Electronic Circuit Applications
Final Project
Stepper Motor Control

Andrew Grasberger

December 16, 2012

Contents

Abstract 2
Introduction 2
Background/Theory 2
Design 3
Results 6
Conclusion 6
Appendix 7

Abstract

A MATLAB interface was created to take a velocity profile and output it to a stepper motor. MATLAB takes a velocity profile and sends it to a μ controller through a UART serial interface. The μ controller interprets the data and controls H-Bridges to control the coils of a stepper motor.

Introduction

Stepper motors have an advantage over regular motors. They have the ability to turn in discrete amounts (2° or less) which makes them useful where precision is necessary. When the drive shaft is coupled with a lead screw, these motors can provide linear movement very precisely. These motors have become widely used in home-brew CNC's and 3D printer's. The downside to using these motors is that the way they are configured requires specialized circuitry to get them to function, which can be expensive. This project was undertaken in order to obtain a better personal understanding of how the control circuitry works, allowing for future work with these technologies.

Background/Theory

The motor used for this project was a bipolar, two coil stepper motor. In this configuration, there is a magnet in the center with a certain polarity and then there are lots of coils around the outside connected in two sets. The coils on the outside can have current flowing through them in either direction, producing different polarities of magnetism. So, by having current flowing in either direction, the permanent magnet in the center can be attracted or repelled

For standard motors, there is one coil that can have current running through it in either direction, and for opposite directions of current flow, the motor turns opposite directions. The required circuitry for running a standard motor is an H-bridge. Two H-bridges can be seen in Fig. 1. In this configuration, the control lines 1a and 1b must inverses of each other. If 1a is a logic 1 and 1b is a logic 0 then the current flows from Vcc to the pMOS resistor on right side of the coil, through the coil and then through the nMOS transistor on the bottom left of the coil to ground. Switching the values of 1a and 1b lets current flow the other way. For the stepper motors, there is an H-bridge for each coil so the current can flow either direction, producing the alternate polarities necessary for stepper motor operation. This requires both of the H-bridges pictured in Fig. 1.

The H-bridges are based on CMOS technology (i.e. nMOS and pMOS transistors). To get current to flow through an nMOS transistor, the gate voltage must be above the threshold gate voltage (about 5V). PMOS transistors are slightly trickier because the voltage at the gate must be at least the voltage at the source in order to turn off, or else current will leak through.

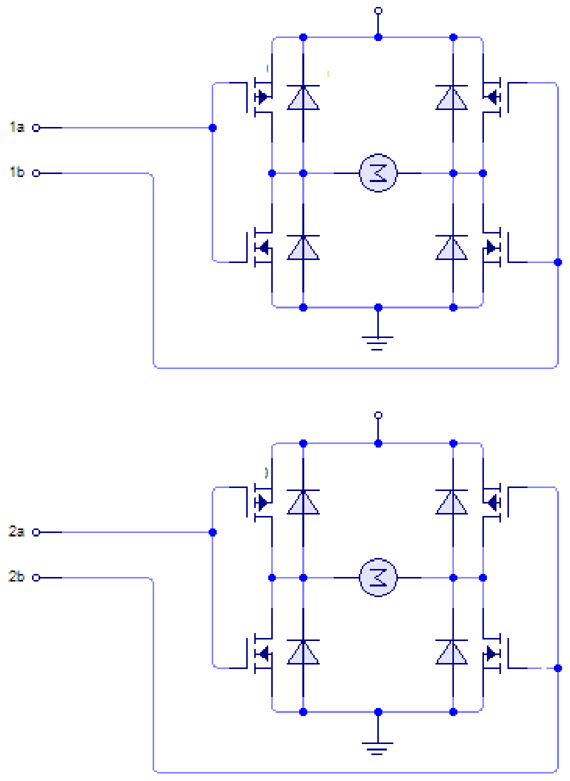

Figure 1: H-Bridges, one for Each Coil

Design

The overall implementation of the design was a MATLAB command program that would take velocity profiles and send them over a serial line to the MSP430G2553 (MSP430). The MSP430 processes the commands from MATLAB and sent them through MOSFET drivers to H-bridges which controls the coils of the stepper motor.

The stepper motors need power to operate, and the power that they run off of is much higher than the logic-level power that comes from the μcontroller

In order to get the logic level voltage from the MSP430 (3.3V) to the voltage that the motors run on (12V, which is also the required gate voltage for the pMOS transistors) buffers were used. Using a buffer along with pull-up resistors to Vcc allows there to be a power-level signal that corresponds in state to the logic level signal. Buffers either output a high voltage (5V) which would allow the node to be pulled up to 12V by the resistor, or they can be at ground, so the node is going to be held at 0.

The buffers used to get the logic level signal to a signal that can be used on the gates of the transistors are inverting buffers (SN7406) made by TI. There are non-inverting buffers as well (SN7407) but since the signal at 1a and 1b need to be inverse of each other, the inverting buffers were easier to implement. These buffers typically operate at a voltage of 5V but the threshold voltage for a logic 1 on an input is 2V so the 3.3V output from the

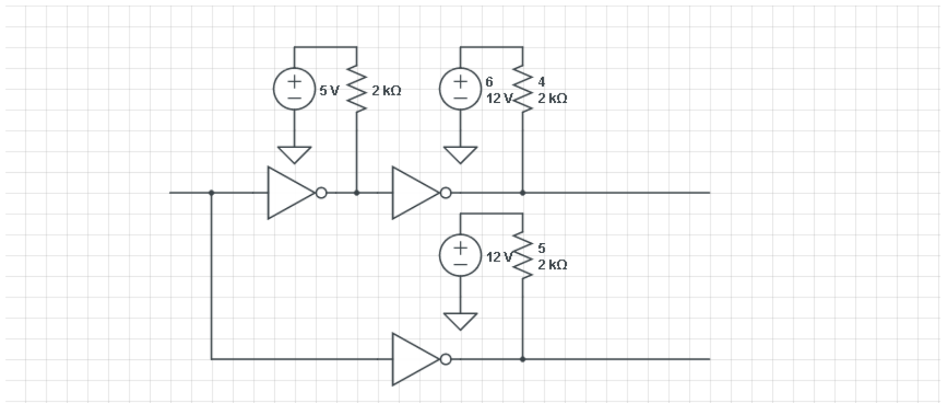

Figure 2: MOSFET Drivers from Hex Inverting Buffers

MSP430 is enough to register as a logic 1.

In Fig. 2 the line on the left is the input line from the MSP430. This signal goes to two buffers. Assuming the signal is a logic 1, both of the next buffers' outputs go to ground. The second buffer on the top branch inverts the signal once again, outputting a 1 which is drawn high by the pull up resistor attached to the 12V source. This gives opposite signals on the two outputs on the right side of the figure, which connect to the H-bridge in Fig. 1. Each H-bridge has one of the buffer circuits (like in Fig. 2) to control the transistors and the status of the coil. To get the motor to turn clockwise, the control lines for the buffer circuits have to follow the logic table in Table 1. The motors can also be run backwards by implementing these steps in reverse order. In the MSP430 code in the Appendix, there are functions that put these logic values on the pins to go forwards or backwards.

Line 1	Line 2
1	1
0	1
0	0
1	0

Table 1: Turning Motor Clockwise

The software side of the project involved taking a velocity function in MATLAB and mapping it to the appropriate signals on the output pins of the MSP430. Getting a velocity profile to go to the motor required a slightly hacky process on the MATLAB and MSP430 sides of the code. To send the velocity information over the serial lines, it was sent as an ASCII 'char' value. This meant that there was a limited range of discrete values that were valid. On the MATLAB side, that means that a range had to be defined where the input velocity function would be valid. The way the code is set up, there are hard limits on the velocity at ± 10. These values correspond to motor operating at its maximum speed in the forward or reverse directions. The velocity range in between -10 and 10 is converted to the 'data' range of 25 to 125 and rounds to the nearest integer, because there are no fractional ASCII values. This is a problem because it means that there is a limited resolution for the motor velocity. An extreme version of this is seen in Fig. 3 where there are only 10 velocity values for a cos curve. Fig. 4 shows what the cos curve looks like when it is divided up to 125 different levels. It is relatively good but for precise velocity control and when there multiple motors would be interacting, a better system should be found.

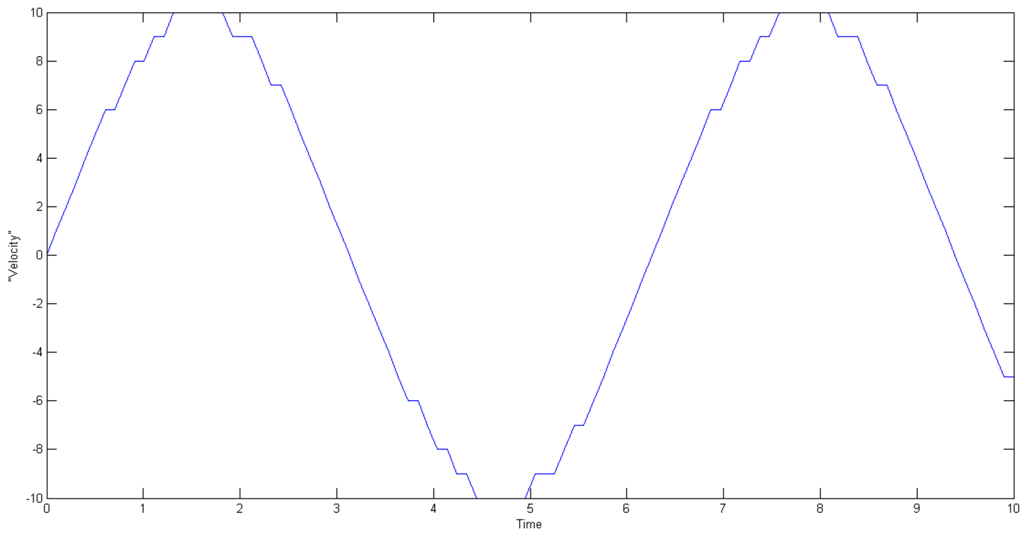

Figure 3: Output for low Number of Output Levelsl

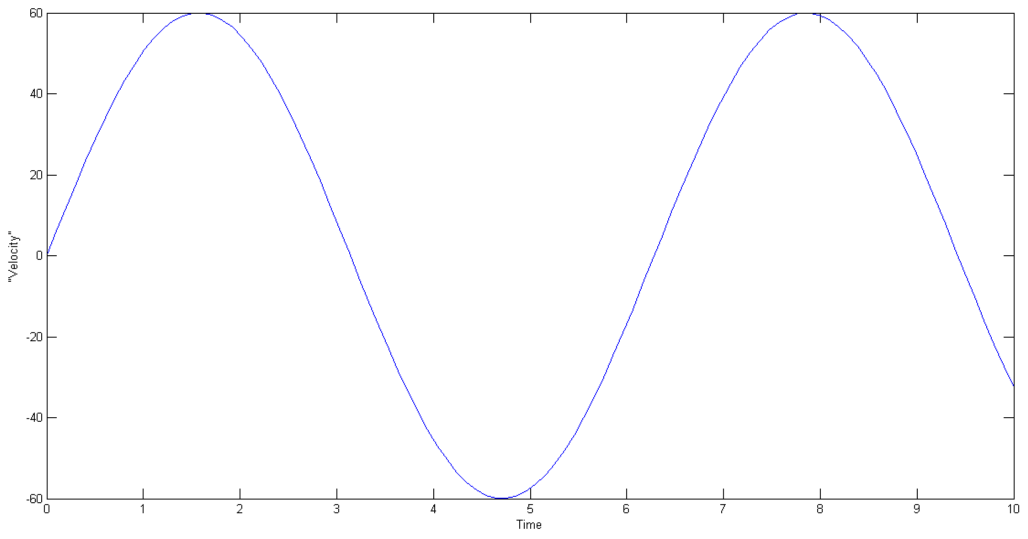

Figure 4: Output for High Number of Output Levelsl

From MATLAB, the ASCII value is then sent over the serial line using UART to the MSP430. The MSP430 takes the value and determines if it is a positive or negative number and determines its magnitude. It then changes the length of the delay function inversely proportionally to the magnitude of the velocity (a higher velocity means a shorter delay between each cycling of the coil polarity). The pseudocode below gives an idea of how this could be implemented (full code listings in the Appendix). The forward and backward functions being called each set the coils to the correct polarities following the order (or reverse order for the backwards function) of Table 1.

```
d = serialinput - 75
#define Delay
    for i < 100 + 100/|d|
if d>0
    forward()
else d<0
    backward ()
```

Results

The project worked successfully. Inputting a cosine curve into the MATLAB program yielded an output velocity on the motor that would having an increasing and then decreasing velocity in one direction before reversing direction and increasing and decreasing in velocity. It would continue to follow this pattern, which is characteristic of a velocity following a cos curve.

Conclusion

The project worked well overall. The one big flaw at this point is the limits on the output from MATLAB to the motor. This needs to be calibrated better and it would be nice to be able to send non-integer values for greater accuracy. Other extensions could be adding functionality to start with a position function and have MATLAB differentiate it to get the velocity profile. Also being able to connect two motors to trace out a shape in two dimensions would be another step towards building a functional printer.

Appendix

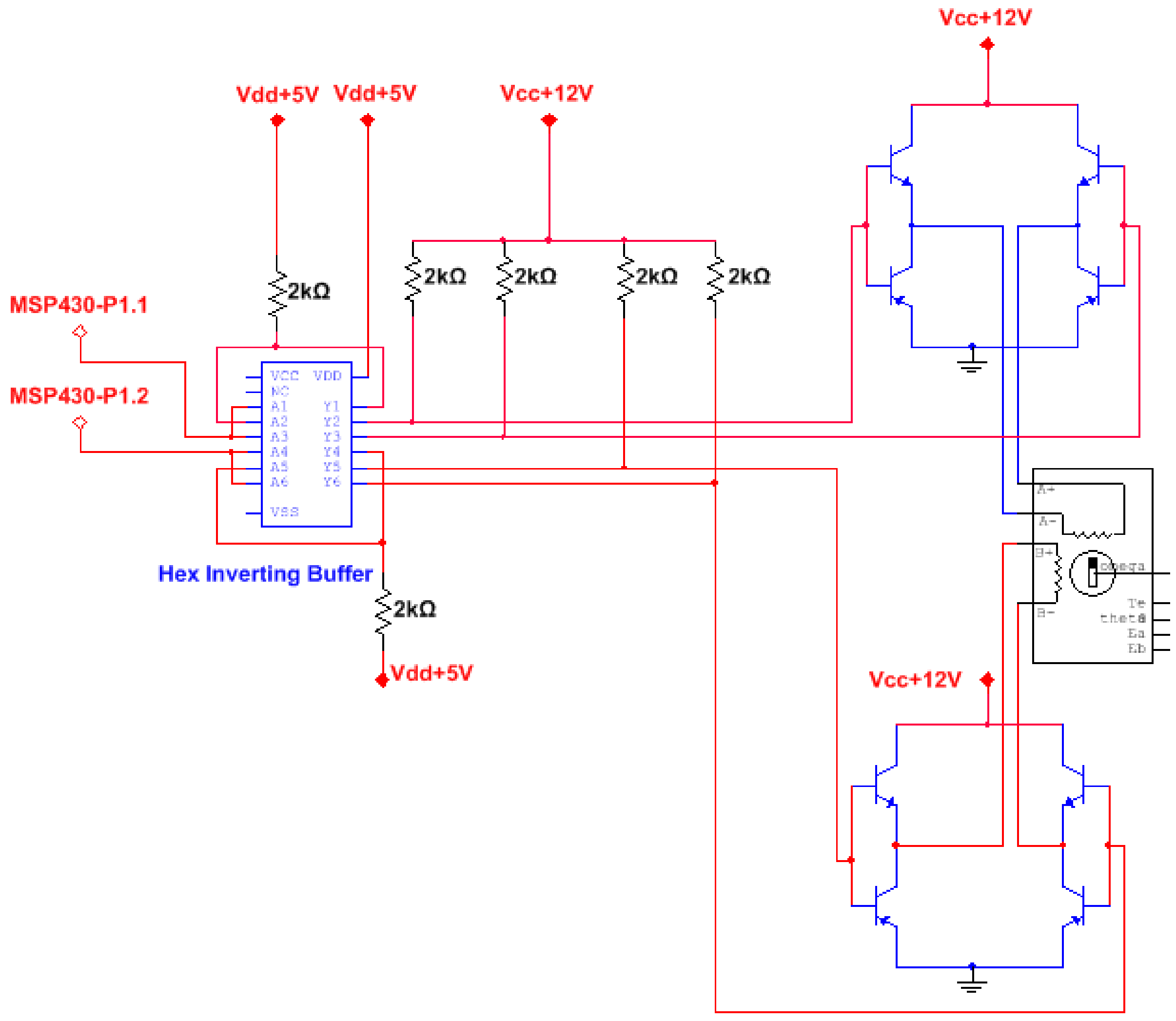

Figure 5: Overall Schematic for the Circuit

```
/* Andrew Grasberger
 * December 4, 2012
 * This code accepts velocity values from MATLAB
 * over a UART serial interface and interprets them
 * as delays so the velocity of the motor changes
 * along with the velosity value from MATLAB
 */
#include<msp430.h>                          //include header files and
                                            //initialize variables
int f;
int i;
int q;
#define DELAY   for(i=1;i<q;i++)          //the delay function, the value of q
                                          //is changed to
volatile int d;                           //change motor speed


void forward(void)                        // move clockwise by changing how
                                          //coils are turned on/off
  {

      P1OUT |= 0x40;
      P1OUT |= 0x20;

      DELAY;
      P1OUT &= 0x00;

      P1OUT &= ~0x40;
      P1OUT |= 0x20;

      DELAY;
      P1OUT &= 0x00;

      P1OUT &= ~0x40;
      P1OUT &= ~0x20;

      DELAY;
      P1OUT &= 0x00;

      P1OUT &= ~0x20;
      P1OUT |= 0x40;

      DELAY;
      P1OUT &= 0x00;
  }

void backward(void)                        //move counter clockwise by
                                            //turning coils on/off
  {

      P1OUT &= ~0x20;
      P1OUT |= 0x40;

      DELAY;
```

```
        P1OUT &= 0x00;

        P1OUT &= ~0x40;
        P1OUT &= ~0x20;

        DELAY;
        P1OUT &= 0x00;

        P1OUT &= ~0x40;
        P1OUT |= 0x20;

        DELAY;
        P1OUT &= 0x00;

        P1OUT |= 0x40;
        P1OUT |= 0x20;

        DELAY;
        P1OUT &= 0x00;
  }

void init(void)                                         // initialize UART interface
                                                        //with a baud rate of 9600
  {
    BCSCTL1 = CALBC1_1MHZ;                            // Set DCO
    DCOCTL = CALDCO_1MHZ;
    P1SEL = BIT1 + BIT2 ;                             // P1.1 = RXD, P1.2=TXD
    P1SEL2 = BIT1 + BIT2 ;                             // P1.1 = RXD, P1.2=TXD
    UCA0CTL1 |= UCSSEL_2;                             // SMCLK
    UCA0BR0 = 104;                                    // 1MHz 9600
    UCA0BR1 = 0;                                      // 1MHz 9600
    UCA0MCTL = UCBRS0;                                // Modulation UCBRSx = 1
    UCA0CTL1 &= ~UCSWRST;                             // **Initialize USCI
                                                      //state machine**
    IE2 |= UCA0RXIE;                                  // Enable USCI_A0 RX interrupt

    __bis_SR_register(GIE);                           // interrupts enabled
  }

void main(void)
  {
   WDTCTL = WDTPW + WDTHOLD;                         // Stop watchdog timer
   P1DIR |= 0xff;                                    //clear all pins and set them
                                                     //as outputs
   P1OUT &= 0x00;
    init();                                          //initialize the UART
                                                     //communication
    while(1)
    {
      if(d > 0)                                    //call the appropriate functions
                                                 //for what direction to move
      {
        forward();
      }
      else if(d < 0)
      {
        backward();a
      }
      else(d == 0);

      f = d;
      if(f<0)                                   //the moving functions interpret
                                                  //positive numbers so
      {                                         // make it always positive but d
                                              //already determined the direction
        f = -1*f;
      }
      q = 100 + 100/f;                        //q is the delay parameter, make
                                              //it inversely proportional to the
                                              //value of f (d)
                                              // so that for higher velocities,
                                              //the delay is shorter
    }

  }

#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{

    UCA0MCTL = UCBRS0;                              // Modulation UCBRSx = 1
    UCA0CTL1 &= ~UCSWRST;                           // **Initialize USCI
                                                    //state machine**
    IE2 |= UCA0RXIE;                                // Enable USCI_A0 RX interrupt

    __bis_SR_register(GIE);                         // interrupts enabled

      d = UCA0RXBUF - 75;                             //convert the sent value to a
                                                      //positive/neagive scale
}
```

main.c

```
% Andrew Grasberger December 4, 2012
%E72 Final Project This code interfaces with an MSP430 over the UART Serial
%interface, sending the value of a function at a certain time Which is
%interpreted as a delay by the ucontroller, so the function Inputted is the
%velocity function

s = serial('COM13','BAUD',9600);           % Create serial object and set
                                           %the baud rate
fopen(s)                                   % Open the serial port for r/w
t = linspace(0,8,100);                     %make an arbitrary time vector
r = 10*sin(t);                             % set the value of the velocity
                                           %profile
prompt = 'Enter a character (q to exit): '; %prompt for ending the code

while (myChar ~= 'q')                      % While user hasn't typed 'q'
    for i = 1:length(r)                    %go over the entire length of the
                                           %input function
        if r(i) > 10                       %the first two parts of the
                                           %if/else statement
            x = 125                        %set hard limits on the max
                                           %and min speed of the motor

        elseif r(i)< -10
            x = 0
        else                               % set the value of 75 as the '0'
                                           %value and have the
                if r(i)>0                  %delay in between motor steps
                                           %varry around it

                    x = 75 + round(5*r(i))
                else
                    x = 75 + round(5*r(i))
                end
        end
        fprintf(s,'%c',x);                %write the value of x over
                                          %the serial line
        pause(.5)                         %wait before writing the next value

    end
    myChar = input(prompt, 's');           % Get user input
end

fclose(s);                                 % Close the serial port
delete(s);                                 % Delete the serial object
```

Serial_com.m