International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-1 Issue-1, June 2012

Decoding Quadrature Signals using AT90CAN128

Akhilesh Gannavarapu, Vatte Suresh

Abstract: ***A code is written for decoding the quadrature signals from the rotary encoder, to get a pulse train along with direction signal, which shows one transition period, and the direction of the rotation, whether in clockwise or anti clockwise directions respectively. It is especially important for machines in Industries which have to monitor the transition periods, or for machines used for Image Processing. Using AVR Studio 4, and AVR GCC, a C code for decoding the quadrature signals is being presented, which gives the pulse for one cycle and the direction in which the encoder is being rotated. With a change in direction, there is a change in the direction pulse, and the pulse indicates the completion of one transition cycle. It is explained in detail in the documentation.***

Index Terms: ***Quadrature decoder, AVR Studio v. 4.18, AVR GCC, transition state***

I. INTRODUCTION

The quadrature decoder produces two square wave signals, 90 degrees out of phase with one another. When the encoder is rotated in one direction, Phase A leads Phase B. When the encoder is rotated in the other direction, Phase B leads Phase A. With a quadrature encoder, you can both count the number of rotation pulses and detect the direction of rotation. Fig 1 shows how a quadrature signal looks like, when the encoder is rotated in both clockwise and counterclockwise directions. The transition states and flow chart also play a prominent role in designing an efficient code for the Quadrature decoder, while following the Data Sheet is of great importance, as to use the correct pins for Interrupts, or for Watch dog, or for using various other operations to be performed with in the code.

Fig 2 shows the Flow chart, and Fig 3 shows the transition diagram, which is explained in greater detail in the documentation.

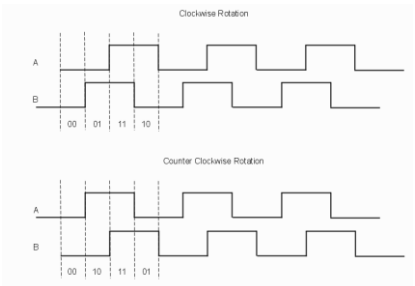
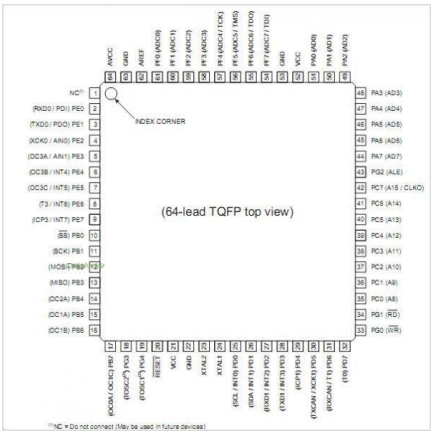


Fig1: Quadrature phase shifted signals.

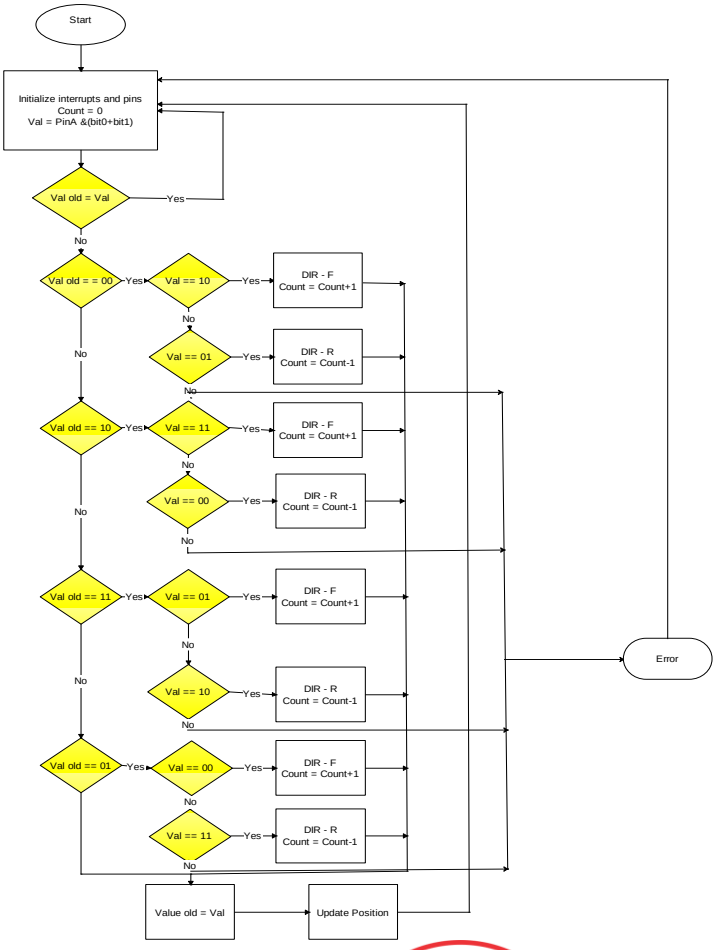
Revised Manuscript Received on 30 June 2012
***Correspondence Author(s)**
Gannavarapu Akhilesh*, Department of Electrical and Electronic Engineering, JNTU-H, Hyderabad, India.
Vatte Suresh, Department of Electrical and Electronic Engineering, JNTU-H, Hyderabad, India.

Pin Configuration of AT90CAN 128



II. PROCEDURE

A. Flow Chart



Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

B. Transition State and Transition Table

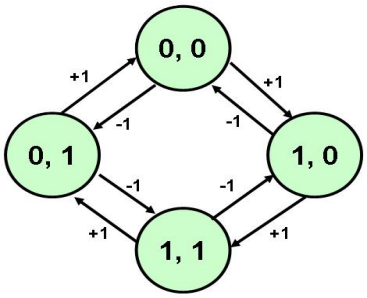


Fig 3: Transition Diagram.

State	Clockwise Transition To Here	Counter-Clockwise Transition to Here
0,0	0,1 to 0,0	1,0 to 0,0
1,0	0,0 to 1,0	1,1 to 1,0
1,1	1,0 to 1,1	0,1 to 1,1
0,1	1,1 to 0,1	0,0 to 0,1

Table 1: Transition States.

The above state transition diagram shows the direction in which the encoder is rotating. From the flow chart, it's apparent that each state can go in two directions, i.e. the forward direction, or the backward direction. It is further illustrated in the document how the path has been chosen for both, the forward rotation and the backward rotation. The State Transition table gives a better idea about the states which correspond to a number (1, 2, 3 and 4), which is used in the written code.

C. Concept and explanation

The quadrature decoder produces two square wave signals, 90 degrees out of phase with one another. When the encoder is rotated in one direction, Phase A leads Phase B. When the encoder is rotated in the other direction, Phase B leads Phase A. With a quadrature encoder, you can both count the number of rotation pulses and detect the direction of rotation.

Software Decoding

A software quadrature decoder is often implemented as an interrupt-driven state machine, based upon a state table.

Let's build a state table by referring to the phase diagram, above. For the moment, just consider the clockwise rotation case.

Starting with the Phase A = 0, Phase B = 0 conditions and moving left to right, the states you encounter are as such.

	Phase A	Phase B
State 0	0	0
State 1	1	0
State 2	1	1
State 3	0	1

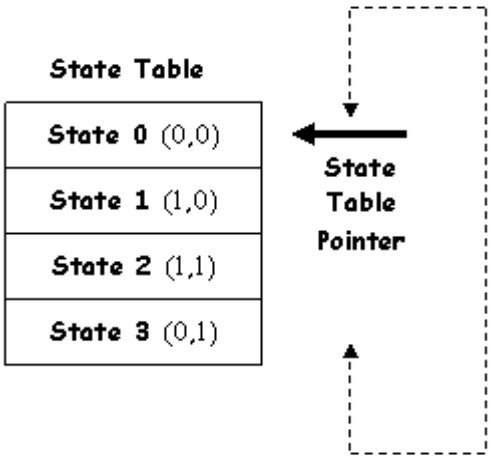
Now look at the diagram for counterclockwise rotation. Find the Phase A = 0, Phase B = 0 state. Moving left to right, the next state will be Phase A = 0, Phase B = 1. Then Phase A= 1, Phase B = 1, and finally Phase A = 1, Phase B = 0.

Compare the CCW states with the CW state sequence in the table above.

Phase A = 0, Phase B = 0 is State 0. Phase A = 0, Phase B = 1 corresponds to State 3. The next corresponds to State 2, and the last corresponds to State 1. Notice that the state sequence is the same for either direction of rotation. Only the direction you take to step through the state table varies.

So you can use the same state table for both clockwise and counterclockwise rotation.

The state table is actually circular, with State 3 wrapping around to State 0, and State 0 wrapping around to State 3, as shown in the figure below.



If you measure your current state and find it in the table, the next state must be either the state directly above or the state directly below your current state. If the state in question is the one directly above your current state, then the direction of rotation is counterclockwise. If the state in question is the one directly below your current state, then the direction of rotation is clockwise.

Note that "Clockwise," "Counterclockwise," "Phase A," and "Phase B" are relative terms. In reality, the phases can be reversed with respect to the direction of rotation. In the real world, if you build a decoder that's telling you the wrong direction of rotation, just swap the Phase A and Phase B connections.

Quadrature Decoder Algorithm

Let's assume that you want a counter to increment when Phase A leads Phase B, and you want the counter to decrement when Phase B leads Phase A.

Set up your state table to match the states in the table above, and establish a state table pointer. Configure your interrupts for interrupt-on-change for both the Phase A and Phase B inputs.

Initialization:

Initialize the counter.

Measure the current state of the Phase A and Phase B inputs. Locate that state in the table. Initialize your state pointer to that state.

Enable interrupts.

Interrupt routine:

If the state is the one preceding your state pointer, decrement the counter.

If the state is the one following your state pointer, increment the counter.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

Point the state pointer to the current state.

The resolution produced by this algorithm depends upon your interrupt configuration.

If you generate interrupts on only one transition, say the positive-going transition of either the Phase A or Phase B line, you will produce half the number of counts per revolution as there are cells in your track. For a 250 cell track, you will recover 125 counts per revolution.

If you configure your system to generate interrupts on two transitions -- say, the positive-going transitions on both the Phase A and Phase B lines -- you will achieve the same resolution as your wheel. If you have 250 cells in your track, you will recover 250 counts per revolution.

If you are able to generate interrupts on all four transitions, that is, both the positive and negative-going transitions on both the Phase A and Phase B lines, you will double your wheel's resolution. A 250 cell track will produce 500 counts per revolution.

III. CODE

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

#ifndef ENCODER_H
#define ENCODER_H
#define direction_bit PD2;
#define enc_position PD3;
he external interrupt pins on the processor are:

NT0 -> PD0 (PORTD, pin 0)
NT1 -> PD1 (PORTD, pin 1)
// Interrupt Configuration
#define ENCA_SIGNAL SIG_INTERRUPT0 // Interrupt signal for Ch. A
#define ENCA_INT INT0 // matching INTx bit in GIMSK/EIMSK
#define ENCA_ICR EICRA // matching Int. Config Register (MCUCR,EICRA/B)
#define ENCA_ISC00 ISC00 // matching Interrupt Sense Config bit0
#define ENCA_ISC01 ISC01 // matching Interrupt Sense Config bit1

#define ENCB_SIGNAL SIG_INTERRUPT1 //
#define ENCB_INT INT1 //
#define ENCB_ICR EICRA // matching Int. Config Register (MCUCR,EICRA/B)
#define ENCB_ISC10 ISC10 // matching Interrupt Sense Config bit0
#define ENCB_ISC11 ISC11 // matching Interrupt Sense Config bit1


// PhaseA Port/Pin Configuration PORTD 7 11
// *** PORTx, DDRx, PINx, and Pxn should all have the same letter for "x" ***
#define ENC_A_PORT PORTD // PhaseA port register
#define ENC_A_DDR DDRD // PhaseA port direction register
#define ENC_A_PORTIN PIND // PhaseA port input register
#define ENC_A_PIN PD0 // PhaseA port pin

// Phase B quadrature encoder output should connect to this direction line:
// *** PORTx, DDRx, PINx, and Pxn should all have the same letter for "x" ***
#define ENC_B_PORT PORTD // PhaseB port register
#define ENC_B_DDR DDRD // PhaseB port direction register
#define ENC_B_PORTIN PIND // PhaseB port input register
#define ENC_B_PIN PD1 // PhaseB port pin
// configure interrupts for any change triggering
  EICRA |= (1 << ENCA_ISC00); //SET
  EICRA &= ~(1 << ENCA_ISC01); //CLEAR

  // configure interrupts for any change triggering
  EICRA |= (1 << ENCB_ISC10); //SET
  EICRA &= ~(1 << ENCB_ISC11); //CLEAR
ISR (INT0_vect) // FIRES ON A TRANSITION (L->H & H->L)
{


  // DETERMINE WHICH STATE WE ARE IN...
  if ((ENC_A_PORTIN & (1<<ENC_A_PIN)) == 0) // CHANNEL A PIN IS LOW
  {
      if ((ENC_B_PORTIN & (1<<ENC_B_PIN)) == 0) // CHANNEL B PIN IS LOW
      {

        if(LastState == 3)
        {
          dir=0;
        PORTD &=~(1<<PD3);

        } else {

          dir=1;
        PORTD &=~(1<<PD3);
      }
       EncoderState = 0;
       } else {
if(LastState == 0)
        {
          dir=0;
        PORTD |=(1<<PD3);

        } else{

          dir=1;
        PORTD &=~(1<<PD3);
```


Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

```
72
73              }
74          }EncoderState = 1;
75    }
76if(dir==0)
77    {
78    PORTD &= ~(1<<PD2);
79    }
80     else {
81    PORTD |= (1<<PD2);
82    }
83    LastState = EncoderState;
}
```

Table 2: An example from the working code (Note : Not complete)

IV. RESULTS

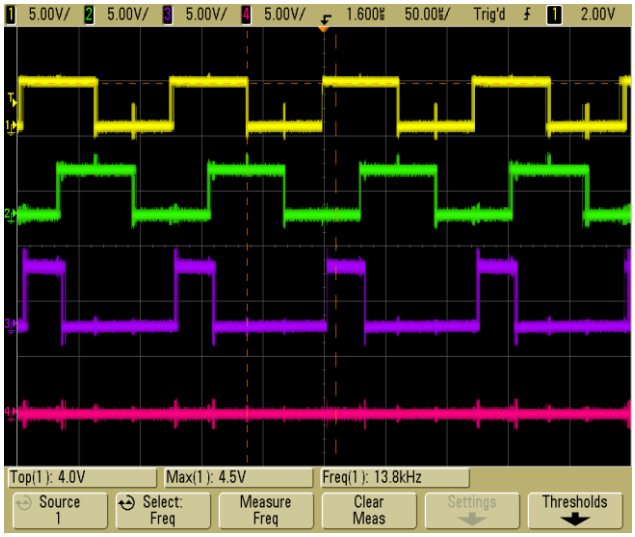
Clockwise Direction

Yellow – The first input (Phase A)

Green – The second input (Phase B)

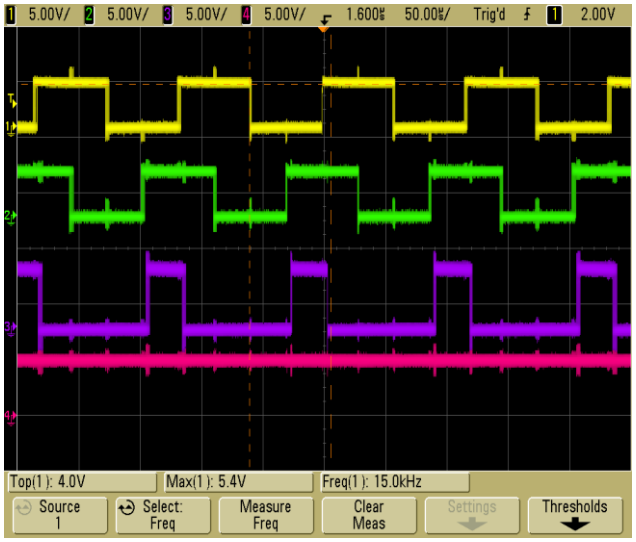
Pink – The first output (Pulse count)

Blue – The second output (direction)



Anti-Clockwise Direction

Yellow – The first input (Phase A)
Green – The second input (Phase B)
Pink – The first output (Pulse count)
Blue – The second output (Direction)



V. CONCLUSIONS

Calculations:

1. In Forward Direction:

Distance from the falling edge of 1st waveform to the rising edge of the pulse – 14.4 micro seconds.

Distance from the falling edge of 2nd waveform to the rising edge of the pulse – 6.8 micro seconds.

2. In Backward Direction:

Distance from the falling edge of the 1st waveform to the rising edge of the pulse – 12 micro seconds.

Distance from the falling edge of 2nd waveform to the rising edge of the pulse – 24.8 micro seconds.

- The size of the code is around 600 bytes. A processor with 8 pins would be enough (2 inputs, 2 outputs, and 2 interrupts), as well as on its architectural basis.
- Noticed on a number of occasions that there is a delay, when the direction changes. Though for a rotary encoder, that won't cause any problems.
- Although the code can be made simpler, using a specific analogy, it won't meet the 2 interrupt configuration, or cover the entire transition. Hence leaving chances for error to occur. This is why the more complicated route has been taken.

REFERENCES

1. Data Sheet of AVR AT90CAN128

AUTHOR PROFILE

Gannavarapu Akhilesh Completed his Bachelor of Technology from JNTU H, published various papers in International Journals and Conferences such as ICCA '12, IISTE, IJSCE etc to name a few, including publishing a book on his thesis work about TCSC. He has worked with Professors from Electrical Engineering department in Germany for a period of 3 months.

Vatte Suresh Completed his Bachelor of Technology from JNTU H, published various papers in International Journals such as IISTE, IJSCE etc to name a few.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)