/\*

\* stepper\_motor\_lab.c

\*

\* Created: 10/24/18 7:13:18 PM

\* Author : juanruelas

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

unsigned char btn;

/\* USED FOR CREATING A SYNCRONOUS STATE MACHINE\*/

volatile unsigned char TimerFlag = 0; // TimerISR() sets this to 1. C programmer should clear to 0.

// Internal variables for mapping AVR's ISR to our cleaner TimerISR model.

unsigned long \_avr\_timer\_M = 1; // Start count from here, down to 0. Default 1 ms.

unsigned long \_avr\_timer\_cntcurr = 0; // Current internal count of 1ms ticks

void TimerOn() {

// AVR timer/counter controller register TCCR1

TCCR1B = 0x0B;// bit3 = 0: CTC mode (clear timer on compare)

// bit2bit1bit0=011: pre-scaler /64

// 00001011: 0x0B

// SO, 8 MHz clock or 8,000,000 /64 = 125,000 ticks/s

// Thus, TCNT1 register will count at 125,000 ticks/s

// AVR output compare register OCR1A.

OCR1A = 125; // Timer interrupt will be generated when TCNT1==OCR1A

// We want a 1 ms tick. 0.001 s \* 125,000 ticks/s = 125

// So when TCNT1 register equals 125,

// 1 ms has passed. Thus, we compare to 125.

// AVR timer interrupt mask register

TIMSK1 = 0x02; // bit1: OCIE1A -- enables compare match interrupt

//Initialize avr counter

TCNT1=0;

\_avr\_timer\_cntcurr = \_avr\_timer\_M;

// TimerISR will be called every \_avr\_timer\_cntcurr milliseconds

//Enable global interrupts

SREG |= 0x80; // 0x80: 1000000

}

void TimerOff() {

TCCR1B = 0x00; // bit3bit1bit0=000: timer off

}

void TimerISR() {

TimerFlag = 1;

}

// In our approach, the C programmer does not touch this ISR, but rather TimerISR()

ISR(TIMER1\_COMPA\_vect) {

// CPU automatically calls when TCNT1 == OCR1 (every 1 ms per TimerOn settings)

\_avr\_timer\_cntcurr--; // Count down to 0 rather than up to TOP

if (\_avr\_timer\_cntcurr == 0) { // results in a more efficient compare

TimerISR(); // Call the ISR that the user uses

\_avr\_timer\_cntcurr = \_avr\_timer\_M;

}

}

// Set TimerISR() to tick every M ms

void TimerSet(unsigned long M) {

\_avr\_timer\_M = M/10;

\_avr\_timer\_cntcurr = \_avr\_timer\_M;

}

enum CLOCKWISE { A, AB, B, BC, C, CD, D, DA } CLOCKWISE;

enum COUNTER\_CLOCKWISE { D1, DC, C1, CB, B1, BA, A1, AD } COUNTER\_CLOCKWISE;

// state machine for going clockwise

void tick\_clkwise(){

// transitions

switch(CLOCKWISE){

case A:

CLOCKWISE = AB; break;

case AB:

CLOCKWISE = B; break;

case B:

CLOCKWISE = BC; break;

case BC:

CLOCKWISE = C; break;

case C:

CLOCKWISE = CD; break;

case CD:

CLOCKWISE = D; break;

case D:

CLOCKWISE = DA; break;

case DA:

CLOCKWISE = A; break;

}

// actions

switch(CLOCKWISE){

case A:

PORTA = 0b00000001; break;

case AB:

PORTA = 0b00000011; break;

case B:

PORTA = 0b00000010; break;

case BC:

PORTA = 0b00000110; break;

case C:

PORTA = 0b00000100; break;

case CD:

PORTA = 0b00001100; break;

case D:

PORTA = 0b00001000; break;

case DA:

PORTA = 0b00001001; break;

}

}

// state machine for going counter clockwise

void tick\_cnt\_clkwise(){

// transitions

switch(COUNTER\_CLOCKWISE){

case A1:

COUNTER\_CLOCKWISE = DA; break;

case AB:

COUNTER\_CLOCKWISE = A1; break;

case B1:

COUNTER\_CLOCKWISE = AB; break;

case BC:

COUNTER\_CLOCKWISE = B1; break;

case C1:

COUNTER\_CLOCKWISE = BC; break;

case CD:

COUNTER\_CLOCKWISE = C1; break;

case D1:

COUNTER\_CLOCKWISE = CD; break;

case DA:

COUNTER\_CLOCKWISE = D1; break;

}

// actions

switch(COUNTER\_CLOCKWISE){

case A1:

PORTA = 0b00000001; break;

case AB:

PORTA = 0b00000011; break;

case B1:

PORTA = 0b00000010; break;

case BC:

PORTA = 0b00000110; break;

case C1:

PORTA = 0b00000100; break;

case CD:

PORTA = 0b00001100; break;

case D1:

PORTA = 0b00001000; break;

case DA:

PORTA = 0b00001001; break;

}

}

int main(void) {

DDRB = 0x00; PORTB = 0xFF; // Configure port B's 8 pins as inputs

DDRA = 0xFF; PORTA = 0x00; // Configure port A's 8 pins at outputs

TimerSet(30);

TimerOn();

//unsigned int t;

while (1) {

while (!TimerFlag);

TimerFlag = 0;

//tick\_clkwise();

btn = PINB;

if((btn & 0x01) == 0x01){

tick\_clkwise();

}

else if((btn & 0x02) == 0x02){

tick\_cnt\_clkwise();

}

}

}