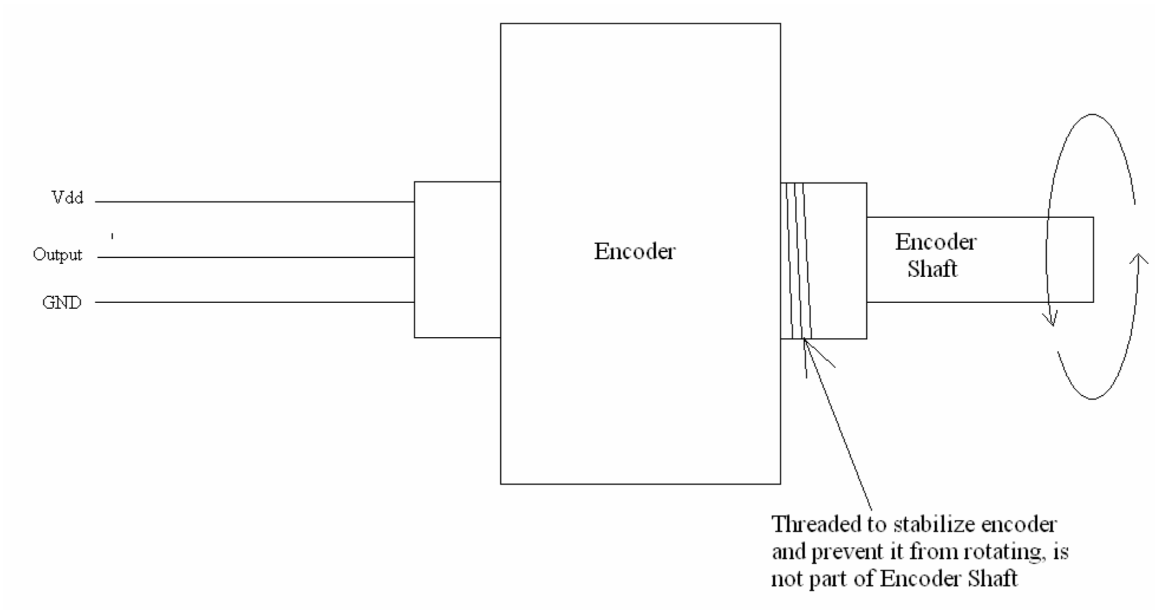
Vdd
Output
GND
Encoder
Encoder Shaft
Threaded to stabilize encoder and prevent it from rotating, is not part of Encoder Shaft

Figure 2: A schematic of the shaft encoder.

Coupling

In order to use the US Digital magnetic shaft encoder, the encoder shaft must first be coupled to the DC motor shaft. Several of the motors in the Microprocessors's lab have a small portion of the motor shaft jutting out of the back of the motor; coupling the encoder to the motor shaft here is ideal since it will not get in the way of normal shaft operation. In order to physically couple the two shafts together, US Digital markets a number of couplers to attach the encoder to various shaft sizes. According to their applications engineer, however, rubber tubing should work just as well. The rubber washer-like inserts used in standard screwdriver packaging can also be successfully used for coupling. For ease of coupling, be sure to purchase the ball bearing option!

Velocity Calculations

Before the shaft encoder can be used in a digital system, the change in A/D conversion values between sampling periods must be correlated to an actual angular velocity of the encoder shaft. Since the Harrisboard 2.0 provides a regulated 3.3V power supply for peripherals, 3.3V is used as Vdd for the purposes of these calculations.

The encoder works by varying an analog output from GND at the zero degrees shaft position to V_{dd} at roughly one full rotation. Thus, the voltage output of the encoder over any single turn can be represented by the equation:

$$bA = V \tag{1}$$

where V is the encoder output voltage, A is the shaft angle (in degrees) and b is a scaling factor. Given $V = 3.3\text{V}$ at $A = 360°$, $b = 0.009167$.

Given a desired shaft speed of X rpm, to find the change in A/D values between sampling, first convert to revolutions per second by dividing by 60:

$$Z = \frac{X}{60}\,\text{rps} \tag{2}$$

where Z is the speed in revolutions per second and X is the shaft speed as defined above. Next, use the sampling encoder sampling rate S to determine the desired change in angle ΔA between measurements:

$$\Delta A = \frac{Z}{S} * 360 \tag{3}$$

This can be converted into a desired encoder ΔV by plugging ΔA into equation (1).

Once ΔV has been found, the desired change in A/D readout between samples can be determined. Since the PIC is based on an 8-bit architecture, it is common for only the upper 8 bits of the 10 bit A/D value to be used. Assuming an 8 bit A/D value, the input voltage range of 0 to 3.3V is essentially divided into 256 discrete voltage steps of $\frac{3.3V}{256} = 0.012890624\frac{V}{ADval}$. Thus, the desired difference between any two successive values is:

$$\Delta AD = 0.012890625 \Delta V \tag{4}$$

Since the encoder essentially produces a sawtooth waveform, whenever the shaft completes a revolution, equation (4) no longer holds true and ΔAD becomes a relatively large negative number. If the shaft is guaranteed to always be traveling in the forward direction, then if ΔAD is negative, the shaft must have completed a revolution. In order to get the change in position out of this, merely subtract the ΔAD value from 0x100 which should account for the shaft "overflow." If the shaft is expected to travel both forwards and backwards, "overflow" can essentially be estimated by comparing the last two ΔAD values. If the latest ΔAD is negative and the previous value is positive, assuming that the motor changes direction relatively infrequently compared to the number of revolutions made by the shaft, it is safe to assume that the shaft has "overflowed" and 0x100 should be added to the ΔAD value.

Cost

Part Number	Details	Price
MA-2-A	Analog Absolute Shaft Encoder	$29.00
B8	(1/8" ball bearing option)	$6.00
CA-7941-1FT	Connector with cables attached	$5.00

References

Application of Absolute Shaft Encoder in PID Motor Controller:
<Prof Harris, please include hyperlink to our PID motor contoller Microp's Project>

MA2 Absolute Shaft Encoder Purchase Site:
http://www.usdigital.com/products/ma2/

Useful sight on designing your own encoder:
http://www.geology.smu.edu/~dpa-www/robo/Encoder/pitt_html/encoders.html

Appendix A: Final C code

```
/* Encoder_Reader.c:  Sample code for reading and
 *                    calculating velocity from an analog
 *                    output absolute shaft encoder.
 * Authors:  Andrew Danowitz, Amir Adibi
 * Date:  December 2006
*/

#include <p18f452.h>
#include <timers.h>

short ADLast[2] = {0,0}; //previous AD values
short ADPres;            //latest AD value


/* Function Prototypes */
void main(void);
void isr(void);

#pragma code low_vector = 0x18
void low_interrupt(void)
{
     _asm
          GOTO isr
     _endasm
}

#pragma code
void main(void)
{
     TRISA = 0xFF;  //Set A/D converter port for input

     //Set up timer 1 to interrupt roughly once every 10ms
     INTCON = 0xF0; //enable interrups
     TMR0H = 0x00;  //set timer initial values
     TMR0L = 0x3D;
     T0CON = 0xC7;  //start the timer

     //wait for timer interrupt
     while(1)
     {

     }

}
```

```
#pragma interruptlow isr
void isr(void)
{
     //reset the timer
     INTCON = 0xF0; //enable interrups
     TMR0H = 0x00;  //set timer initial values
     TMR0L = 0x3D;
     T0CON = 0xC7;  //start the timer

     //Configure the A/D converter to read from encoder
     ADCON1 = 0x00;
     ADCON0 = 0x8D;

     //Wait for valid A/D data
     while(ADCON0bits.GO==1)
     {
     }

     //Read A/D value
     ADPres = ADRESH;

     //if the voltage has only decreased this last round,
     //then we assume that the shaft has just completed a
     //rotation and is still moving forward.
     if (ADPres>=0x00F0 && ADLast[1]<=0x0010)
     {
          //If the shaft did overflow, then the difference
          //between the new and old angle values is 0x100
          //plus the new eight bit angle value minus the
          //old angle value
          dAD = 0x0100+ADLast[1] - ADPres;
     }

     //If the shaft hasn't completed a revolution, the
     //change in angle is just new minus old
     else
     {
          dAD = ADLast[1]-ADPres;
     }

     //Push the latest AD value into the stack
     ADLast[0] = ADLast[1];
     ADLast[1] = ADPres;
}
```