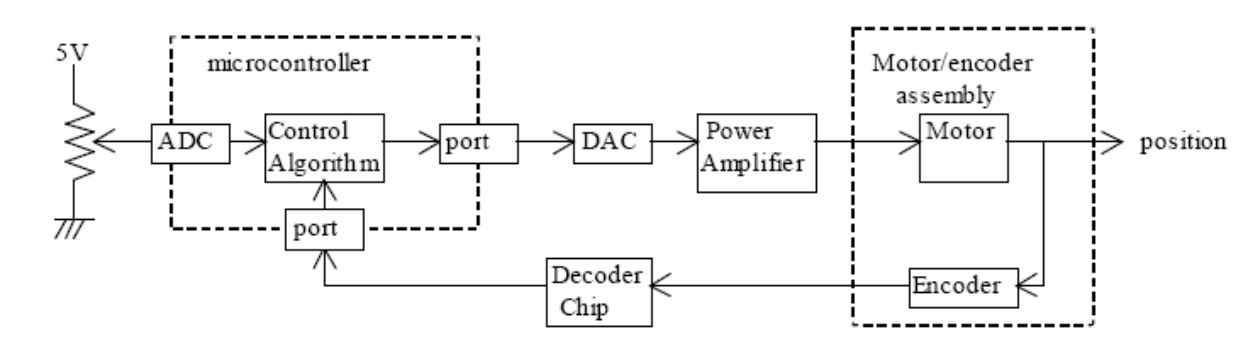
University of California, Berkeley
EE128, Fall 2006

Lab 7 – A Microcontroller Based Position/Speed Controller

Introduction

In this lab, we will develop and evaluate a microcontroller based position/speed control system. The system consists of a Atmel mircrocontroller, a Digital-to-Analog Converter, a power amplifier, a motor, an encoder, an decoder chip, and a Analog-to-Digital converter. The following block diagram shows the interconnection between these components.



Component Descriptions

As indicated in the above block diagram, an on-chip ADC channel is used to convert the analog position commend input to digital. The controller output is sent to a 12-bit DAC chip (Analog Device, AD7247A). The output of the DAC spans –5v (all zeros) to +5v (all 1's). The power amplifier is the same power amplifier used in the magnetic ball levitation experiment. The motor is a DC brush type motor and the optical encoder is by HP. The following paragraph is provided by the retailer of the motor/encoder assembly.

> Pancake Motor, Japan Servo, #DF10BE22-01, 127K9720. Permanent magnet. Reversible. Ball bearing. Continuous duty. Rated 24 VDC. No-load speed 3200 rpm @ 0.180 amp. 24 VDC. 1700 rpm 3.0 amps @ 30 oz-in load. Hewlett Packard 9100 series optical encoder on rear with 500 counts per revolution Two channel quadrature output which is TTL compatible. Encoder operates on 5 VDC @ 40 mA. max Motor dimensions: 4 3.4" max. dia. The decoder chip is HCTL-2016 by Agilent Technologies.
>
> The decoder chip is an up/down counter that counts the quadrature signals from the encoder. The output of the decoder chip is an 8-bit binary number. The counter can be reset by the microcontroller. The following figure is the schematics of the interface board.

Project Goal and Grading

The goal of the project is to close the position and speed control loop and to achieve the highest performance possible. The performance of your controller will be evaluated by the following criteria:

· Steady state error
· Frequency response
· Step transient response, i.e., overshoot, damping characteristics, settling time (both small step and
large step)
· Disturbance rejection

Your project will be graded base on the following factors:

· Controller performance (50%)
· Modeling and analysis of the system (30%)
· Clarity and completeness of the report. (20%)

The list of items you need to do is:

- Determine the Moment of Inertia
- Determine the Transfer function of the System + Controller
    - PID Controller (Hint: The two below are the same but with certain coefficients zero)
    - PD Controller
    - PI Controller
- Calculate coefficients for the three controllers
- Implement controllers
- Measure transient and steady-state characteristics
- Comment about what sampling frequency we should use
- Compare the experimental characteristics with the theoretical characteristics

A Sample Control Program

The following is a working sampling control program. This program closed the control loop with a simple PD controller. You can use this program as a template for your program. The logic flow of the program is shown in the flow chart below.

```
#include<mega16.h>

int abc=0,Cnt,LowCnt, HighCnt, adc_data, Output, speed, OldCnt;
int timer = 178;
float F_Cnt,position,F_speed,control,old_position, ref;

// This is a timer 0 initiated ISR.
interrupt [TIM0_OVF] void timer0_isr(void)
{
        ADCSRA=ADCSRA|0x40; // start ADC conversion
        TCNT0 = timer; // re-initialize counter
}

// This ISR is initiated by ADC at each end-of-conversion
interrupt [ADC_INT] void adc_isr(void)
{
        adc_data=-(ADCW<<1); //Get the Analog input
        PORTA.7=0; // Enable the output of the decoder chip & stop updating
        PORTA.6=0; //select high byte
        #asm("nop") // wait for the data
        #asm("nop") // wait for the data
        HighCnt=PIND; // Get the higher 8-bit count
        PORTA.6=1; //select low byte
        #asm("nop") // wait for the data
        #asm("nop") // wait for the data
        LowCnt=PIND; // Get the lower 8-bit count
        PORTA.7=1; // Output disable -> resume updating
        OldCnt=Cnt; // Save the old count
        Cnt= (HighCnt << 8)|LowCnt; // Combine the high and low counts
        F_Cnt = (float) Cnt;
        old_position = position;
        position=F_Cnt/318.3;
        F_speed = (position-old_position)/0.010;

        // STEP RESPONSE
        /*
        if (abc==100)
        {
                if (ref)
                {
                        ref = 0;
                }
                else
                {
                        ref = 1;
                }

                abc=0;
        }
        abc++;
        */

        // CONTROL LAW
        //-------------------------------------------------------
        control =  0.01*F_speed + 0.5*position + ref;
        //-------------------------------------------------------

        control = control/0.00244 + 2048;

        // CLIPPING/SATURATION CODE
        if (control < 0)
```

```
        {
                Output = 0;
        }
        else if (control > (2048+2047))
        {
                Output = (2048+2047);
        }
        else
        {
                Output = (int) control;
        }

        PORTB=Output; //send the low control byte to DAC
        PORTC=(Output>>8)|0xF0; //send the higher control byte
        PORTC.5=0; // strobe the data into DAC (falling edge)
        PORTC.5=1; // strobe the data into DAC (raising edge);
        //Output = (int) F_Cnt;
        Output = position/0.00244 + 2048;
        PORTB=Output; //send the low control byte to DAC
        PORTC=(Output>>8)|0xF0; //send the higher control byte
        PORTC.6=0; // strobe the data into DAC (falling edge)
        PORTC.6=1; // strobe the data into DAC (raising edge);
}

void main(void)
{
        DDRB = 0xff; // Port D all output (DAC lower 8-bit data)
        DDRC = 0xff; // Port C all output (DAC data and control)
        DDRA = 0xF0; // Port A upper nibble output (decoder control)
        DDRD = 0x0; // Port D all input (Decoder chip)
        abc=0;
        TCCR0 = 0x05; // divide by N pre-scalar (This and the next instruction
        // (the number 0x5 & 130) determine the sampling frequency).
        // you need to decide the best numbers to use.
        TCNT0=timer; // initialize timer 2
        TIMSK = 0x01; // unmask timer 2
        TIFR = 0x01; // do something to the interrupt flag reg.
        ADMUX = 0; // select analog channel 0
        ADCSRA = 0xCF; // ADC on, divide by 64, enable int. and start cover,
        #asm("sei") ; // Enable global interrupt
        while(1);
} // dummy loop
```