```
// stepper motor driver for spectrometer
// with manual speed and direction input
// and adjustable scan speed and direction
// stepper motor driver set to 32usteps/step (32Hz clk = 1 step/sec)

// filename pwm.c
// ATtiny84 (14 pin DIP)

// changes made to code
// change scan/man polarity 9/27/16 (now man = 1, scan = 0)
// done so that when remote is  disconnected internal pullup sets to manual
// also added 10M pull up/down on manual whiper to whiper at midpoint when
// remote not connected (before it would start stepping if remote disconnected)

#define F_CPU 1000000L            // use 1MHz sys clk

#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include <avr/pgmspace.h>// allow array data to be stored in program space


// value to load into ADMUX to select A/D channels 0-2
const char chan_joy = 0;  // joytick for man speed ctrl, springloaded to center
const char chan_scan = 7;// use to set scan speed when not in manual mode

// deadband for joystick movement (i.e. no action near middle or ends, loose ~10% of travel)
const char mid_dead = 15;// abs center 0-15 (of +/- 511 not used pot center not accurate
const char end_dead = 119;        // end at 119 of 127 possible (pot doesn't go to ends)
                                  // still want max speed

const unsigned int duty  = 5;     // PWM high for 10us when turned on (1MHz sys clock)
const unsigned int mid_ref = 512;// default joystick center position
```

```
// lookup table for PWM register 0-119 (120 points with last two max speed)
// max period for 10us dead time between pulses is 65530 (2^16-1-5), 2X5 = 10us
// min period for 50KHz pwm is 10, 2X10 = 20us
// 10bit A/D, +/-512 from midpoint, remove 15 from middle (deadband), shift 2 (div by 4)
// and remove few more bits (joystick travel doesn't go to ends)
// want even logarithmic steps between min & max speed (same % speed increase each step)
// 65530 = 10*X^n, n = 0-118
// ln(6553) = 118 ln X, X = 0.074471849, e^X = 1.077315016, change of 7.7% between points
const uint16_t look_tbl[] PROGMEM = {65530, 60827, 56462, 52410, 48648, 45157, 41916, 38908,
36116, 33524, 31118, 28885, 26812, 24888, 23102, 21444, 19905, 18476, 17150, 15919, 14777, 13716,
12732, 11818, 10970, 10183, 9452, 8774, 8144, 7560, 7017, 6514, 6046, 5612, 5209, 4836, 4489,
4166, 3867, 3590, 3332, 3093, 2871, 2665, 2474, 2296, 2131, 1978, 1836, 1705, 1582, 1469, 1363,
1266, 1175, 1090, 1012, 940, 872, 810, 751, 697, 647, 601, 558, 518, 481, 446, 414, 384, 357,
331, 307, 285, 265, 246, 228, 212, 197, 183, 169, 157, 146, 136, 126, 117, 108, 101, 93, 87, 80,
75, 69, 64, 60, 55, 51, 48, 44, 41, 38, 35, 33, 31, 28, 26, 24, 23, 21, 20, 18, 17, 16, 15, 13,
13, 12, 11, 10, 10};


// I/O pin map
// PA0 pin13 (ADC0) joystick whiper input (manual scan speed input from remote)
// PA7 pin 6 (ADC7) scan_spd whiper input (sets scan speed when not in manual mode, from remote)

// PB0 pin 2 fast/slow input for manual scan (0 = fast, 1 = slow)
// PB1 pin 3 scan/man input from remote (manual joystick = 0, scanning = 1)
// PB2 pin 5 scan direction input (jumper in remote battery door, 0 = up, 1 = down)

// PA3 pin 10 direction out (dir on motor driver)
// PA4 pin 9 out to SLEEP NOT on motor driver (shared with SCLK)
// PA5 pin 8 (OC1B) PWM out to motor driver (shared with MISO)

// PA1 pin 12 Input (Back (count down) limit switch (active high))
// PA2 pin 11 Input (Forward (count up) limit switch (active high))

// programming pins
// PB3 pin 4 RST NOT
// PA6 pin 7 MOSI
// PA5 pin 8 MISO (shared with PWM out to step)
// PA4 pin 9 SCLK (shared with SLEEP NOT out to driver)


// function to get data from lookup table
inline uint16_t getEntry(int index) {
  return (uint16_t)pgm_read_dword_near(look_tbl + index);
}


int main(void)
{
  unsigned int joystick; // A/D reading of speed joystick
  unsigned int scan_spd; // A/D reading of scan speed pot
  unsigned int temp;              // temp var
  int pwm = 0;                    // var used to call PWM lookup table
  char dir = 0;                   // step direction in manual mode (0 = down, 1 = up)
  char dir_scan = 1;              // step direction in scan mode (0 = down, 1 = up)
  int slp_cnt = 0;                // turn off driver after about one second with no movement
  int slp_scan = 0;               // monitor if first time through scan loop

  // set I/O pin direction and pullups
  // pull up active if DDR bit set for input and PORT bit is high
  // DDR bit = 1 for output & 0 for input
  DDRA = 0x78;           // PA0-PA2 & PA7  inputs, PA3-PA6 outputs
//  PORTA = 0x16;        // pullups on PA1-PA2, if PA4 low (SLEEP NOT = 1, enable driver)
  PORTA = 0x06;          // pullups on PA1-PA2, PA4 = SLEEP NOT = 0, disable driver)
  DDRB = 0x08;           // PB0-PB2 inputs, PB3 output (unused)
  PORTB = 0x07;          // enable pull-ups on inputs

  // ADC setup
  DIDR0 = 0x81;          // disable digital inputs on ADC0 & ADC7
  ADMUX = 0x00;          // Vref for A/D Vcc to GND, (REFS[1:0] = 00 (ADMUX[7:6])
                         // single ended inputs, MUX[5:3] = 000 (ADMUX[5:3])
                         // ADC0 selected on MUX, MUX[2:0] = 000 (ADMUX[2:0])
  ADCSRA = 0x83; // power up ADC (ADEN = 1, ADCSRA[7])
```

```
                         // set ADC prescaler div by 8 (ADPS[2:0] = 011, ADCSRA[2:0])
                         // ADCSRB = 0x00 (default) unipolar, left adj output

  // PWM setup (phase correct mode) WGM[3:0]=1011
  // freq = clock div by 2*OCR1A (count), sys clcok = 1Mhz
  TCCR1A = 0b00100011;   // OC1A disconnected from output (COM1A[1:0]=00)
                         // COM1B non invert output (COM1B[1:0]=01)
                         // WGM1[1:0]=11
//  TCCR1B = 0b00010010;// use sys clk, prescaler div by 8 (CS[2:0]=010) (slow)
  TCCR1B = 0b00010001;   // use sys clk, prescaler div by 1 (CS[2:0]=001) (fast)
                         // WGM1[3:2]=10
  OCR1A = 0;             // turn off pwm (output low)
  OCR1B = duty;          // load PWM dutycycle reg (never changed)

//  scan_spd = 0;
//  pwm = 0;


  while(1){                      // loop forever

   // in scan mode
   if ((PINB & (1<<PB1)) == 0){   // if scan/man low (scan)
    if (slp_scan == 0){           // if first time through loop
      TCCR1B = 0b00010010;        // use sys clk, prescaler div by 8 (CS[2:0]=010) (slow)
      if (PINB & (1<<PB2)){       // read scan direction jumper
        PORTA |= (1<<PA3);        // PA3 = 1 (rot dir high, count up)
        dir_scan = 1;             // scan direction set for scanning up
      }else{
        PORTA &= ~(1<<PA3);       // PA3 = 0 (rot dir low, count down)
        dir_scan = 0;             // scan direction set for scanning down
      }

      ADMUX = chan_scan;        // A/D MUX set for scan speed pot (chan 7)
      ADCSRA |= (1<<ADSC);        // start A/D conversion (ADSC=1)
      while (ADCSRA & (1<<ADSC));// wait for conversion to complete (ADSC=0)
      scan_spd = ADCW;            // read sacn speed pot (unsigned 16 bit int, 10 bit A/D)
      pwm = scan_spd >> 3;        // div by 8
      if (pwm > end_dead) // stay in array index bounds
        pwm = end_dead;           // all values beyond this are max array index
      PORTA |= (1<<PA4);  // PA4 = 1 (wake up motor driver, i.e. hold & stepping)

      // limit switch check
      if (dir_scan == 1){         // if scan direction is up
        if ((PINA & (1<<PA1)) == 0x00)     // and upper limit switch OK (LOW)
          OCR1A = getEntry(pwm);  // get pwm value from lookup table
        else{                              // if upper limit shitch tripped (HIGH)
          OCR1A = 0;                       // turn pwm off
          PORTA &= ~(1<<PA4);              // PA4 = 0 (put motor driver to sleep, i.e. no current)
        }
      }else{                               // if scan direction is down
        if ((PINA & (1<<PA2)) == 0x00)     // and lower limit switch OK (LOW)
          OCR1A = getEntry(pwm);  // get pwm value from lookup table
        else{                              // if lower limit shitch tripped (HIGH)
          OCR1A = 0;                       // turn pwm off
          PORTA &= ~(1<<PA4);              // PA4 = 0 (put motor driver to sleep, i.e. no current)
        }
      }

      slp_scan = 1;                        // mark as not first time through loop
    }

    // limit switch check
    if (dir_scan == 1)                     // if scan direction is up
      if ((PINA & (1<<PA2)) == 0x00)       // and upper limit switch OK (LOW)
        slp_scan = 1;                      // do nothing instruction (slp_scan already = 1)
      else{
        OCR1A = 0;                         // turn pwm off
        PORTA &= ~(1<<PA4);                // PA4 = 0 (put motor driver to sleep, i.e. no current)
      }
    else                                   // if scan direction is down
      if ((PINA & (1<<PA1)) == 0x00)       // and lower limit switch OK (LOW)
        slp_scan = 1;                      // do nothing instruction (slp_scan already = 1)
```

```
   else{
     OCR1A = 0;                          // turn pwm off
      PORTA &= ~(1<<PA4);                // PA4 = 0 (put motor driver to sleep, i.e. no current)
   }

 }
 else{

  // in manual mode
   slp_scan = 0;

  // read joystick position
  ADMUX = chan_joy;            // A/D MUX set for joystick (chan 0)
  ADCSRA |= (1<<ADSC);  // start A/D conversion (ADSC=1)
  while (ADCSRA & (1<<ADSC));   // wait for conversion to complete (ADSC=0)
  joystick = ADCW;             // read joystick (unsigned 16 bit int, 10 bit A/D)

  if (joystick > mid_ref){      // if joystick moved forward
    temp = joystick - mid_ref;// calc difference (temp positive)
    dir = 1;                   // count up
  }else{                       // if joystick moved backwards
    temp = mid_ref - joystick;// calc difference (temp positive)
    dir = 0;                   // count down
  }

  // check slow/fast switch position
  if (PINB & (1<<PB0))  // If high (slow)
    TCCR1B = 0b00010010;       // use sys clk, prescaler div by 8 (CS[2:0]=010) (slow)
  else                  // if low (fast)
    TCCR1B = 0b00010001;       // use sys clk, prescaler div by 1 (CS[2:0]=001) (fast)

  // if joystick position out of deadband take steps at appropriate rate
  if (temp > mid_dead){        // if joystick moved past center dead band
    slp_cnt = 0;                       // zero sleep counter
    PORTA |= (1<<PA4);         // PA4 = 1 (wake up motor driver, i.e. hold & stepping)
    if (dir == 1)                      // if joystick to the right (count up)
```

```c
        PORTA |= (1<<PA3);                // PA3 = 1 (rot dir pin high)
      else                                // if joystick to the left (count down)
        PORTA &= ~(1<<PA3);               // PA3 = 0 (rot dir pin low)

      pwm = (temp - mid_dead) >> 2;       // remove midband and div by 4
      if (pwm > end_dead)         // stay in array index bounds
        pwm = end_dead;                   // all values beyond this are max array index

      // limit switch check
      if (dir == 1){                      // if counting up (dir = 1)
        if ((PINA & (1<<PA2)) == 0x00)    // and upper limit switch OK (LOW)
          OCR1A = getEntry(pwm);  // get pwm value from lookup table
        else{                             // if upper limit shitch tripped (HIGH)
          OCR1A = 0;                      // turn pwm off
          PORTA &= ~(1<<PA4);             // PA4 = 0 (put motor driver to sleep, i.e. no current)
        }
      }else{                              // if counting down (dir = 0)
        if ((PINA & (1<<PA1)) == 0x00)    // and lower limit switch OK (LOW)
          OCR1A = getEntry(pwm);  // get pwm value from lookup table
        else{                             // if lower limit shitch tripped (HIGH)
          OCR1A = 0;                      // turn pwm off
          PORTA &= ~(1<<PA4);             // PA4 = 0 (put motor driver to sleep, i.e. no current)
        }
      }

    }else{
      OCR1A = 0;                  // when joystick in middle turn off pwm (output low)
      if (slp_cnt < 10000)                // don't cut power to driver right away
        ++slp_cnt;
      else
        PORTA &= ~(1<<PA4);               // PA4 = 0 (put motor driver to sleep, i.e. no current)
    }
   }
  }

return 0;
```

```
    }
```