+10 31 4 help

Using a rotary encoder with AVR Micro controller

I'm having trouble getting a rotary encoder to work properly with AVR micro controllers. The encoder is a mechanical ALPS encoder, and I'm using Atmega168.

Clarification

I have tried using an External Interrupt to listen to the pins, but it seems like it is too slow. When Pin A goes high, the interrupt procedure starts and then checks if Pin B is high. The idea is that if Pin B is high the moment Pin A went high, then it is rotating counter clock-wise. If Pin B is low, then it is rotating clock-wise. But it seems like the AVR takes too long to check Pin B, so it is always read as high.

I've also tried to create a program that simply blocks until Pin B or Pin A changes. But it might be that there is too much noise when the encoder is rotated, because this does not work either. My last attempt was to have a timer which stores the last 8 values in a buffer and checks if it is going from low to high. This did not work either.

I have tried scoping the encoder, and it seems to use between 2 and 4ms from the first Pin changes till the other Pin changes.

microcontroller avr encoder atmega

edited Aug 29 '08 at 17:47

asked Aug 29 '08 at 16:58
Marius
29.8k 13 83 121

5 Answers

I have a webpage about rotary encoders and how to use them, which you might find useful.

Unfortunately without more information I can't troubleshoot your particular problem.

Which microcontroller pins are connected to the encoder, and what is the code you're currently using to decode the pulses?

Ok, you're dealing with a few different issues, the first issue is that this is a mechanical encoder, so you have to deal with switch noise (bounce, chatter). The data sheet indicates that it may take up to 3mS for the parts to stop bouncing and creating false outputs.

You need to create a debounce routine. The simplest of which is to continuously check to see if A goes high. If it does, start a timer and check it again in 3 ms. If it's still high, then you can check B - if it's not high then you ignore the spurious pulse and continue looking for A high. When you check B, you look at it, start a timer for 3 ms, and then look at B again. If it was the same both times, then you can use that value - if it changes within 3 ms then you have to do it again (read B, wait 3 ms, then read it again and see if it matches).

The atmega is fast enough that you shouldn't have to worry about these checks going slowly, unless you're also running a slow clock speed.

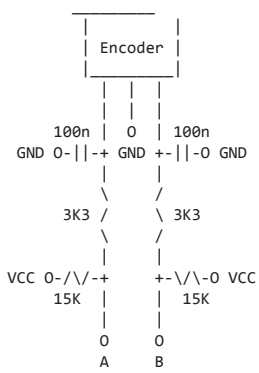
Once you deal with the mechanical noise, then you want to look at a proper gray code routine - the algorithm you're following won't work unless you also decrement if A is high when B goes low. Generally people store the last value of the two inputs, and then compare it to the new value of the two inputs and use a small function to increment or decrement based on that. (Check out the heading "high resolution reading" on the website I mentioned above for the table). I combine the two readings into a four bit number and use a simple array to tell me whether I increment or decrement the counter, but there are solutions that are even more advanced, and optimize for code size, speed, or ease of code maintenance.

edited May 7 '12 at 8:54
XTL
515 4 16

answered Aug 29 '08 at 19:37
Adam Davis
50.8k 39 190 293

Adding an analog lowpass filter greatly improves the signal. With the lowpass filter, the code

on the AVR was really simple.



Ah, the wonders of ASCII art :p

Here is the program on the AVR. Connect A and B to input PORTB on the avr:

```
#include <avr/io.h>

#define PIN_A (PINB&1)
#define PIN_B ((PINB>>1)&1)

int main(void){
    uint8_t st0 = 0;
    uint8_t st1 = 0;
    uint8_t dir = 0;
    uint8_t temp = 0;
    uint8_t counter = 0;
    DDRD = 0xFF;
    DDRB = 0;
    while(1){
    if(dir == 0){
        if(PIN_A & (!PIN_B)){
            dir = 2;
        }else if(PIN_B & (!PIN_A)){
            dir = 4;
        }else{
            dir = 0;
        }
    }else if(dir == 2){
        if(PIN_A & (!PIN_B)){
            dir = 2;
        }else if((!PIN_A) & (!PIN_B)){
            counter--;
            dir = 0;
        }else{
            dir = 0;
        }
    }else if(dir == 4){
        if(PIN_B & (!PIN_A)){
            dir = 4;
        }else if((!PIN_A) & (!PIN_B)){
            counter++;
            dir = 0;
        }else{
            dir = 0;
        }
    }else if(PIN_B & PIN_A){
        dir = 0;
    }
        PORTD = ~counter;
    }
    return 0;
}
```

This code works unless you rotate the encoder really fast. Then it might miss a step or two, but that is not important, as the person using the encoder won't know how many steps they have turned it.

edited Oct 22 '12 at 11:34

answered Nov 23 '08 at 11:41

Marius
29.8k 13 83 121

1 It is ok as a "hobbyist" solution. However, the extra hardware (resistors/capacitors) is not to be underestimated. That is why software debouncing is a "better" solution (IMHO). – Philibert Perusse Feb 2 '09 at 22:45

1 Nice thing about the hardware solution, though, is it provides additional protection from ESD from the user. Not optimized for that, of course, but a little bonus. – Adam Davis Feb 2 '09 at 22:54

1 I think your circuit doesn't show a passive LP filter from the encoder to the µC. You have to exchange the positions of the resistors and capacitors. – Thomas Vultura Aug 17 '09 at 8:08

@ThomasVultura thanks, fixed! – Marius Oct 22 '12 at 11:34

Note that this filter will result in VCC when the switch is open, and 18% of VCC when the switch is closed. A filter like that recommended in this bourns datasheet will go cleanly to VCC and GND, as well as using one fewer value of part. – Nick Johnson Apr 23 '14 at 10:42

Speed should not be a problem. Mostly all mechanical switches need debounce routines. If you wanna do this with interrupts turn off the interrupt when it triggers, start a timer that will turn it back on after a couple of ms. Will keep your program polling-free >:)

edited Nov 2 '08 at 18:10

answered Oct 21 '08 at 20:30

monowerker
2,262 16 17

What exactly are you having problems with? I assume you've been able to hook the pins of the encoder to your PIC as per the technical specifications linked on the Farnell page you gave, so is the problem with reading the data? Do you not get any data from the encoder? Do you not know how to interpret the data you're getting back?

answered Aug 29 '08 at 17:18

Daniel Jennings
2,574 2 17 40

```
/* into 0 service rutine */
if(CHB)
{
  if(flagB)
   Count++;
  FlagB=0;
}
else
{
  if(FlagB)
   count--:
  FlagB=0:
}

/* into 1 service rutine */
FlagB=1;

/* make this give to you a windows time of 1/4 of T of the encoder resolution
   that is in angle term: 360/ (4*resolution)
 */
```

edited Dec 30 '08 at 14:58

dbr
73.4k 34 190 267

answered Nov 3 '08 at 0:02

camilo