Arduino: Playground

search

The playground is a publicly-editable wiki about Arduino.

Manuals and Curriculum

Hardware and Related Initiatives

Board Setup and Configuration

Development Tools

Interfacing With Hardware

- Output
- Input
- Storage
- Communication

Interfacing with Software

Code Library and Tutorials

Suggestions & Bugs

Electronics Technique

Sources for Electronic Parts

Arduino People/Groups & Sites

Exhibition

Languages

PARTICIPATE

- create an account
- suggestions
- formatting suggestions
- all recent changes
- PmWiki
- WikiSandBox training
- Basic Editing
- Cookbook (addons)
- Documentation Index)

View Edit History Print Login Logout

Reading Rotary Encoders

here's my first contribution to this wiki. I hope this is the right place for it.

A rotary or "shaft" encoder is an angular measuring device. It is used to precisely measure rotation of motors or to create wheel controllers (knobs) that can turn infinitely (with no end stop like a potentiometer has). Some of them are also equipped with a pushbutton when you press on the axis (like the ones used for navigation on many music controllers). They come in all kinds of resolutions, from maybe 16 to at least 1024 steps per revolution, and cost from 2 to maybe 200 EUR.

I've written a little sketch to read a rotary controller and send it's readout via RS232.

It simply updates a counter (encoder0Pos) every time the encoder turns by one step, and sends it via serial to the PC.

This works fine with an ALPS STEC12E08 encoder which has 24 steps per turn. I can imagine it could fail with encoders with higher resolution, or when it rotates very quickly (think: motors), or when you extend it to accomodate multiple encoders. Please give it a try.

I learned about how to read the encoder from the file encoder.h included in the arduino distribution as part of the AVRLib. Thanks to its author, Pascal Stang, for the friendly and newbie-proof explanation of the functionings of encoders there. here you go:

```
/* Read Quadrature Encoder
 * Connect Encoder to Pins encoder0PinA, encoder0PinB, and +5V.
 *
 * Sketch by max wolf / www.meso.net
 * v. 0.1 - very basic functions - mw 20061220
 *
 */


int val;
int encoder0PinA = 3;
int encoder0PinB = 4;
int encoder0Pos = 0;
int encoder0PinALast = LOW;
int n = LOW;
```

edit SideBar

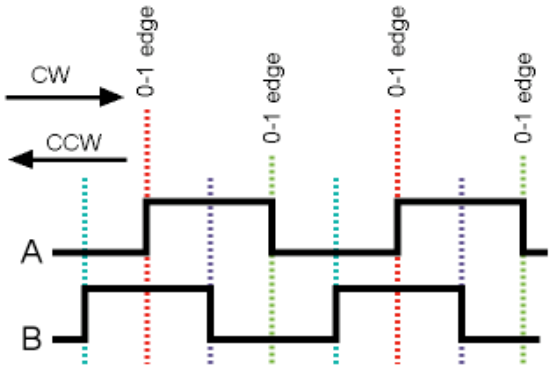
```
void setup() {
  pinMode (encoder0PinA,INPUT);
  pinMode (encoder0PinB,INPUT);
  Serial.begin (9600);
}

void loop() {
  n = digitalRead(encoder0PinA);
  if ((encoder0PinALast == LOW) && (n == HIGH)) {
    if (digitalRead(encoder0PinB) == LOW) {
      encoder0Pos--;
    } else {
      encoder0Pos++;
    }
    Serial.print (encoder0Pos);
    Serial.print ("/");
  }
  encoder0PinALast = n;
}
```

Oh, a few notes:

- encoder0Pos will be counting forever, that means that if you keep turning into the same direction, the serial message will become longer (up to 6 characters), costing more time to transmit.
- you need to make sure yourself (on the PC side) that nothing bad happens when encoder0Pos overflows - if the value becomes larger than the maximum size of an INT (32,767), it will flip to -32,768! and vice versa.
- suggestion for improvement: make it spit out the counter only when it is polled from the PC. Count only the relative change of the encoder between two polls.
- obviously, if you add more code to the loop(), or use higher resolution encoders, there is a possibility that this sketch will not see every individual step. The better way of counting encoder steps is to use an interrupt on every flank of the signal. The library I mentioned above does just that, but currently (2006-12) it doesn't compile under the arduino environment - or I just don't know how to do so...

I'm not sure about the etiquette of this, but I'm just going to add onto this tutorial. Paul Badger

Below is an image showing the waveforms of the A & B channels of an encoder.



This might make it a little more clear how the code above works. When the code finds a low-to-high transition on the A channel, it checks to see if the B channel is high or low and then increments/decrements the variable to account for the direction that the encoder must be turning in order to generate the waveform found.

One disadvantage of the code above is that it is really only counting one fourth of the possible transitions. In the case of the illustration, either the red or the lime green transitions, depending on which way the encoder is moving.

Below is some code that uses an interrupt. When the Arduino sees a **change** on the A channel, it immediately skips to the "doEncoder" function, which parses out both the low-to-high and the high-to-low edges, consequently counting twice as many transitions. I didn't want to use both interrupt pins to check the other two classes of transition on the B channel (the violet and cyan lines in the chart above), but it doesn't seem much more complicated to do so.

Using interrupts to read a rotary encoder is a perfect job for interrupts because the interrupt service routine (a function) can be short and quick, because it doesn't need to do much.

I used the encoder as a "mode selector" on a synthesizer made solely from an Arduino chip. This is a pretty casual application, because it doesn't really matter if the encoder missed pulses, the feedback was coming from the user. Where the interrupt method is going to shine is with encoders used for feedback on motors - such as servos or robot wheels. In those applications, the microcontroller can't afford to miss any pulses or the resolution of movement is going to suffer.

One side note: I used the Arduino's pullup resistors to "steer" the inputs high when they were not engaged by the encoder. Hence the encoder common pin is connected to ground. The sketch above fails to mention that some pulldown resistors (10k is fine) are going to be needed on the inputs since the encoder common is attached to +5V.

```
/* read a rotary encoder with interrupts
   Encoder hooked up with common to GROUND,
   encoder0PinA to pin 2, encoder0PinB to pin 4 (or pin 3 see below)
   it doesn't matter which encoder pin you use for A or B

   uses Arduino pullups on A & B channel outputs
   turning on the pullups saves having to hook up resistors
   to the A & B channel outputs

*/

#define encoder0PinA  2
#define encoder0PinB  4

volatile unsigned int encoder0Pos = 0;

void setup() {


  pinMode(encoder0PinA, INPUT);
  digitalWrite(encoder0PinA, HIGH);       // turn on pullup resistor
  pinMode(encoder0PinB, INPUT);
  digitalWrite(encoder0PinB, HIGH);       // turn on pullup resistor

  attachInterrupt(0, doEncoder, CHANGE);  // encoder pin on interrupt 0 - pin 2
  Serial.begin (9600);
  Serial.println("start");                // a personal quirk

}

void loop(){
// do some stuff here - the joy of interrupts is that they take care of themselves
}
```

```
void doEncoder(){
  if (digitalRead(encoder0PinA) == HIGH) {   // found a low-to-high on channel A
    if (digitalRead(encoder0PinB) == LOW) {  // check channel B to see which way
                                             // encoder is turning
      encoder0Pos = encoder0Pos - 1;         // CCW
    }
    else {
      encoder0Pos = encoder0Pos + 1;         // CW
    }
  }
  else                                        // found a high-to-low on channel A
  {
    if (digitalRead(encoder0PinB) == LOW) {   // check channel B to see which way
                                              // encoder is turning
      encoder0Pos = encoder0Pos + 1;          // CW
    }
    else {
      encoder0Pos = encoder0Pos - 1;          // CCW
    }

  }
  Serial.println (encoder0Pos, DEC);          // debug - remember to comment out
                                              // before final program run
  // you don't want serial slowing down your program if not needed
}

/*  to read the other two transitions - just use another attachInterrupt()
in the setup and duplicate the doEncoder function into say,
doEncoderA and doEncoderB.
You also need to move the other encoder wire over to pin 3 (interrupt 1).
*/
```