Reading Rotary Encoders

ALPS STEC12E07 Encoder

Contents

- Introduction
  - Example 1: Read :Quadrature Encoder (checks the Encoder in loop()
  - Further Description and Encoder Waveform
  - Incremental Rotary Encoders, explained, illustrated and exemplified. Both operating modes are investigated: /w w/o interrupts.
- Libraries
  - *New* This code gives library performance without the compiled size overhead, with very efficient interrupt routines. Uses the External Interrupt pins of the MCU.
  - *New* A high performance Encoder library is now available, with 4X counting and very efficient interrupt routines. Uses the External Interrupt pins of the MCU.
  - *New* ClickEncoder Library for encoders with button Works on any IO-Pin (uses a Timer), supports acceleration (rotate faster -> increment faster, feels very natural); Button reports Click, DoubleClick, Held, Released. Example Included.
  - *New* Interrupt Library (the Encoder interrupts the processor). Utilizes any of the ATmega328P pins via the PinChangeInt library.
  - SoftTimer library introduces an alternative way to organize your business logic, and has a built in driver for rotary encoders.
  - AdaEncoder rotary encoder library (uses ... the PinChangeInt library). (AdaEncoder wiki documentation includes a short discussion of switch bounce). (AdaEncoder Arduino Forum discussion).
  - "Quadrature Encoder too Fast for Arduino" is an extremely non-portable library for handling high-speed, high-resolution encoders on the Arduino.
  - "Quadrature Encoders in Arduino, done right. Done right." uses "interrupt on any change of A or B" to squeeze the maximum possible resolution out of an encoder -- count on both low-to-high and high-to-low transitions, both A and B.
  - Encoder Polling (Arduino Forum discussion).
  - "Arduino Library for Rotary Encoders with Velocity Sense".
  - "RotaryEncoder"
  - Rotary encoder library for use in loop(). No example sketch given.
- Debounce Circuitry
- Examples

- Uses 1 Interrupt pin but misses half the state transitions
- Uses both External Interrupt pins
- Uses 1 External Interrupt pin (misses half the state transitions), wraps the Encoder into a C++ class.
- Uses both External Interrupt pins, Counts in 1 Direction only.
    - Uses both External Interrupt pins, simplifies the interrupt service routines of the above.
- Fast. Uses both External Interrupt pins, after the initial read, does not read the state of the pins.
- Uses both External Interrupt pins. Based on the Circuits@home code.
- Uses 1 Interrupt pin but misses half the state transitions.
- Uses both External Interrupt pins. "XOR method"
- Another example using the above "XOR method"
- loop() Example, and the Encoder interrupts the processor.
- Int0 & Int1 example using bitRead() with debounce handling and true Rotary Encoder pulse tracking
- Multiple encoders, software debounce, all state transitions, without interrupts

- Discussion
    - On Interrupts
    - On Speed

Introduction

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 its 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 accommodate 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:

Example 1

```
/* 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 ;

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;
}
```

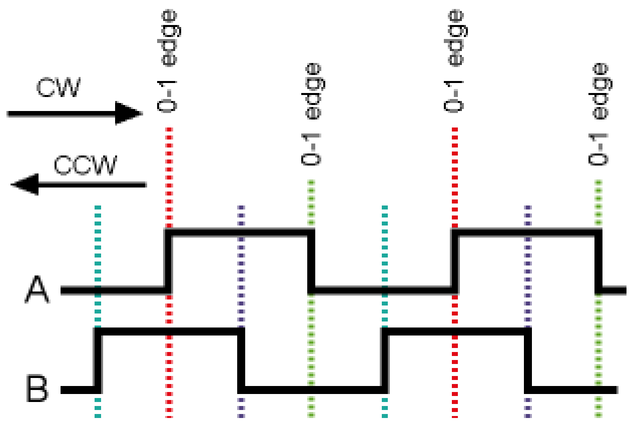
Oh, a few notes:

- 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...

Further Description, Including Encoder Waveform

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.

Interrupt Example

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 pull-up 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 pull-down 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 pull-ups on A & B channel outputs
   turning on the pull-ups 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 pull-up resistor
  pinMode(encoder0PinB, INPUT);
  digitalWrite(encoder0PinB, HIGH);       // turn on pull-up 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 pinA and pinB are both high or both low, it is spinning
   * forward. If they're different, it's going backward.
   *
   * For more information on speeding up this process, see
   * [Reference/PortManipulation], specifically the PIND register.
   */
  if (digitalRead(encoder0PinA) == digitalRead(encoder0PinB)) {
    encoder0Pos++;
  } else {
    encoder0Pos--;
  }

  Serial.println (encoder0Pos, DEC);
}

/* See this expanded function to get a better understanding of the
 * meanings of the four possible (pinA, pinB) value pairs:
 */
void doEncoder_Expanded(){
  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).
*/
```

- BY:dskv ***CAUTION!!!***

Careful when using Serial.Print inside an interrupt function, most of the time it will fail, but it works sometimes, the worst of programming bugs. It is documented in a number of places: "https://groups.google.com/a/arduino.cc/forum/#!topic/developers/HKzEcN6gikM" "http://forum.arduino.cc/index.php?topic=94459.0" "http://forum.jeelabs.net/node/1188.html"

Interrupt Example (the Encoder interrupts the processor). Uses both Interrupt pins

Code for reading encoder using 2 interrupts on pin 2 & pin3

Note: the code below uses 2 interrupts to read the full resolution of the encoder. The code above used 1 interrupt. It read half the resolution by only checking EncoderPin A for position, but it freed up an interrupt pin.

```
#define encoder0PinA 2

#define encoder0PinB 3

volatile unsigned int encoder0Pos = 0;

void setup() {

  pinMode(encoder0PinA, INPUT);
  pinMode(encoder0PinB, INPUT);

// encoder pin on interrupt 0 (pin 2)

  attachInterrupt(0, doEncoderA, CHANGE);

// encoder pin on interrupt 1 (pin 3)

  attachInterrupt(1, doEncoderB, CHANGE);

  Serial.begin (9600);

}

void loop(){ //Do stuff here }

void doEncoderA(){
  // look for a low-to-high on channel A
  if (digitalRead(encoder0PinA) == HIGH) {

    // check channel B to see which way encoder is turning
    if (digitalRead(encoder0PinB) == LOW) {
      encoder0Pos = encoder0Pos + 1;         // CW
    }
    else {
      encoder0Pos = encoder0Pos - 1;         // CCW
    }
  }

  else   // must be a high-to-low edge on channel A
  {
    // check channel B to see which way encoder is turning
    if (digitalRead(encoder0PinB) == HIGH) {
      encoder0Pos = encoder0Pos + 1;          // CW
    }
    else {
      encoder0Pos = encoder0Pos - 1;          // CCW
    }
  }
  Serial.println (encoder0Pos, DEC);
  // use for debugging - remember to comment out

}

void doEncoderB(){

  // look for a low-to-high on channel B
  if (digitalRead(encoder0PinB) == HIGH) {

   // check channel A to see which way encoder is turning
    if (digitalRead(encoder0PinA) == HIGH) {
      encoder0Pos = encoder0Pos + 1;         // CW
    }
    else {
      encoder0Pos = encoder0Pos - 1;         // CCW
    }
  }

  // Look for a high-to-low on channel B

  else {
    // check channel B to see which way encoder is turning
    if (digitalRead(encoder0PinA) == LOW) {
      encoder0Pos = encoder0Pos + 1;          // CW
    }
    else {
      encoder0Pos = encoder0Pos - 1;          // CCW
    }
  }

}
```

Interrupt Example (the Encoder interrupts the processor). Uses a single interrupt pin, wraps the Encoder into a C++ class.

class wrapper

by mikkoh [01/2010]

wrapped above example (example with one interrupt) into a class and reduced code size from the doEncoder function a bit (hope it's still readable to the most of you). An example usage is contained in the class documentation.

```
#ifndef __ENCODER_H__
#define __ENCODER_H__

#include "WProgram.h"

class Encoder {
 /*
   wraps encoder setup and update functions in a class

   !!! NOTE : user must call the encoders update method from an
   interrupt function himself! i.e. user must attach an interrupt to the
   encoder pin A and call the encoder update method from within the
   interrupt

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

   // ------------------------------------------------------------------------------------------------
   // Example usage :
   // ------------------------------------------------------------------------------------------------
       #include "Encoder .h"

       Encoder encoder(2, 4);

       void setup() {
           attachInterrupt(0, doEncoder  , CHANGE);
           Serial.begin (1 15200);
           Serial.println("start");
       }

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

       void doEncoder(){
           encoder .update();
           Serial.println( encoder  .getPosition() );
       }
   // ------------------------------------------------------------------------------------------------
   // Example usage end
   // ------------------------------------------------------------------------------------------------
 */
public:

   // constructor : sets pins as inputs and turns on pullup resistors

   Encoder( int8_t PinA, int8_t PinB) : pin_a ( PinA), pin_b( PinB ) {
       // set pin a and b to be input
       pinMode(pin_a, INPUT);
       pinMode(pin_b, INPUT);
       // and turn on pull-up resistors
       digitalW rite(pin_a, HIGH);
       digitalW rite(pin_b, HIGH);
   };

   // call this from your interrupt function
```

```
    void update () {
        if (digitalRead(pin_a)) digitalRead(pin_b) ? position++ : position--;
        else digitalRead(pin_b) ? position-- : position++;
    };

    // returns current position

    long int getPosition () { return position; };

    // set the position value

    void setPosition ( const long int p) { position = p; };

private:

    long int position;

    int8_t pin_a;

    int8_t pin_b;
};

#endif // __ENCODER_H__
```

Interrupt Example (the Encoder interrupts the processor).

Uses both External Interrupt pins, Counts in 1 Direction only.
Editor's note: Although this code claims efficiency gains, note that it uses the digitalRead() library functions, which according to http://jeelabs.org/2010/01/06/pin-io-performance/ is 50 times as slow as direct port reads.

Efficiency gain for rotary encoder counting

by m3tr0g33k

Paul Badger's work and the original post are enlightening and useful - you will need to understand what they have said before this makes sense to you (I hope it does!).

My project is a data logger where three analogue inputs are sampled each time a rotary encoder pulse steps clockwise. On an Arduino, time is of the essence to get this data sampled and saved somewhere (I have not included the 'saving somewhere' part of this project yet.) In order to save some processor cycles, I have slightly redesigned the interrupt system to maintain a pair of Boolean states outside of the interrupt loop.

The idea is to set a Boolean state for A or B when there is a positive going edge on encoder output A or B. So, when you get an interrupt from A, and it's positive going, you set A_set=true. Then you test if B_set is false. If it is, then A leads B which means a clockwise step (increment position counter).

Likewise, when you get an interrupt from B, and it's positive going, you set B_set=true. Then you test if A_set is false. If it is, then B leads A, which means a counter-clockwise step (decrement position counter).

An important difference from previous code examples is when there is an interrupt on A or B which is negative going, you just set A_set or B_set to false, respectively, no further work is needed, reducing the time spent servicing the interrupt.

Anyway, enough explanation of code in words, here is the code:

```
#define encoder0PinA 2
#define encoder0PinB 3

volatile unsigned int encoder0Pos = 0;
unsigned int tmp_Pos = 1;
unsigned int valx;
unsigned int valy;
unsigned int valz;

boolean A_set;
boolean B_set;


void setup() {

  pinMode(encoder0PinA, INPUT);
  pinMode(encoder0PinB, INPUT);

// encoder pin on interrupt 0 (pin 2)
  attachInterrupt(0, doEncoderA, CHANGE);

// encoder pin on interrupt 1 (pin 3)
  attachInterrupt(1, doEncoderB, CHANGE);

  Serial.begin (9600);
}


void loop(){
//Check each second for change in position
  if (tmp_Pos != encoder0Pos) {
    Serial.print("Index:"); Serial.print(encoder0Pos, DEC); Serial.print(", V     alues: ");
    Serial.print(valx, DEC); Serial.print(", ");
    Serial.print(valy , DEC); Serial.print(", ");
    Serial.print(valz, DEC); Serial.println();
    tmp_Pos = encoder0Pos;
  }
  delay(1000);
}


// Interrupt on A changing state
void doEncoderA(){

  // Low to High transition?
  if (digitalRead(encoder0PinA) == HIGH) {
    A_set = true;
    if (!B_set) {
      encoder0Pos = encoder0Pos + 1;
      valx=analogRead(0);
      valy=analogRead(1);
      valz=analogRead(2);
    }
  }

  // High-to-low transition?
  if (digitalRead(encoder0PinA) == LOW) {
    A_set = false;
  }

}
```

```
// Interrupt on B changing state
void doEncoderB(){

  // Low-to-high transition?
  if (digitalRead(encoder0PinB) == HIGH) {
    B_set = true;
    if (!A_set) {
      encoder0Pos = encoder0Pos - 1;
    }
  }

  // High-to-low transition?
  if (digitalRead(encoder0PinB) == LOW) {
    B_set = false;
  }
}
```

The rest of the code around the improved interrupt routines is just to demonstrate that it works. As I said, I only want to sample when going CW (which is forwards on my sampling trolley). When the encoder is going CCW, I just update the counter.

The loop{} prints the current encoder position and the corresponding sampled data values every second, but only if the encoder position has changed. You can have fun seeing how far you can rotate your encoder in a second! I have managed nearly 300 steps or 1.5 revolutions on my rotary encoder.

There is one issue with the logic in this code. If you are changing direction a lot then you may wish to know that if you change direction in the middle of a step, your counter will not update. This is a half-step hysteresis. Under most circumstances, this is not noticeable or important, but think whether it is important to you!

Hope this potential speed increase helps someone!

m3tr0g33k

Interrupt Example (the Encoder interrupts the processor). Uses both External Interrupt pins, simplifies the interrupt service routines of the above.

Tighten up the ISRs

You can reduce the size of the interrupt service routines considerably by looking at A_set == B_set to determine lag vs lead.

The ISRs are then just

```
// Interrupt on A changing state
void doEncoderA(){
  // Test transition
  A_set = digitalRead(encoderPinA) == HIGH;
  // and adjust counter + if A leads B
  encoderPos += (A_set != B_set) ? +1 : -1;
}

// Interrupt on B changing state
void doEncoderB(){
  // Test transition
  B_set = digitalRead(encoderPinB) == HIGH;
  // and adjust counter + if B follows A
  encoderPos += (A_set == B_set) ? +1 : -1;
}
```

Basically, if the pin that changed now matches the other pin, it's lagging behind it, and if the pin that changed is now different, then it's leading it.

Net result: two lines of code to process the interrupt.

The entire sketch is

```
enum PinAssignments {
  encoderPinA = 2,
  encoderPinB = 3,
  clearButton = 8
};

volatile unsigned int encoderPos = 0;
unsigned int lastReportedPos = 1;

boolean A_set = false;
boolean B_set = false;


void setup() {

  pinMode(encoderPinA, INPUT);
  pinMode(encoderPinB, INPUT);
  pinMode(clearButton, INPUT);
  digitalWrite(encoderPinA, HIGH);  // turn on pull-up resistor
  digitalWrite(encoderPinB, HIGH);  // turn on pull-up resistor
  digitalWrite(clearButton, HIGH);

// encoder pin on interrupt 0 (pin 2)
  attachInterrupt(0, doEncoderA, CHANGE);
// encoder pin on interrupt 1 (pin 3)
  attachInterrupt(1, doEncoderB, CHANGE);

  Serial.begin(9600);
}


void loop(){
  if (lastReportedPos != encoderPos) {
    Serial.print("Index:");
    Serial.print(encoderPos, DEC);
    Serial.println();
    lastReportedPos = encoderPos;
  }
  if (digitalRead(clearButton) == LOW)  {
    encoderPos = 0;
  }
}

// Interrupt on A changing state
void doEncoderA(){
  // Test transition
  A_set = digitalRead(encoderPinA) == HIGH;
  // and adjust counter + if A leads B
  encoderPos += (A_set != B_set) ? +1 : -1;
}

// Interrupt on B changing state
void doEncoderB(){
  // Test transition
  B_set = digitalRead(encoderPinB) == HIGH;
  // and adjust counter + if B follows A
  encoderPos += (A_set == B_set) ? +1 : -1;
}
```

More encoder links

- a good explanation of grey codes and absolute encoders

---

this code did work better for me than most others, with good explanation

---

Interrupt Example (the Encoder interrupts the processor).

Uses both External Interrupt pins, after the initial read, does not read the state of the pins.

Fast encoder reading: using just interrupts

I also had to face with the problem of reading encoder signals and, after many trials, I'm happy to let you know a new way to deal with them, inspired by all the previous suggestions. I tried it on AMT encoder and it work really good. Unlikely, the other methods failed, the counting rate was too fast. To escape reading Arduino's port programming, I thought it may be even faster just using the interrupt pins. Here is the code:

```
//PIN's definition
#define encoder0PinA  2
#define encoder0PinB  3


volatile int encoder0Pos = 0;
volatile boolean PastA = 0;
volatile boolean PastB = 0;

void setup()
{

  pinMode(encoder0PinA, INPUT);
  //turn on pullup resistor
  //digitalWrite(encoder0PinA, HIGH); //ONLY FOR SOME ENCODER(MAGNETIC)!!!!
  pinMode(encoder0PinB, INPUT);
  //turn on pullup resistor
  //digitalWrite(encoder0PinB, HIGH); //ONLY FOR SOME ENCODER(MAGNETIC)!!!!
  PastA = (boolean)digitalRead(encoder0PinA); //initial value of channel A;
  PastB = (boolean)digitalRead(encoder0PinB); //and channel B
//To speed up even more, you may define manually th    e ISRs
// encoder A channel on interrupt 0 (Arduino's pin 2)
  attachInterrupt(0, doEncoderA, RISING);
// encoder B channel pin on interrupt 1 (Arduino's pin 3)
  attachInterrupt(1, doEncoderB, CHANGE);

}


void loop()
{
 //your stuf f....ENJOY! :D
}

//you may easily modify the code get quadrature..
//..but be sure this wouldn't let Arduino back!
void doEncoderA()
{
     PastB ? encoder0Pos--:  encoder0Pos++;
}

void doEncoderB()
{
     PastB = !PastB;
}
```

I hope this can help you.

by carnevaledaniele [04/2010]

____\\

Rotary encoder library for use in loop(). No example sketch given.

Here's a complete library code for working with Encoders:

```
#include <inttypes.h>
#include "HardwareSerial.h"

// 12 Step Rotary Encoder with Click //
// http://www .sparkfun.com/products/91  17 //
#define EncoderPinA 20          // Rotary Encoder Left Pin //
#define EncoderPinB 19  // Rotary Encoder Right Pin //
#define EncoderPinP 21   // Rotary Encoder Click //

// ====================================================================================== //
class Encoder
{
public:
        Encoder()
        {
                pinMode(EncoderPinA, INPUT);
                digitalW rite(EncoderPinA, HIGH);
                pinMode(EncoderPinB, INPUT);
                digitalW rite(EncoderPinB, HIGH);
                pinMode(EncoderPinP , INPUT);
                digitalW rite(EncoderPinP , HIGH);
```

```
            Position = 0;
            Position2 = 0;
            Max = 127;
            Min = 0;
            clickMultiply = 10;
        }

        void Tick(void)
        {
            Position2 = (digitalRead(EncoderPinB) * 2) + digitalRead(EncoderPinA);;
            if (Position2 != Position)
            {
                isFwd = ((Position == 0) && (Position2 == 1)) || ((Position == 1) && (Position2 == 3)) ||
                    ((Position == 3) && (Position2 == 2)) || ((Position == 2) && (Position2 == 0));
                if (!digitalRead(EncoderPinP)) { if (isFwd) Pos += clickMultiply; else Pos -= clickMultiply; }
                    else { if (isFwd) Pos++; else Pos--; }
                if (Pos < Min) Pos = Min;
                if (Pos > Max) Pos = Max;
            }
            Position = Position2;
        }

        int getPos(void)
        {
            return (Pos/4);
        }

        void setMinMax(int _Min, int _Max)
        {
            Min = _Min*4;
            Max = _Max*4;
            if (Pos < Min) Pos = Min;
            if (Pos > Max) Pos = Max;
        }

        void setClickMultiply(int _clickMultiply)
        {
            clickMultiply = _clickMultiply;
        }

    private:
        int clickMultiply;
        int Max;
        int Min;
        int Pos;
        int Position;
        int Position2;
        int isFwd;
    };
```

Interrupt Example (the Encoder interrupts the processor).

Uses both External Interrupt pins. Based on the Circuits@home code. Does not debounce. Notice that the Serial.print() functions can take milliseconds to return, so the interrupt code takes relatively long. This may change the behavior of the code when those statements are removed (as some bounces and even some transitions may be missed if the interrupt routine takes a comparatively long time). -Ed.

```
/*
  RotaryInterrupts - a port-read and interrupt based rotary encoder sketch
  Created by Joshua Layne (w15p), January 4, 201   1.
  based lar gely on: http://www .circuitsathome.com/mc  u/reading-rotary-encoder  -on-arduino
  Released into the public domain.
*/
```

```
#define ENC_A 2
#define ENC_B 3
#define ENC_POR T PIND
uint8_t bitShift = 2; // change to suit your pins (of   fset from 0,1 per port)
// Note: Y ou need to choose pins that have Interrupt c   apability.

int counter;
boolean ticT oc;

void setup()
{
  pinMode(ENC_A, INPUT);
  digitalW rite(ENC_A, HIGH);
  pinMode(ENC_B, INPUT);
  digitalW rite(ENC_B, HIGH);
  Serial.begin (1 15200);
  Serial.println("Start");
  counter = 0;
  ticToc = false;
  // Attach ISR to both interrupts
  attachInterrupt(0, read_encoder  , CHANGE);
  attachInterrupt(1, read_encoder  , CHANGE);
}

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

void read_encoder()
{
  int8_t enc_states[] = {0,-1,1,0,1,0,0,-1,-1,0,0,1,0,1,-1,0};
  static uint8_t encoderState = 0;
  static uint8_t stateIndex = 0;
  static uint8_t filteredPort = 0;
  uint8_t filter = 0x03; // base filter: 0b0000001   1
  filter <<= bitShift;

  Serial.print("raw port value: ");
  Serial.println(ENC_POR  T, BIN);

  Serial.print("filter bitmask: ");
  Serial.println(filter , BIN);

  filteredPort = ENC_POR  T & filter;
  Serial.print("filtered port state: ");
  Serial.println(filteredPort, BIN);

  Serial.print("old encoder state: ");
  Serial.println(encoderState, BIN);

  encoderState &= filter; // filter out everything except the rotary encoder pins
  Serial.print("filtered old encoder state: ");
  Serial.println(encoderState, BIN);

  encoderState <<= 2; // shift existing value two bits to the left
  Serial.print("filtered and shifted (<<2) old encoder state: ");
  Serial.println(encoderState, BIN);

  encoderState |= filteredPort; // add filteredport value
  Serial.print("old encoder state + port state: ");
  Serial.println(encoderState, BIN);

  stateIndex = encoderState >> bitShift;
  Serial.print("encoder state index: ");
  Serial.println(stateIndex, DEC);

  if (ticT oc) {
```

```
  Serial.print("counter tic: ");
  Serial.println(enc_states[stateIndex], DEC);
  counter += enc_states[stateIndex];
  Serial.print("counter: ");
  Serial.println(counter , DEC);
  }
  ticToc = !ticToc;

  Serial.println("----------");
}
```

Interrupt Example (the Encoder interrupts the processor).

Uses 1 Interrupt pin but misses half the state transitions. Claims to be fast but uses digitalRead() instead of direct port manipulation (see http://jeelabs.org/2010/01/06/pin-io-performance/)

<< 2011/06 >>

After try all examples, I think that I have the fastest and working form for the SparkFun Rotary encoder. Only one thing more: IT IS NEEDED TO INSERT A CAPACITOR BETWEEN CENTRAL PAD AND THE PAD B OF THE ENCODER.

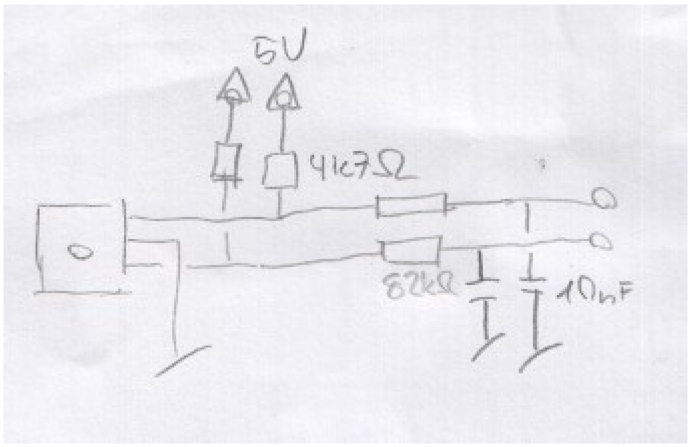
```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12,11,5,4,8,7);

#define encoder0PinA  2
#define encoder0PinB  3


volatile int encoder0Pos = 0;
volatile boolean PastB = 0;
volatile boolean update = false;

void setup()
{
  lcd.begin(16,2);
  lcd.print("Pos:");

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

  attachInterrupt(1, doEncoderB, FALLING);
}

void loop()
{
  if (update){
    update = false;
    PastB? encoder0Pos++:encoder0Pos--;
    lcd.setCursor(7,0);
    lcd.print("     ");
    lcd.setCursor(7,0);
    lcd.print(encoder0Pos,DEC);
  }
}
```

```
void doEncoderB()
{
  PastB=(boolean)digitalRead(encoder0PinA);
  update = true;
}
```

Debounce Circuit

7.8.2011 deif

After lots of tries i built a circuit to reduce the bouncing a lot.

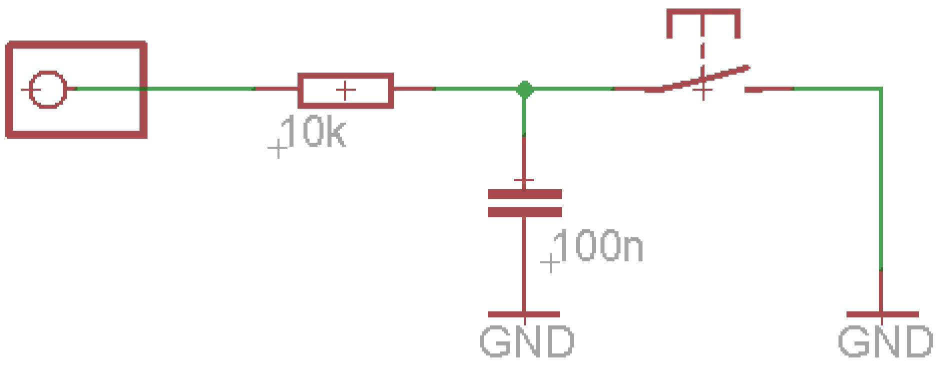


This is the minimal code i use:

```
ISR(INT0_vect){
        int a;
        a = PIND & 0x0c;

        if ((a == 0x0c) || (a == 0)){
                encoderCount++;
        }
        else {
                encoderCount--;
        }
}


void setupPinInterrupt(){
  EICRA = 0x01; //int0 on pinchange
  EIMSK = 0x01; //enable interrupt 0
  EIFR = 0; //clear flags
}
```

Another debouncer

(2011-12-11 by _bse_)



Works very well for me. Used in conjunction with internal pull-ups.

Interrupt Example (the Encoder interrupts the processor). Uses both External Interrupt pins.

Note: Uses digitalRead() instead of direct port manipulation in the interrupt routines (see http://jeelabs.org/2010/01/06/pin-io-performance/)

The XOR flavour method

XORing the actual PinB state with the previous PinA state, give us the increase or decrease of the encoder count. Few code lines, full encoder precision and achieve the Arduino max acquisition rate.

Have fun!

15 August 2011 by Bruno Chaparro

```
#define encoder0PinA 2
#define encoder0PinB 3
volatile unsigned int encoder0Pos = 0;
unsigned int tmp = 0;
unsigned int Aold = 0;
unsigned int Bnew = 0;
void setup() {
  pinMode(encoder0PinA, INPUT);
  pinMode(encoder0PinB, INPUT);
// encoder pin on interrupt 0 (pin 2)
  attachInterrupt(0, doEncoderA, CHANGE);
// encoder pin on interrupt 1 (pin 3)
  attachInterrupt(1, doEncoderB, CHANGE);
// set up the Serial Connection
  Serial.begin (1 15200);
}
void loop(){
  //Check each changes in position
  if (tmp != encoder0Pos) {
    Serial.println(encoder0Pos, DEC);
    tmp = encoder0Pos;
  }
  delay(500);
}
// Interrupt on A changing state
void doEncoderA(){
  Bnew^Aold ? encoder0Pos++:encoder0Pos--;
  Aold=digitalRead(encoder0PinA);
}
// Interrupt on B changing state
void doEncoderB(){
  Bnew=digitalRead(encoder0PinB);
  Bnew^Aold ? encoder0Pos++:encoder0Pos--;
}
```

I don't know how the above code works (XOR), but when combined with the PinChangeInt library, I can use any I/O pin as an interrupt and I was able to create 2 quadrature encoders on my carpet rover that work beautifully.

I am using a Solarbotics SB Freeduino board with an ATmega328P.

I would actually like to know more of how the above XOR code works. I had to say something sometime about this, but this is great.

Thank you!

Interrupt Example using the above "XOR method".

The XOR method driving an LCD and LED

Using the XOR method, by Bruno Chaparro, above, I put together a demo that drives an LCD bargraph and LED.

29 December 2011 Mark Amos

```
// increments/decrements a counter based on the movement of a rotary encoder and
// displays the results on an LCD in digital and bar graph form.
// rotary encoder is a 5 pin. Pins, left to right:
//   Encoder pin B - connect to D2 (interrupt 0)
//   +5 VDC
//   Encoder pin A - connect to D3 (interrupt 1)
//   NC
//   Ground
// Pin D5 is used to light up an LED connected to ground with a 1K resistor
// using PWM with brightness proportional to the encoder position.

#include <LiquidCrystal.h>
#define encoder0PinA 3
#define encoder0PinB 2
#define analogOutPin 5

LiquidCrystal lcd(13,12,1 1,10,9,8,7);
volatile unsigned int encoder0Pos = 0;
unsigned int tmp = 0;
unsigned int Aold = 0;
unsigned int Bnew = 0;

void setup() {
  pinMode(encoder0PinA, INPUT);
  pinMode(encoder0PinB, INPUT);

  lcd.begin(20,2);
  lcd.clear();

  // encoder A on interrupt 1 (pin 3)
  attachInterrupt(1, doEncoderA, CHANGE);
  // encoder B on interrupt 0 (pin 2)
  attachInterrupt(0, doEncoderB, CHANGE);
  // set up the Serial Connection
  Serial.begin (1 15200);
  Serial.println("Starting");
}

void loop(){
  //if position has changed, display it on serial and bar graph
  if (tmp != encoder0Pos) {
    tmp = encoder0Pos;
    Serial.println(tmp, DEC);
    lcd.setCursor(0,0);
    lcd.print(tmp);
    lcd.print("    ");
    lcd.setCursor(0,1);
    //scale the range of the LCD bar  graph from 0-1023 to 0-20 by dividing by 50
    for (int loopCnt = 0; loopCnt < (tmp / 50) +1 ; loopCnt++) {
      lcd.write(1);
    }
    lcd.print("                    ");
    //scale the encorer0Pos from 0 - 1023 to the range of the PWM (0 - 255) by dividing by 4.
    analogW rite(analogOutPin, tmp / 4);
  }
}

// Interrupt on A changing state
void doEncoderA(){
  // if Bnew = Aold, increment, otherwise decrement
  Bnew^Aold ? encoder0Pos++:encoder0Pos--;
  Aold=digitalRead(encoder0PinA);
  // check for underflow (< 0)
  if (bitRead(encoder0Pos, 15) == 1) encoder0Pos = 0;
  // check for overflow (> 1023)
  if (bitRead(encoder0Pos, 10) == 1) encoder0Pos = 1023;
  constrain(encoder0Pos, 0, 1023);
}

// Interrupt on B changing state
void doEncoderB(){
  Bnew=digitalRead(encoder0PinB);
  // if Bnew = Aold, increment, otherwise decrement
  Bnew^Aold ? encoder0Pos++:encoder0Pos--;
  // check for underflow (< 0)
  if (bitRead(encoder0Pos, 15) == 1) encoder0Pos = 0;
  // check for overflow (> 1023)
  if (bitRead(encoder0Pos, 10) == 1) encoder0Pos = 1023;
}
```

Interrupt Library (the Encoder interrupts the processor).

Utilizes any of the ATmega328P pins via the PinChangeInt library.
by GreyGnome

The AdaEncoder library was created for working with basic 2-pin quadrature encoders such as the following:
https://www.adafruit.com/products/377
http://www.sparkfun.com/products/9117
From the Introduction:
This library interfaces with 2-pin encoders (2 pins A and B, then a common pin C). It does not indicate every state change, rather, it reports only when the decoder is turned from one detent position to the next. It is interrupt-driven and designed to be fast and easy to use. The interrupt routine is lightweight, and the programmer is then able to read the direction the encoder turned at their leisure (within reason; what's important is that the library is reasonably forgiving). The library is designed to be easy to use (it bears repeating :-)) and it is reasonably immune to switch bounce.

See the project page at: http://code.google.com/p/adaencoder/
See a speed discussion at: http://code.google.com/p/adaencoder/wiki/Speed
See the PinChangeInt library project page at: http://code.google.com/p/arduino-pinchangeint/

Here's an example with two encoders connected. Encoder a is connected to pins 2 and 3, b is connected to 5 and 6:

```
#include <PinChangeInt.h> // necessary otherwise we get undefined reference errors.
#include <AdaEncoder .h>

#define a_PINA 2
#define a_PINB 3
#define b_PINA 5
#define b_PINB 6

int8_t clicks=0;
char id=0;

void setup()
{
  Serial.begin(1 15200);
  AdaEncoder::addEncoder('a', a_PINA, a_PINB);
  AdaEncoder::addEncoder('b', b_PINA, b_PINB);
}

void loop()
{
  encoder *thisEncoder;
  thisEncoder=AdaEncoder::genie(&clicks, &id);
  if (thisEncoder != NULL) {
    Serial.print(id); Serial.print(':');
    if (clicks > 0) {
      Serial.println(" CW");
    }
    if (clicks < 0) {
       Serial.println(" CCW");
    }
  }
}
```

Another Interrupt Library THAT REALLY WORKS (the Encoder interrupts the processor and debounces like there is no tomorrow).

by rafbuff

I tried most of the above but found that they do not reliably count steps up and down. Most have trouble with debouncing. While I use all the tricks regarding interrupt usage and efficiency above, I found this one to work best when precision counts...

```
/* interrupt routine for Rotary Encoders
   tested with Noble RE0124PVB 17.7FINB-24 http://www    .nobleusa.com/pdf/xre.pdf - available at pollin.de
   and a few others, seems pretty universal

   The average rotary encoder has three pins, seen from front: A C B
   Clockwise rotation A(on)->B(on)->A(of   f)->B(of f)
   CounterCW rotation B(on)->A(on)->B(of   f)->A(of f)

   and may be a push switch with another two pins, pulled low at pin 8 in this case
   raf@synapps.de 20120107

*/

// usually the rotary encoders three pins have the ground pin in the middle
enum PinAssignments {
  encoderPinA = 2,   // right
  encoderPinB = 3,   // left
  clearButton = 8    // another two pins
};

volatile unsigned int encoderPos = 0;  // a counter for the dial
unsigned int lastReportedPos = 1;   // change management
static boolean rotating=false;      // debounce management

// interrupt service routine vars
boolean A_set = false;
boolean B_set = false;


void setup() {

  pinMode(encoderPinA, INPUT);
  pinMode(encoderPinB, INPUT);
  pinMode(clearButton, INPUT);
 // turn on pullup resistors
  digitalW rite(encoderPinA, HIGH);
  digitalW rite(encoderPinB, HIGH);
  digitalW rite(clearButton, HIGH);

// encoder pin on interrupt 0 (pin 2)
  attachInterrupt(0, doEncoderA, CHANGE);
// encoder pin on interrupt 1 (pin 3)
  attachInterrupt(1, doEncoderB, CHANGE);

  Serial.begin(9600);  // output
}

// main loop, work is done by interrupt service routines, this one only prints stuf     f
void loop() {
  rotating = true;  // reset the debouncer

  if (lastReportedPos != encoderPos) {
    Serial.print("Index:");
    Serial.println(encoderPos, DEC);
    lastReportedPos = encoderPos;
  }
  if (digitalRead(clearButton) == LOW )  {
    encoderPos = 0;
  }
}
```

```
// Interrupt on A changing state
void doEncoderA(){
  // debounce
  if ( rotating ) delay (1);  // wait a little until the bouncing is done

  // Test transition, did things really change?
  if( digitalRead(encoderPinA) != A_set ) {  // debounce once more
    A_set = !A_set;

    // adjust counter + if A leads B
    if ( A_set && !B_set )
      encoderPos += 1;

    rotating = false;  // no more debouncing until loop() hits again
  }
}

// Interrupt on B changing state, same as A above
void doEncoderB(){
  if ( rotating ) delay (1);
  if( digitalRead(encoderPinB) != B_set ) {
    B_set = !B_set;
    //  adjust counter - 1 if B leads A
    if( B_set && !A_set )
      encoderPos -= 1;

    rotating = false;
  }
}
```

---

loop() Example, and the Encoder interrupts the processor

Uses a single External Interrupt pin and also requires loop() for debouncing.

Software Debounce

By jonfraz 09/11

First post from me also, hope it's useful.

I've been playing around with a pretty cheap mechanical encoder and found bouncing was a big issue when I experimented with the other code above.

Ideally I would debounce it with hardware, but I'm a noob and lack the knowledge/components. However, thanks to Hifiduino I managed to get my encoder working pretty well with a simple bit of software debouncing.

The code uses an interrupt to detect any signal change from the encoder, but then waits 2 milliseconds before calculating the encoder position:

```
*/ Software Debouncing - Mechanical Rotary Encoder */

#define encoder0PinA 2
#define encoder0PinB 4

volatile unsigned int encoder0Pos = 0;
```

```
static boolean rotating=false;

void setup() {
  pinMode(encoder0PinA, INPUT);
  digitalWrite(encoder0PinA, HIGH);
  pinMode(encoder0PinB, INPUT);
  digitalWrite(encoder0PinB, HIGH);

  attachInterrupt(0, rotEncoder, CHANGE);
  Serial.begin (9600);
}

void rotEncoder(){
  rotating=true;
  // If a signal change (noise or otherwise) is detected
  // in the rotary encoder, the flag is set to true
}

void loop() {
  while(rotating) {
    delay(2);
    // When signal changes we wait 2 milliseconds for it to
    // stabilise before reading (increase this value if there
    // still bounce issues)
    if (digitalRead(encoder0PinA) == digitalRead(encoder0PinB)) {
      encoder0Pos++;
    }
    else {
      encoder0Pos--;
    }
    rotating=false; // Reset the flag back to false
    Serial.println(encoder0Pos);
  }
}
```

Int0 & Int1 example using bitRead() with debounce handling and true Rotary Encoder pulse tracking

J.Carter(of Earth)

- This tiny amount of code is focused on keeping the interrupts fast and totally responsible for encoder position

```
// 'threshold' is the Debounce Adjustment factor for the Rotary Encoder.
//
// The threshold value I'm using limits it to 100 half pulses a second
//
// My encoder has 12 pulses per 360deg rotation and the specs say
// it is rated at a maximum of 100rpm.
//
// This threshold will permit my encoder to reach 250rpm so if it was connected
// to a motor instead of a manually operated knob I
// might possibly need to adjust it to 25000. However, this threshold
// value is working perfectly for my situation
//
volatile unsigned long threshold = 10000;


// 'rotaryHalfSteps' is the counter of half-steps. The actual
// number of steps will be equal to rotaryHalfSteps / 2
//
volatile long rotaryHalfSteps = 0;


// Working variables for the interrupt routines
//
volatile unsigned long int0time = 0;
volatile unsigned long int1time = 0;
volatile uint8_t int0signal = 0;
volatile uint8_t int1signal = 0;
volatile uint8_t int0history = 0;
volatile uint8_t int1history = 0;

void int0()
    {
    if ( micros() - int0time < threshold )
        return;
    int0history = int0signal;
    int0signal = bitRead(PIND,2);
    if ( int0history==int0signal )
        return;
    int0time = micros();
    if ( int0signal == int1signal )
        rotaryHalfSteps++;
    else
        rotaryHalfSteps--;
    }

void int1()
    {
    if ( micros() - int1time < threshold )
        return;
    int1history = int1signal;
    int1signal = bitRead(PIND,3);
    if ( int1history==int1signal )
        return;
    int1time = micros();
    }


void setup()
    {
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);

    attachInterrupt(0, int0, CHANGE);
    attachInterrupt(1, int1, CHANGE);
    }

void loop()
    {
    long actualRotaryTicks = (rotaryHalfSteps / 2);
    }
```

Multiple Encoders - for user interfacing

by Shannon White

This is an example that supports multiple encoders, and is designed for hand-turned encoders, not motors. This is meant for user interfacing, not hardware monitoring.

You need two pins for each encoder, so on a Nano, theoretically, you could put 7 encoders on the digital pins and 3 more on the analog pins (with some code rewriting). You might even be able to squeeze one more out of the left over digital and analog pins. I have tested it only with two encoders (all I needed), but it worked flawlessly and I turned both at the same time. I could not detect misses through watching the output and feeling the detents tick by. It is good enough for a user interface.

If you needed many encoder knobs, you could use a Nano to read in and track the values, and then send the updates to another microcontroller to do something about it. You might need to save a couple of pins for the rx/tx.

This could be improved by converting it to use PINx to read the values simultaneously instead of digitalRead().

It includes a software debouncer. It detects all four states of transition, so every knob turn gets registered. It does not use any external interrupts, which would limit the number of encoders to 1 (full read) or 2 (every other position read). Since we are not using interrupts, the accuracy at speeds faster than hand turning would be awful, the faster it went the worse it would be.

Connections and Layout:

- Connect the common (middle) pin on each encoder to 5v.
- Connect a 10k resistor between ground and each pin on the board you are going to connect to an encoder. This pulls the pins low to avoid fluttering mis-reads.
- Connect the appropriate board pins the the A and B pins on the encoder. You will need to then define which pins these are in the code.
- Make sure all encoder-linked pins on your board also have one of those grounding resistors connected (you didn't miss place one).

Hope this helps someone.

```
/// Multiple Encoders
///  Designed for hand-turned encoders, not accurate enough for motors
///  Developed as an multi-encoder interface to a Nano


/////////////// Prepare some global variables
// number of encoders
int numEnc = 2;
// define the A pins on each encoder (in order)
int encPinA[2] = {2,6};
// define the B pins on each encoder (in order)
int encPinB[2] = {3,7};
// last mode of each pin (HIGH/LOW) for comparison - see if it changed
int lastModeA[2];
int lastModeB[2];
// current mode of each encoder pin (HIGH/LOW)
int curModeA[2];
int curModeB[2];
// current and last encoder positions
int encPos[2];
int encPosLast[2];
// utility variables
int change = 0;
int c = 0;
```

```
///////////////// Initialize the program
void setup () {
  Serial.begin(9600);
  // set up each encoder's values
  for (c=0;c<numEnc;c++) {
    // tell us what it is doing
    Serial.print("Initializing encoder ");
    Serial.println(c);
    // set the pins form INPUT
    pinMode(encPinA[c], INPUT);
    pinMode(encPinB[c], INPUT);
    // set the modes and positions - on first read, it may change position once
    //   depending on how the encoders are sitting (having a HIGH position that
    //   gets compared to the initial LOW setting here in the first iteration of
    //   the loop).
    lastModeA[c] = LOW ;
    lastModeB[c] = LOW ;
    curModeA[c] = LOW ;
    curModeB[c] = LOW ;
    encPos[c] = 0;
    encPosLast[c] = 0;
  }
  Serial.println("---- Ready -------------------------------");
}


///////////////// Body of the program
void loop () {
  // read in current values
  // set the change variable to 0.  If there is an encoder position change,
  //   this gets changed to 1.  This lets a later portion of the loop that
  //   a change has been made and it doesn't have to compare all the modes
  //   again.
  change = 0;
  // loop through each of the encoders
  for (c=0;c<numEnc;c++) {
    // read the current state of the current encoder's pins
    curModeA[c] = digitalRead(encPinA[c]);
    curModeB[c] = digitalRead(encPinB[c]);
    // compare the four possible states to figure out what has happened
    //   then encrement/decrement the current encoder's position
    if (curModeA[c] != lastModeA[c]) {
      if (curModeA[c] == LOW) {
        if (curModeB[c] == LOW) {
          encPos[c]--;
        } else {
          encPos[c]++;
        }
      } else {
        if (curModeB[c] == LOW) {
          encPos[c]++;
        } else {
          encPos[c]--;
        }
      }
    }
    if (curModeB[c] != lastModeB[c]) {
      if (curModeB[c] == LOW) {
        if (curModeA[c] == LOW) {
          encPos[c]++;
        } else {
          encPos[c]--;
        }
      } else {
        if (curModeA[c] == LOW) {
          encPos[c]--;
        } else {
          encPos[c]++;
        }
      }
    }
    // set the current pin modes (HIGH/LOW) to be the last know pin modes
    //   for the next loop to compare to
    lastModeA[c]=curModeA[c];
    lastModeB[c]=curModeB[c];
    // if this encoder's position changed, flag the change variable so we
    //   know about it later
    if (encPos[c] != encPosLast[c]) {
      change = 1;
    }
  }

  if (change == 1) {
    // if an encoder has changed, do something with that information
    // here, I am just going to print all the encoder's positions
    //   if any of them change
    for (c=0;c<numEnc;c++) {
      Serial.print("  Position");
      Serial.print(c);
      Serial.print(": ");
      Serial.print(encPos[c]);
      encPosLast[c]=encPos[c];
    }
    Serial.println();
    // debounce - if there has been a change, wait for a bit (so to speak) to let the
    //   bounces settle - change to your liking
    delay(100);
  }
}
```

On Interrupts

by GreyGnome

The ATmega328P has two different kinds of interrupts: “external”, and “pin change”. There are only two external interrupt pins, INT0 and INT1, and they are mapped to Arduino pins 2 and 3. These interrupts can be set to trigger on RISING or FALLING signal edges, or on low level. Most of the sketches and libraries given on this page use one or two of the External Interrupt pins. The interrupt is theoretically very quick because you can set the hardware to tell you how you want the interrupt to trigger. Each pin can have a separate interrupt routine associated with it.

On the other hand the pin change interrupts can be enabled on any or all of the ATmega328p's signal pins. They are triggered equally on RISING or FALLING signal edges, so it is up to the interrupt code to determine what happened (did the signal rise, or fall?) and handle it properly. Furthermore, the pin change interrupts are grouped into 3 “port”s on the MCU, so there are only 3 interrupt vectors (subroutines) for the entire body of 19 pins. This makes the job of resolving the action on a single interrupt even more complicated. The interrupt routine should be fast, but complication is the enemy of speed.

Interrupts disrupt the normal flow of the program. This can be a bit of a curse, although the compiler adds some code for you in order to take away much of the pain of the interrupt. The benefit is that, unlike polling in loop(), you have a better chance of getting all the transitions from your device.

What if, for example, the Arduino is talking to an I2C device via the Wire library? You may not realize that the I2C library has a busy wait portion in it, which means that your 16MHz processor is busy waiting on a 100khz serial communication. That communication is actually being performed in hardware, so there's no reason that the processor must wait, but that's how the library was designed. So if you use polling in loop() to check on your rotary encoder, you may miss a quick transition if your code is waiting for a communication to end.

This is not an issue with an interrupt, because your rotary encoder will trigger the interrupt routine even if the Wire library is busy in a while() loop. For this reason, you should be careful using code that polls a switch in loop() on more complicated projects, and understand how long each section of your code will take in worse case scenarios.

On Speed

by GreyGnome

If your application is speed-critical, know that digitalRead() and digitalWrite() are relatively slow. See http://jeelabs.org/2010/01/06/pin-io-performance/.

From that link:

Let me just conclude with: there are order-of-magnitude performance implications, depending on how you do things. As long as you keep that in mind, you’ll be fine.
To get a closer look at the implications, I actually measured the speed.

Speed Tests

From http://arduino-pinchangeint.googlecode.com/files/PinChangeInt%20Speed%20Test-1.3.pdf
Tested External Interrupts against Pin Change Interrupt, to look at the relative speeds vs. the External Interrupts, and also to better judge the performance hit of using digitalRead() and digitalWrite().

Test	Pin triggered	Interrupt Type	Loop time, 100,000 iterations, ms.	Average time per loop (us)
1	2	External	1420	14.20
2	2	Pin Change	2967	29.67

Next, we turned on the LED Pin using direct port manipulation, then turned it on using digitalWrite() under Arduino 022 and Arduino 1.0. This test is enabled by #define'ing the COMPAREWRITES compiler directive in the source code. If DIRECTPORT is #define'ed the LED pin is turned on and off using direct port manipulation. If #undef'ed, digitalWrite() is used to manipulate the LED pin.

Test	Pin triggered	Interrupt Type	Arduino Version	LED pin turned on using...	Loop time, 100,000 iterations, ms.	Average time per loop (us)
1	2	External	022	Direct Port Manipulation	1603	16.03
2	2	External	022	digitalWrite()	2232	22.32
3	2	External	1.0	digitalWrite()	2245	22.45

Now read the LED Pin using direct port manipulation, then read it using digitalWrite() under Arduino 022 and Arduino 1.0. This test is enabled by #define'ing the COMPAREREADS compiler directive in the source code. If DIRECTPORT is #define'ed the LED pin is read using direct port manipulation. If #undef'ed, digitalRead() is used to manipulate the LED pin.

Test	Pin trig-gered	Interrupt Type	Arduino Version	LED pin turned on using...	Loop time, 100,000 iterations, ms.	Average time per loop (us)
1	2	External	022	Direct Port Manipulation	1559	15.59
2	2	External	022	digitalRead()	2188	21.88
3	2	External	1.0	digitalRead()	2189	21.89

Share

NEWSLETTER

Enter your email to sign up

©2017 Arduino

Copyright Notice

Contact us