

Assignment On:

|  |
| --- |
| CSE3027.1 : Introduction to Embedded Systems |

Submitted To:

|  |
| --- |
| Faculty Name : Rezwan Khan |
| Department of English,Southeast University. |

Submitted By:

|  |  |  |
| --- | --- | --- |
| Name | ID | Group Name |
| 1.Md. AL Emran | 2017000000030 | Alpha |
| 2.Name: Al-Amin Hosain | 2016100000142 | Alpha |
| 3.Mohammad Al Imran Sagor | 2015200000062 | Alpha |



Final Code

**1.Absolute Position Encoder**

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <stdio.h>

#define F\_CPU 16000000UL

#define FOSC 16000000

#define BAUD 9600 /\*\*< Baud Rate in bps. refer page 179 of 328p datasheet. \*/

#define MYUBRR FOSC/16/BAUD-1

#define numSlots 20

#define DELAY 100

int result;

volatile int count = 0;

volatile int revolution = 0;

volatile uint8\_t flag = 0;

void USART\_init(unsigned int ubrr) {

UCSR0C = (0 << USBS0) | (3 << UCSZ00);

UCSR0A = 0b00000000; /// Step 2. Set UCSR0A in Normal speed, disable multi-proc

UBRR0H = (unsigned char) (ubrr >> 8); /// Step 3. Load ubrr into UBRR0H and UBRR0L

UBRR0L = (unsigned char) ubrr;

UCSR0B = 0b00011000; /// Step 4. Enable Tx Rx and disable interrupt in UCSR0B

}

int USART\_send(char c, FILE \*stream) {

while (!(UCSR0A & (1 << UDRE0))) { ; }

UDR0 = c; /// Step 2. Write char to UDR0 for transmission

}

int USART\_receive(FILE \*stream) {

while (!(UCSR0A & (1 << RXC0)));

return UDR0; /// Step 2. Get and return received data from buffer

}

ISR(PCINT0\_vect) {

if (count < 20) {

count++;

} else {

revolution++;

count = 0;

flag = 1;

}

}

void init\_pcint0() {

PORTD |= (1 << PD2); /// pullup

EICRA |= (1 << ISC01);

EICRA |= (1 << ISC00);

EIMSK |= (1 << INT0); /// Enable INT0\*/

DDRB &= ~(1 << PB2);

PORTB |= (1 << PB2);

PCICR |= (1 << PCIE0);

PCIFR |= (1 << PCIF0);

PCMSK0 |= (1 << PCINT0);

}

void init\_ADC() {

ADMUX = 0b01000000;

ADCSRA = 0b10000111; }

uint16\_t read\_ADC(uint8\_t ch) {

ch &= 0b00000111; // AND operation with 7

ADMUX = (ADMUX & 0xF8) | ch; // clears the bottom 3 bits before ORing

ADCSRA |= (1 << ADSC);

while (ADCSRA & (1 << ADSC)) ;

return (ADC);

}

uint8\_t graycode\_table[64] = { 0b00000000, 0b00000001, 0b00000011, 0b00000010,

0b00000110, 0b00000111, 0b00000101, 0b00000100, 0b00001100, 0b00001101,

0b00001111, 0b00001110, 0b00001010, 0b00001011, 0b00001001, 0b00001000,

0b00011000, 0b00011001, 0b00011011, 0b00011010, 0b00011110, 0b00011111,

0b00011101, 0b00011100, 0b00010100, 0b00010101, 0b00010111, 0b00010110,

0b00010010, 0b00010011, 0b00010001, 0b00010000, 0b00110000, 0b00110001,

0b00110011, 0b00110010, 0b00110110, 0b00110111, 0b00110101, 0b00110100,

0b00111100, 0b00111101, 0b00111111, 0b00111110, 0b00111010, 0b00111011,

0b00111001, 0b00111000, 0b00101000, 0b00101001, 0b00101011, 0b00101010,

0b00101110, 0b00101111, 0b00101101, 0b00101100, 0b00100100, 0b00100101,

0b00100111, 0b00100110, 0b00100010, 0b00100011, 0b00100001, 0b00100000

};

int main() {

init\_ADC();

USART\_init(MYUBRR);

stdout = fdevopen(USART\_send, NULL);

stdin = fdevopen(NULL, USART\_receive);

init\_pcint0();

sei(); /// Enable global interrupt

//Defining angle range;

double minAngle[64];

double maxAngle[64];

double anglePerDivision = 5.625;

DDRB = 0x3F;

PORTB |= (1 << PD5);

PORTB &= ~(1 << PD6);

while (1) {

uint8\_t i = 0;

uint8\_t j = 0;

result = read\_ADC(0b01000000);

\_delay\_ms(100);

for (i = 0; i < 64; i++) {

PORTB = graycode\_table[i];

if (i == 0) {

minAngle[i] = 0;

maxAngle[i] += anglePerDivision;

}

if(i >0 && i<64) {

minAngle[i] = maxAngle[i - 1];

maxAngle[i] += anglePerDivision;

}

if (result == graycode\_table[i]) {

printf("Range is %.3lf degree to %.3lf degree.", minAngle[i],

maxAngle[i]);

\_delay\_ms(DELAY);

break;

}

}

}

}

**2.Accurate Delay**

#include <avr/io.h>

#include <avr/interrupt.h>

#include <stdio.h>

#include <inttypes.h> // to print uint32\_t

#define SLOTS\_PER\_REV 20 /\*\*< Total slots in a encoder disc \*/

#define FOSC 16000000

#define BAUD 57600 /\*\*< Baud Rate in bps. refer page 179 of 328p datasheet. \*/

#define MYUBRR FOSC/16/BAUD-1

volatile uint32\_t revolution\_count; /\*\*< Counter for revolution \*/

volatile int slot\_count; /\*\*< Counter for slots \*/

volatile uint32\_t n; /\*\*< n to count number of times TCNT1 overflowed \*/

volatile uint32\_t elapse\_time=0;/\*\*< Stores cumulative revolution completion time \*/

volatile uint8\_t flag = 0, print\_flag=0;

volatile int count = 0;

void USART\_init(unsigned int ubrr){

UCSR0C = (0<<USBS0)|(3<<UCSZ00);

UCSR0A = 0b00000000;/// Step 2. Set UCSR0A in Normal speed, disable multi-proc

UBRR0H = (unsigned char)(ubrr>>8);/// Step 3. Load ubrr into UBRR0H and UBRR0L

UBRR0L = (unsigned char)ubrr;

UCSR0B = 0b00001000;/// Step 4. Enable Tx Rx and disable interrupt in UCSR0B

}

int USART\_send(char c, FILE \*stream){

while ( !( UCSR0A & (1<<UDRE0)) ){;}

UDR0 = c; /// Step 2. Write char to UDR0 for transmission

return 0;

}

int USART\_receive(FILE \*stream){

while ( !(UCSR0A & (1<<RXC0)) );

return UDR0;/// Step 2. Get and return received data from buffer

}

ISR(TIMER1\_OVF\_vect)

{

n++;

}

void timer1\_init()

{

TCCR1A = 0b00000000; ///Step 1. normal mode

TCCR1B = 0b00000011; ///Step 2. 1:64 prescaler, internal clock

TIMSK1 = 0b00000001; ///Step 3. enable Timer 1 overflow interrupt

}

ISR(INT0\_vect){

if (flag ==0) {

TCNT1 =0;

n = 0;

flag =1; /// Make the flag 1 so that it never enters again

}

if (slot\_count < SLOTS\_PER\_REV) {

slot\_count ++; /// increase counter

}

else {

count++;

revolution\_count ++; /// increase revolution counter

print\_flag=1; /// tell main loop that it may print new value

slot\_count =0; /// reset to start count for next revolution

elapse\_time += n \* 262144 + (uint32\_t) (TCNT1\*4);

TCNT1 =0; /// reset counter to calculate next revolution

n = 0;

}

}

void accurate\_blocking(){

//accurate blocking/without interrupt/using timer counter

//code will be written

}

void accurate\_non\_blocking(){

//accurate non-blocking/without interrupt/using timer counter

//code will be written

}

int microseconds(int value){

return value / 1000000;

}

int milliseconds(int value){

return value / 1000;

}

int seconds(int value){

return value;

}

int main()

{

//our group member's last 2 digit of ID number

int a = 30;

int b = 42;

int c = 62;

slot\_count =0;

revolution\_count = 0;

USART\_init(MYUBRR); /// Initialize USART

stdout =fdevopen(USART\_send,NULL); /// setup printf()

DDRD &= ~(1<<DDD2);/// Set int0 as input

EICRA |= (1<<ISC01);

EICRA |= (1<<ISC00);

EIMSK|= (1<<INT0);/// Enable INT0

timer1\_init(); /// Initiate Timer1

sei();/// Enable global interrupt

while(1){

if(print\_flag==1) {

print\_flag =0;

printf("Number of revolution is %"PRId32" time taken %"PRId32" uS \r\n", revolution\_count,elapse\_time);

}

};

printf("end\n\r");

}

**3.PWM Signal**

#include<avr/io.h>

#include<util/delay.h>

#define F\_CPU 16000000UL

uint16\_t A[] = { 0, 300, 600, 900, 1200, 1500, 1800, 2100, 2400, 6300, 6600,

6900, 7200, 7500, 7800, 8100, 8400, 8700, 9000, 9300, 9600, 2700, 3000,

3300, 3600, 3900, 4200, 4500, 4800, 5100, 5400, 5700, 6000, 9900, 10200,

49035, 48785, 48535, 48285, 48035, 47785, 47535, 47285, 47035, 46785,

46535, 46285, 46035, 45785, 45535, 45285, 45035, 44785, 44535, 44285,

13500, 13800, 14100, 14400, 14700, 15000, 15300, 15600, 15900, 16200,

16500, 16800, 17100, 17400, 17700, 18000, 18300, 18600, 18900, 19200,

19500, 19800, 20100, 20400, 20700, 21000, 21300, 21600, 65535, 16987,

17472, 17957, 18442, 18927, 19412, 19897, 20382, 20867, 21352, 21837,

22322, 22807, 23292, 23777, 24262, 24747, 25232, 25717, 26202, 26687,

52428, 51804, 51180, 50556, 49932, 49308, 48684, 48060, 12900, 19909,

19424, 18939, 18454, 17969, 17484, 16999, 16514, 16029, 15544, 52535,

52285, 52035, 51785, 51535, 51285, 51035, 50785, 50535, 50285, 50035,

49785, 49535, 49285, 65285, 65035, 64785, 64535, 64285, 64035, 63785,

63535, 63285, 63035, 62785, 62535, 62285, 62035, 61785, 61535, 61285,

61035, 60785, 15300, 15000, 14700, 14400, 14100, 13800, 13500, 13200,

12900, 12600, 60535, 60285, 60035, 59785, 59535, 59285, 59035, 58785,

58535, 58285, 58035, 57785, 57535, 57285, 57035, 56785, 56535, 56285,

56035, 55785, 55535, 55285, 55035, 54785, 54535, 54285, 54035, 53785,

53535, 53285, 53035, 52785, 41820, 43692, 45564, 47436, 49308, 51180,

53052, 26214, 25729, 25244, 24759, 24274, 23789, 23304, 22819, 22334,

21849, 21364, 20879, 203 };

uint16\_t B[] = { 0, 13107, 13592, 14077, 14562, 15047, 15532, 16017, 16502,

16987, 17472, 17957, 18442, 18927, 19412, 19897, 20382, 20867, 21352,

21837, 22322, 22807, 23292, 23777, 24262, 24747, 25232, 25717, 26202,

26687, 52428, 51804, 51180, 50556, 49932, 49308, 48684, 48060, 12900,

13200, 13500, 13800, 14100, 14400, 14700, 15000, 15300, 15600, 47436,

46812, 46188, 45564, 44940, 44316, 43692, 43068, 42444, 41820, 18300,

18000, 17700, 17400, 17100, 16800, 16500, 16200, 15900, 15600, 12900,

13200, 13500, 13800, 14100, 14400, 14700, 15000, 15300, 15600, 15900,

16200, 16500, 16800, 17100, 17400, 17700, 18000, 18300, 18600, 18900,

19200, 19500, 19800, 20100, 20400, 20700, 21000, 21300, 21600, 65535,

65285, 65035, 64785, 64535, 64285, 64035, 63785, 63535, 63285, 63035,

62785, 62535, 62285, 62035, 61785, 61535, 61285, 61035, 60785, 15300,

15000, 14700, 14400, 14100, 13800, 13500, 13200, 12900, 12600, 41196,

40572, 39948, 39324, 38700, 38076, 37452, 36828, 36204, 35580, 34956,

34332, 33708, 33084, 32460, 31836, 31212, 30588, 29964, 29340, 28716,

28092, 27468, 26844, 28716, 30588, 32460, 34332, 36204, 38076, 39948,

41820, 43692, 45564, 47436, 49308, 51180, 53052, 26214, 25729, 25244,

24759, 24274, 23789, 23304, 22819, 22334, 21849, 21364, 20879, 20394,

19909, 19424, 18939, 18454, 17969, 17484, 16999, 16514, 16029, 15544,

15059, 14574, 14089, 13604, 13119 };

uint16\_t E[] = { 0, 13107, 13592, 14077, 14562, 15047, 15532, 16017, 25244,

24759, 24274, 23789, 23304, 22819, 22334, 21849, 21364, 20879, 26202,

26687, 52428, 51804, 51180, 50556, 49932, 49308, 48684, 48060, 47436,

46812, 46188, 45564, 44940, 44316, 43692, 43068, 42444, 41820, 21352,

21837, 22322, 22807, 23292, 23777, 24262, 24747, 25232, 25717, 41196,

10500, 10800, 11100, 11400, 11700, 12000, 12300, 12600, 15900, 16200,

16500, 16800, 17100, 17400, 17700, 18000, 18300, 18600, 12900, 13200,

13500, 13800, 14100, 14400, 14700, 15000, 15300, 15600, 18900, 19200,

19500, 19800, 20100, 20400, 20700, 21000, 21300, 21600, 21300, 21000,

20700, 20400, 20100, 19800, 19500, 19200, 18900, 18600, 18300, 18000,

17700, 17400, 17100, 16800, 16500, 16200, 15900, 15600, 12900, 13200,

40572, 39948, 39324, 38700, 38076, 37452, 36828, 36204, 35580, 28716,

28092, 27468, 26844, 28716, 30588, 32460, 34332, 36204, 38076, 39948,

41820, 43692, 45564, 47436, 49308, 51180, 53052, 26214, 25729, 16502,

16987, 17472, 17957, 18442, 18927, 19412, 19897, 20382, 20867, 20394,

62785, 62535, 62285, 62035, 61785, 61535, 61285, 61035, 60785, 15300,

15000, 14700, 14400, 14100, 13800, 13500, 13200, 12900, 12600, 41196,

40572, 39948, 39324, 38700, 38076, 37452, 36828, 36204, 35580, 34956,

34332, 33708, 33084, 32460, 31836, 31212, 30588, 29964, 29340, 28716,

28092, 27468, 26844, 28716, 30588, 32460, 34332, 36204, 38076, 39948,

41820, 43692, 45564, 47436, 49308, 51180, 53052, 26214, 25729, 25244,

24759, 24274, 23789, 23304, 22819, 22334, 21849, 21364, 20879, 20394,

19909, 19424, 18939, 18454, 17969, 17484, 16999, 16514, 16029, 34956,

34332, 33708, 33084, 32460, 31836, 31212, 30588, 29964, 29340, 15544,

15059, 14574, 14089, 13604, 13119 };

void generate\_waves() {

uint8\_t i = 0, j = 0;

while (1) {

for (int i = 0; i < 512; i++) {

OCR1A = A[i];

OCR1B = B[i];

OCR2A = E[i];

\_delay\_ms(2);

}

}

}

int main(void) {

DDRB |= (1 << PB3) | (1 << PB2) | (1 << PB1);

OCR1A = 0; // initializing OCR1A

OCR1B = 0; // initializing OCR1B

OCR2A = 0;

TCCR1B |= (1 << WGM13) | (1 << WGM12); //

TCCR1A |= (1 << COM1A1) | (0 << COM1A0) | (1 << COM1B1) | (0 << COM1B0)

| (1 << WGM11) | (0 << WGM10);

TCCR1B |= (0 << CS12) | (0 << CS11) | (1 << CS10);

ICR1 = 65535;

generate\_waves();

}

**4.On Board Temperature Sensor**

#include <avr/io.h>

#include <util/delay.h>

#include <stdio.h>

#define F\_CPU 16000000UL

#define FOSC 16000000

#define BAUD 9600 /\*\*< Baud Rate in bps. refer page 179 of 328p datasheet. \*/

#define MYUBRR FOSC/16/BAUD-1

int result;

void USART\_init(unsigned int ubrr) {

UCSR0C = (0 << USBS0) | (3 << UCSZ00);

UCSR0A = 0b00000000; /// Step 2. Set UCSR0A in Normal speed, disable multi-proc

UBRR0H = (unsigned char) (ubrr >> 8); /// Step 3. Load ubrr into UBRR0H and UBRR0L

UBRR0L = (unsigned char) ubrr;

UCSR0B = 0b00011000; /// Step 4. Enable Tx Rx and disable interrupt in UCSR0B

}

int USART\_send(char c, FILE \*stream) {

while (!( UCSR0A & (1 << UDRE0))) { ; }

UDR0 = c; /// Step 2. Write char to UDR0 for transmission

}

int USART\_receive(FILE \*stream) {

while (!(UCSR0A & (1 << RXC0)));

return UDR0; /// Step 2. Get and return received data from buffer

}

void init\_ADC() {

ADMUX = 0b11001000;

ADCSRA = 0b10000110;

}

int main() {

init\_ADC();

USART\_init(MYUBRR);

stdout = fdevopen(USART\_send, NULL);

stdin = fdevopen(NULL, USART\_receive);

while (1) {

ADCSRA |= (1 << ADSC);

while (bit\_is\_set(ADCSRA, ADSC)) {

;

}

result = (ADC - 314) / 1.22;

printf("{\"adc0%d\":%d}\n", bit\_is\_set(ADMUX, 3), result);

\_delay\_ms(100); } }

**5.ADC Trigger Source and Interrupt**

include <avr/io.h>

#include <avr/interrupt.h>

#include <stdio.h>

#include <inttypes.h> // to print uint32\_t

#define SLOTS\_PER\_REV 20 /\*\*< Total slots in a encoder disc \*/

#define FOSC 16000000

#define BAUD 57600 /\*\*< Baud Rate in bps. refer page 179 of 328p datasheet. \*/

#define MYUBRR FOSC/16/BAUD-1

int result;

volatile uint32\_t revolution\_count; /\*\*< Counter for revolution \*/

volatile int slot\_count; /\*\*< Counter for slots \*/

volatile uint32\_t n; /\*\*< n to count number of times TCNT1 overflowed \*/

volatile uint32\_t elapse\_time = 0;/\*\*< Stores cumulative revolution completion time \*/

volatile uint8\_t flag = 0, print\_flag = 0;

void USART\_init(unsigned int ubrr) {

UCSR0C = (0 << USBS0) | (3 << UCSZ00);

UCSR0A = 0b00000000; /// Step 2. Set UCSR0A in Normal speed, disable multi-proc

UBRR0H = (unsigned char) (ubrr >> 8); /// Step 3. Load ubrr into UBRR0H and UBRR0L

UBRR0L = (unsigned char) ubrr;

UCSR0B = 0b00001000; /// Step 4. Enable Tx Rx and disable interrupt in UCSR0B

}

int USART\_send(char c, FILE \*stream) {

while (!( UCSR0A & (1 << UDRE0))) { ; }

UDR0 = c; /// Step 2. Write char to UDR0 for transmission

return 0;

}

int USART\_receive(FILE \*stream) {

while (!(UCSR0A & (1 << RXC0)));

return UDR0; /// Step 2. Get and return received data from buffer

}

ISR(TIMER1\_OVF\_vect) {

n++;

}

void timer1\_init() {

TCCR1A = 0b00000000; ///Step 1. normal mode

TCCR1B = 0b00000011; ///Step 2. 1:64 prescaler, internal clock

TIMSK1 = 0b00000001; ///Step 3. enable Timer 1 overflow interrupt

}

ISR(INT0\_vect) {

if (flag == 0) {

TCNT1 = 0;

n = 0;

flag = 1; /// Make the flag 1 so that it never enters again

}

if (slot\_count < SLOTS\_PER\_REV) /// smaller than total slots

{

slot\_count++; /// increase counter

} else /// Completed one revolution

{

revolution\_count++; /// increase revolution counter

print\_flag = 1; /// tell main loop that it may print new value

slot\_count = 0; /// reset to start count for next revolution

elapse\_time += n \* 262144 + (uint32\_t) (TCNT1 \* 4);

TCNT1 = 0; /// reset counter to calculate next revolution

n = 0;

}

}

void init\_ADC() {

ADMUX = 0b01000000;

ADCSRA = 0b10100111;

ADCSRB = 0b00000110;

DIDR0 = (1 << ADC5D) | (1 << ADC4D) | (1 << ADC3D) | (1 << ADC2D)

| (1 << ADC1D) | (0 << ADC0D);

}

uint16\_t read\_ADC(uint8\_t ch) {

// select the corresponding channel 0~7

// ANDing with ’7′ will always keep the value

// of ‘ch’ between 0 and 7

ch &= 0b00000111; // AND operation with 7

ADMUX = (ADMUX & 0xF8) | ch; // clears the bottom 3 bits before ORing

ADCSRA |= (1 << ADSC);

while (ADCSRA & (1 << ADSC)) ;

return (ADC);

}

void trigger\_ADC(uint8\_t channel) {

ADMUX &= 0xF0;

ADMUX |= (channel & 0x0F);

ADCSRA |= (1 << ADSC);

}

void adc\_interrupt(){

//here adc interrupt code will be written

}

int main() {

slot\_count = 0;

revolution\_count = 0;

USART\_init(MYUBRR); /// Initialize USART

stdout = fdevopen(USART\_send, NULL); /// setup printf()

DDRD &= ~(1 << DDD2); /// Set int0 as input

EICRA |= (1 << ISC01);

EICRA |= (1 << ISC00);

EIMSK |= (1 << INT0); /// Enable INT0

timer1\_init(); /// Initiate Timer1

sei();

while (1) {

result = read\_ADC(0b01000000);

printf("{\"adc0%d\":%d}\n", bit\_is\_set(ADMUX, 0), result);

trigger\_ADC(result);

if (print\_flag == 1) {

print\_flag = 0;

printf("Number of revolution is %"PRId32" time taken %"PRId32" uS \r\n",

revolution\_count, elapse\_time);

}

};

printf("end\n\r");

}