Lab 5: DC Motor Speed Control

by

Takuto Wada

Adam Woo

Engineering 323L

December 19, 2016

Introduction

A direct current (DC) motor is a rotary electrical machine which converts electrical current into mechanical power using the forces induced by the magnetic field. DC motors are used widely today for toys, industrial machines, automotive applications, and many other electronics.

A proportional–integral–derivative (PID) controller continuously calculates an error value as the difference between a desired setpoint and a measured process variable and applies a correction based on proportional, integral, and derivative terms, (denoted as P, I, and D respectively) which give their name to the controller type. PID controller is used to maintain desired speed of the DC motor for any resistance on the shaft of the motor by a load.

Problem Statement

The SiliconLab 8051F120 evaluation board will be used to perform DC motor speed control. To achieve the desired goal for this lab, the specifications that the DC motor embedded system must satisfy are:

- Sending Pulse-Width Modulation (PWM) signal to drive the DC motor at various speed by manipulating the duty cycle. PWM signal will be generated by the programmable counter array (PCA) of the C8051F120 board.
- Being able to adjust the desired speed of the DC motor using the potentiometer.
- Establishing a feedback system to measure the actual speed of the DC motor.
- Implementing a PID controller to make timely adjustment of the duty cycle to maintain the desired speed for various loads on the shaft of the motor.

- Displaying the desired speed and actual speed of the DC motor on the 16x2 LCD Display.

The block diagram in Fig.1 demonstrates how the completed embedded system will operate.

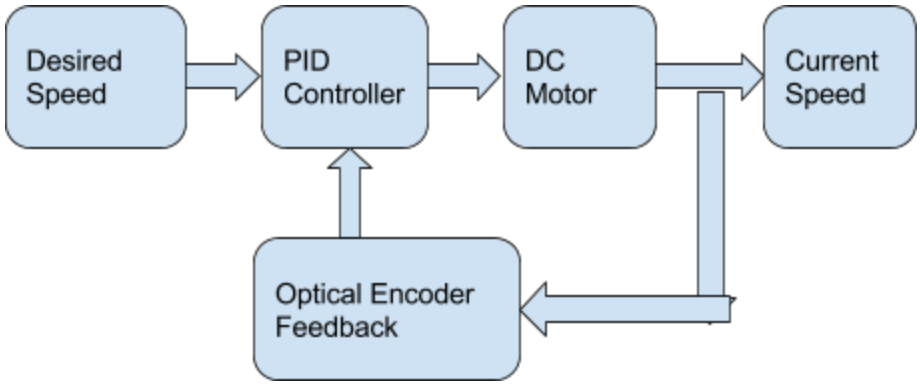

Figure 1 - Block diagram of DC motor embedded system operation

Purpose

The purpose of this lab is to construct the DC motor speed control embedded system using the SiliconLab 8051F120 board. To complete this task, one must understand the hardware component of the embedded system first. Then, one must develop, debug, and implement a C program to spin the DC motor with a PID controller that matches the desired speed with the actual speed.

Materials Used

Table 1 - Materials Used for DC Motor Speed Control

Chips	❑ H-Bridge (SN754410) ❑ Operational Amplifier (SN741)

Other Components	❑ SiliconLab 8051F120 ❑ Potentiometers (1k, 10k) ❑ Resistors (220 Ω x2,1000 Ω , 4700 Ω) ❑ 16x2 LCD Display ❑ DC Motor ❑ Cadet board ❑ Soldering Iron ❑ Wires
Design Application Tools	❑ Silicon Laboratories ❑ Configuration Wizard 2

Hardware Design Strategy and Construction of the System

To accomplish this lab, one must understand the functionality of the two chips, DC motor, potentiometer, 16x2 LCD display, and how they work together with the SiliconLab 8051F120 board. The H-bridge is used convert the digital PWM signal sent by the 8051 to a DC voltage to spin the DC motor. The chip layout of H-bridge and the six connection wires of the DC motor are shown in Fig. 2.

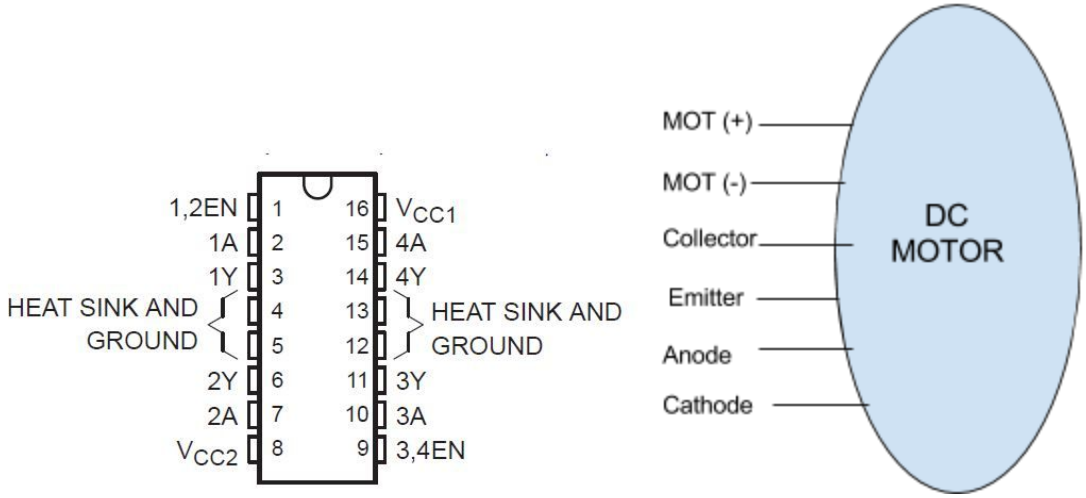

Figure 2- H-bridge Chip and DC Motor Connection wires

For the H-bridge chip to operate as desired, one must set the four pins (1,2EN, 3,4EN, VCC1, and VCC2) to HIGH and four pins to LOW (4 Heat sink and Ground). 1A will be connected to the 8051 and will receive PWM signal. 1Y will be connected to MOT(+) of the DC motor. Three connection wires of the motor (MOT (-), Emitter, and Cathode) will be set to LOW. Collector will be connected to a 1000 Ω resistor and Anode to a 220 Ω resistor and both will be connected to HIGH.

The DC motor has a built in optical encoder that uses a phototransistor/LED sensor to generate 80 sinusoidal signals (0.2-2 volts peak to peak) per turn. This signal must be processed through an operational amplifier to convert the small sinusoidal waves to signals for the 8051 to process. The process in which the actual speed of the motor is calculated will be discussed in *Software Design Strategy* Section. The chip layout of the Operational Amplifier is shown in Fig.3.

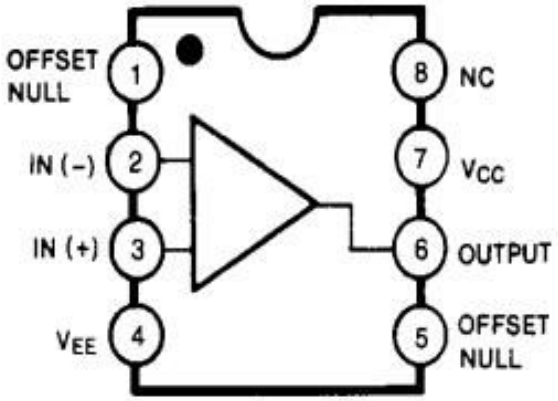

Figure 3 - Operational Amplifier Chip Layout

The Collector wire from the DC motor is connected to the non-inverting input of the op-amp. Inverting input of the op-amp is connected to a reference voltage that was determined appropriately by using the oscilloscope that displayed the output signal from the op-amp. The

reference voltage of 4.75V gave the best output to be processed by the 8051. A voltage divider circuit of 220 Ω and 4700 Ω with input voltage of 5V gave an output of 4.75V.

The potentiometer is used to output various voltages in the range from 0V to 5V by turning the knob. This will be used to determine the desired speed of the motor. This process will be discussed in *Software Design Strategy* Section.

The 16x2 LCD has 14 pins that needs to be wired correctly to properly display desired message on it. The circuit diagram of the 16x2 LCD is shown in Fig.4.

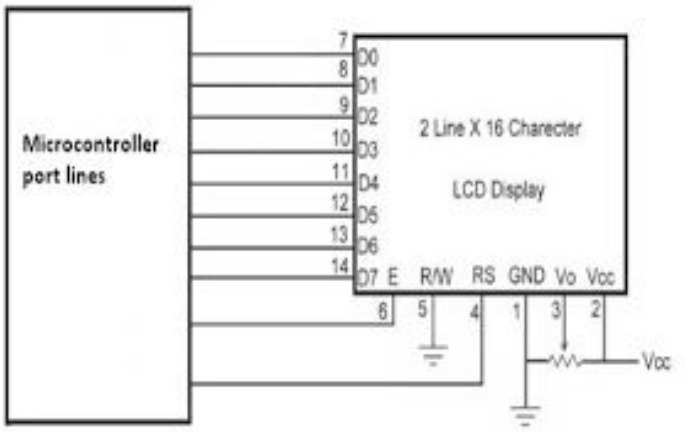

Figure 4 - 2x16 LCD Display Circuit Diagram

Vcc is connected to HIGH and GND is connected to LOW. The operating voltage (VO) is connected to a 10k Ω potentiometer to adjust the contrast of the LCD display. The RS, R/W, and the E pins were connected to P6.0, P6.1, and P6.2 respectively. The eight data bits (DB0-DB7) are connected to Port 5 of the 8051.

The completed *hardware* system is shown in Fig.5 below.

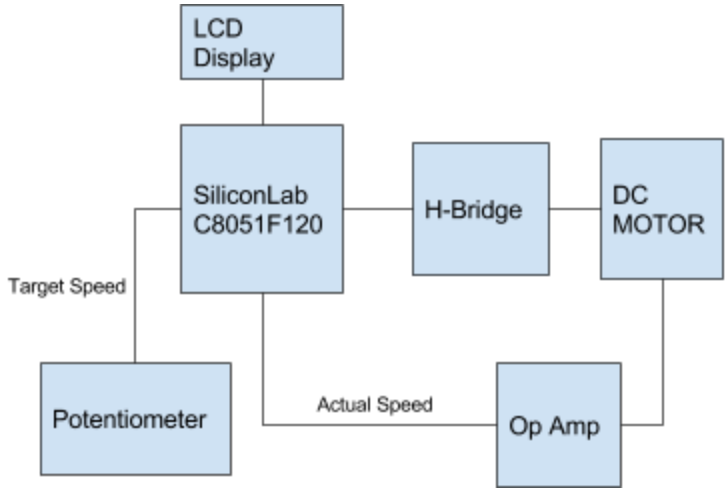

Figure 5 - Completed Hardware of DC motor embedded system

Establishing all desired connection, the completed hardware system of the DC motor embedded system is shown in Fig.6.

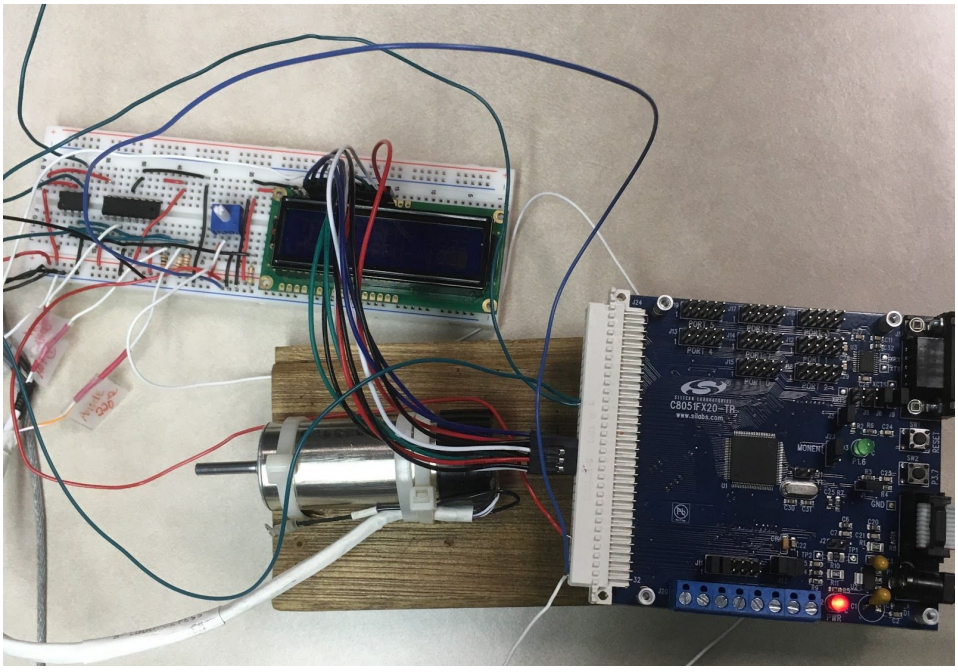

Figure 6 - DC motor embedded system

Software Design Strategy

Global Variables

In the beginning of this program, constants and variables that will be used are defined. Each function will be explained in detail later in this report.

```
#include <c8051F120.h>
#include <stdio.h>
//-----------------------------------------------------------------------
// Global Constants and variables
//-----------------------------------------------------------------------
#define SYSCLK 1225000
char indicator;
char LCD_init_flag = 0; //LCD initialization flag
char xdata LCD_display[32]; //hold the data to be shown on an LCD
int count;   //used as a counter
int max_rpm = 1000;
int min_rpm = 225;
int newcount;
int actual_rpm;
int target_rpm;
long pulse_count;
long voltage_reading;
short k;
short msec_count = 0; //each delay is about 1 msec
short RTH0, RTL0, RTH1, RTL1; //reload values for Timer_0 and Timer_1
unsigned char LCD_pointer=0;
sbit DB7 = P5^7;
sbit LCD_en = P6^2;
sbit LED = P1^6;
sbit RS = P6^0;
sbit RW = P6^1;
sfr16 ADC0 = 0xBE;
sfr16 RCAP3 = 0xCA;
sfr16 TMR3 = 0xCC;
```

Main

The main function will start by disabling watchdog timer to prevent the program from resetting every time an interrupt occurs. Main program calls all functions except the interrupt functions and will enter an infinite loop.

```
void main (void) //calling functions here
{
 WDTCN = 0xde;
 WDTCN = 0xad;
 LCD_Init_ISR();
 Oscillator_Init();
 Timer3_Init();
 Port_Init();
 Timer_Init();
 PCA_Init();
 ADC_Init();
 DISPLAY_ISR();
 Display_String();
 LED = 0;
 while (1);
}
```

Special Function Register Pages

SFRPAGE is a register which stores the current SFR Page Number or Context. It is one byte in size therefore the contents can be stored in a single byte type like char, in this case the variable SFRPAGE_SAVE.

Port Initialization

All ports that are used in the 8051F120 board are set up using Configuration Wizard. P1.6, entire Port 5(P5.0-P5.7), three I/O lines from Port 6(P6.0-P6.2), CEX0, and Timer 0 overflow interrupt will be used for the LCD. Crossbar is enabled.

```
void Port_Init(void)
{
 char SFRPAGE_SAVE = SFRPAGE;
 SFRPAGE = CONFIG_PAGE;
 P1MDOUT = 0x40; //P1.6(LED),
 P5MDOUT = 0xFF; //DB0-DB7 are set push-pull
 P6MDOUT = 0x07; //LCD-en (P6.2) RS(P6.0) and RW(P6.1) are set
 XBR0 = 0x08; //CEX0
 XBR1     = 0x06; /T0 and T0 interrupt is enabled.
 XBR2 = 0x40; // Enable crossbar
 SFRPAGE = SFRPAGE_SAVE;
}
```

Oscillator Initialization

In order for the system to run with the necessary timings, the internal oscillator must be set to the proper frequency. The oscillator is set to 12.25 MHz, which is also set as the system clock source. This will establish the timings for the system operations.

```
void Oscillator_Init (void)
{
 char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page
 SFRPAGE = CONFIG_PAGE; // Set SFR page
 OSCICN = 0x82; //use internal oscillator as system clock 12.25 MHz
 CLKSEL = 0x00; // Select the internal oscillator as SYSCLK source
 SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}
```

Timer Initialization

Timer 0 and Timer 1 will be used for the 16x2 LCD display. Timer 0 and Timer 1 will be used as a 16 bit timer. RTH0 and RTH0 values are set so that 1000 cycles will be about 1 msec. RTH1 and RTL1 values are set to refresh every 10 msec. Timer 0 and Timer 1 are enabled. Timer 0 overflow interrupt and Timer 1 overflow interrupt are also enabled.

```
void Timer_Init(void)
{
 char SFRPAGE_SAVE = SFRPAGE;
 SFRPAGE = TIMER01_PAGE;
 TMOD = 0x11; //Timer_0 and Timer_1 in 16-bit mode
 RTL0 = 0x17; //1000 cycles for about 1 msec
 RTH0 = 0xFC;
 TR0 = 1;
 ET0 = 1;
 RTL1 = 0xEF; //refresh rate of 100Hz (10 msec)
 RTH1 = 0xD8;
 TR1 = 1;
 ET1 = 1;
 EA = 1; //EA and TF0, TF1,and EA are enabled
 SFRPAGE = SFRPAGE_SAVE;
}
```

Timer 3 will be used for ADC . Timer 3 is set to Auto-Reload mode and uses SYSCLOCK as the clock source. RCAP3H and RCAP3L values are set to overflow about every 0.5msec. The sampling rate must be twice as fast as the highest frequency. Thus, by setting the timer 3 overflow interrupt every 0.5msec, accurate reading of the ADC will be ensured. Timer 3 is enabled while Timer 3 overflow interrupt is disabled.

```
void Timer3_Init(void)
{
 char SFRPAGE_SAVE = SFRPAGE;
 SFRPAGE = TMR3_PAGE;

 TMR3CN = 0x00;  // Stop timer 3, clear TF3
 TMR3CF = 0x08;  // use SYSCLK as time base
 RCAP3H  = 0xEC;
 RCAP3L = 0x7C;
 TMR3   = RCAP3;
 EIE2 &= ~0x01;  //disable timer 3 Interrupts
 TR3 = 1; //timer 3 enabled
 SFRPAGE = SFRPAGE_SAVE;
}
```

LCD Initialization

The LCD Initialization occurs using the Timer 0 overflow interrupt (interrupt 1). The interrupt begins by resetting TH0 and TL0 to the reload values. The program then checks if the LCD initialization flag has been set. If this flag has not been set, then the LCD initialization has not been completed. This if statement contains a switch loop that executes the various steps needed to set up the LCD. A example code can be seen below, with the whole code in the appendix.

```
void LCD_Init_ISR(void) interrupt 1
{
 char SFRPAGE_SAVE = SFRPAGE;
 EA = 0;
 TH0 = RTH0;
 TL0 = RTL0;
 EA = 1;
 if (LCD_init_flag == 0 )
 {
        msec_count++;
```

```
        SFRPAGE = CONFIG_PAGE;
        switch (msec_count)
        {
        case 5: case 10: case 11: case 12:      //function set four times
        LCD_en = 1;
        RS = 0;
        RW = 0;
        P5 = 0x3F;
        LCD_en = 0;                 // pulse enable
        break;
        case 13:
        …
        break;
        default:
        break;
        }                       //end of switch-case
 }
 else
 {
        Timer_ReInit();
 }
 SFRPAGE = SFRPAGE_SAVE;
}
```

After the LCD has been initialized by the switch loop, the final case will set the LCD initialization flag, telling the program that the initialization is complete. The program will then call the *Timer_ReInit()* function to reinitialize Timer 0, since it is no longer needed for the LCD.

Timer Re-Initialization

Timer 0 will be reinitialized as a 16-bit counter and the overflow interrupt is disabled. Timer 0 will be used to count the actual speed of the DC motor.

```
void Timer_ReInit(void)
{
 char SFRPAGE_SAVE = SFRPAGE;
 SFRPAGE   = TIMER01_PAGE;
 ET0 = 0;
 TMOD = 0x15;
 EA=0;
 TH0 = 0x00;
 TL0 = 0x00;
 EA=1;
 TR0=1;
 SFRPAGE = SFRPAGE_SAVE;
}
```

Analog to Digital Conversion (ADC) Initialization

An analog to digital converter (ADC) is an electronic device which converts varying analog signals into digital signals so that they can easily be read by the digital devices. The input analog signal has a certain voltage, which is given a numeric value among certain increments. In this case, the ADC has 12 bit incrementation. Therefore, a value of the input signal will correspond to a value between 0x0000 and 0xFFF0. ADC0 is enabled as continuous tracking. ADC is initiated on overflow of Timer 3. ADC0 is set to left justify, internal bias generator is enabled, and ADC0INT interrupt is enabled.

```
void ADC_Init()
{
 char SFRPAGE_SAVE = SFRPAGE;
 SFRPAGE   = ADC0_PAGE;
 ADC0CN          = 0x85;
 REF0CN = 0x03;
 AMX0SL = 0x00;
 EIE2    |= 0x02;
 SFRPAGE = SFRPAGE_SAVE;
}
```

The value in ADC0 will be used to calculate the target rpm between a maximum and minimum speed. The maximum voltage will provide the maximum speed, while the minimum voltage will provide the minimum speed.

Programmable Counter Array (PCA) Initialization

Programmable counter array was set to generate 8-bit PWG signal with 18.75% duty cycle initially. This duty cycle value was set arbitrarily, and can easily be changed to any other speed.

```
void PCA_Init()
```

```
{
 char SFRPAGE_SAVE = SFRPAGE;
 SFRPAGE   = PCA0_PAGE;
 PCA0CN          = 0x40;
 PCA0CPM0  = 0x42;
 PCA0CPH0  = 0xD0;                    //18.75% duty cycle.
 SFRPAGE = SFRPAGE_SAVE;
}
```

Display Interrupt

The display interrupt is triggered by the Timer 1 overflow every 20ms. Similarly to all timer overflow interrupts, the TH1 and TL1 values are reset first. If the LCD has been initialized, the program will run a switch loop that will change the line of the LCD display after the first 16 characters. After the following 16 characters of the second line, the switch loop will then return the cursor to home position.

```
void DISPLAY_ISR(void) interrupt 3
{
 char SFRPAGE_SAVE = SFRPAGE;
 SFRPAGE = CONFIG_PAGE;
 EA = 0;
 TL1 = RTL1; // refresh rate of 50Hz
 TH1 = RTH1; // about 20-msec interval
 EA = 1;

 if (LCD_init_flag == 1)
 {
        switch(LCD_pointer)
        {
        case 16:
        LCD_en = 1;
        RS = 0;
        RW = 0;
        P5 = 0xC0;                    //first char in the 2nd line
        LCD_en = 0;
        for(k=0; k<25; k++);          //add delay to settle
        break;
        case 32:                      //return cursor home
        LCD_en = 1;
        RS = 0;
        RW = 0;
        P5 = 0x02;                    //move cursor to home position
```

```
            LCD_en = 0;
            LCD_pointer = 0;
            for(k=0; k<25; k++);                    //add delay to settle
            break;
            default:
            break;
            }                                       //end of switch

            //display data on LCD
            LCD_en = 1;
            RS = 1;
            RW = 0;
            P5 = LCD_display[LCD_pointer];
            LCD_en = 0;
            LCD_pointer++;
 }

 SFRPAGE = SFRPAGE_SAVE;
}
```

The code after the switch loop updates the display location and then increases the LCD pointer for the next character.

Analog to Digital Interrupt

AD0_ISR interrupt 15 is where all of the calculations for actual and target rpm are done. Each interrupt updates the value of ADC0, which is used to calculate the target rpm between the minimum and maximum rpm. The equation *target_rpm* scales the 12 bit value according to the range of the desired rpm. An if statement prevents the target rpm from dropping below the minimum target rpm.

```
void AD0_ISR(void) interrupt 15
{
 if (newcount==500)
 {
            SFRPAGE=ADC0_PAGE;
            AD0INT = 0;
            target_rpm = (ADC0*max_rpm)/0xFFF0;
            if (target_rpm < min_rpm)
            {
                        target_rpm = min_rpm;
            }
```

```
            SFRPAGE = TIMER01_PAGE;

            pulse_count= TH0*0x100+TL0; //first convert TH0 and TL0 into one number (total pulse count)
            actual_rpm = pulse_count*60/80;
            EA=0;
            TH0=0;
            TL0=0;
            EA=1;

            Display_String();

            error = target_rpm-actual_rpm;
             if (error == 0)
            {
             }
            else if (error > 0 && error < 20)
            {
                        PCA0CPH0--;
            }
            else if (error < 0 && error > -20)
            {
                        PCA0CPH0++;
            }
            else if (actual_rpm <= max_rpm && actual_rpm >= min_rpm)
            {
                        error = (int)(target_rpm - actual_rpm)*.1;
                        PCA0CPH0 = PCA0CPH0 - error;
            }
            else if (actual_rpm <= min_rpm)
            {
                        PCA0CPH0--;
            }
            else if (actual_rpm >= max_rpm)
            {
                        PCA0CPH0++;
            }
 }
 newcount=newcount+1;
}
```

The Timer 0 counter is then read and converted into an integer that can be used to calculate the actual rpm of the motor. After reading the counter, the TH0 and TL0 values are reset to zero,

which allows the next reading to be taken. The *Display_String* function is then called to display the current target and actual rpm.

After displaying the rpm, the program will take different actions depending on the actual rpm. If the actual rpm is between the minimum and maximum target rpms, an equation will be used to correct for any difference between the actual and target values. This equation will vary the duty cycle proportionally to the difference of the actual and target rpm. If the program over compensates, the system will use a series of if statements to slowly speed up or slow down to maintain within the range of desired values and around the target rpm.

Display String

The function *Display_String* will be used to display desired characters and numbers on the 16x2 LCD. The target rpm is displayed in first line, while the second line displays the actual rpm in a similar manner.

```
void Display_String(void)
{
 // first line display Target RPM
 LCD_display[0] = 'T';
 LCD_display[1] = 'A';
 LCD_display[2] = 'R';
 LCD_display[3] = 'G';
 LCD_display[4] = 'E';
 LCD_display[5] = 'T';
 LCD_display[6] = ' ';
 LCD_display[7] = 'R';
 LCD_display[8] = 'P';
 LCD_display[9] = 'M';
 LCD_display[10] = '-';
 LCD_display[11] = ' ';
 //generate the target  rpm number
 LCD_display[12] = '0'+ target_rpm/1000;
 LCD_display[13] = '0'+ (target_rpm%1000)/100;
 LCD_display[14] = '0'+ ((target_rpm%1000)%100)/10;
 LCD_display[15] = '0'+ ((target_rpm%1000)%100)%10;
 //second line display Actual RPM
 … }
```

Results

After all desired connections among the 8051 and all hardware components, the DC motor embedded system was tested for various target rpm. Initially, the range of target rpm of the DC motor was set from 0 to 3000. However, this system failed to perform as desired and the range of the target rpm was set to 225 to 1000. This will be discussed in detail in the *Discussion and Conclusion* Section.

For various target rpm in the range 225 to 1000, the duty cycle of the PWM signal was properly able to be adjusted by the PID controller to provide actual rpm that matches the target rpm. The time for adjustment of the actual rpm took about 1 second at the most extreme. As the difference in target rpm and actual rpm gets bigger, it would take longer for the PID controller to adjust the duty cycle. If the difference is small, the duty cycle will be adjusted in less than a second to match the desired rpm and target rpm.

Discussion and Conclusion

The DC motor embedded system was successful in properly adjusting the duty cycle of the PWM signal to spin the DC motor at the target rpm. While the final system gave the proper rpm, the coding of the 8051 presented some unusual issues.

Theoretically, the counting and calculations for PID should be controlled by the Timer 1 overflow interrupt, but in practice, the program refused to count correctly within this function. To compensate for that error, the counting and caluctionation were placed within the ADC interrupt. This was able to make the counter update according to the varying rmp.

However, the accuracy of the counter then became an issue. There seemed to be a system limitation for the counter to read above 1300 pulses and below 200. This caused errors over about 1000 rpm. Because of this, an rpm limit was set to above 225 rpm and below 1000 rpm. The PID was implemented to work quickly between these two points. The counter was made more accurate by reading every half second. The LED=~LED instruction was used to test the delay between each reading.

Appendix

```
//-----------------------------------------------------------------------
// DC Motor PID Controller
//
// This program was written to set a DC motor's speed and correct for any resistance from a load.
// Because of system limitations, this code operated between 225 - 1000rpm.
//
// by Adam Woo & Takuto Wada
//
// Status: program successfully tested December 2016
//-----------------------------------------------------------------------

#include <c8051F120.h>
#include <stdio.h>

//-----------------------------------------------------------------------
// Global Constants and variables
//-----------------------------------------------------------------------
#define SYSCLK 1225000
char indicator;
char LCD_init_flag = 0; //LCD initialization flag
char xdata LCD_display[32]; //hold the data to be shown on an LCD
int count;
int max_rpm = 1000;
int min_rpm = 200;
int newcount;
int actual_rpm; //test rpm
int target_rpm; //test variable
long pulse_count;
long voltage_reading;
short k;
short msec_count = 0; //each delay is about 1 msec
short RTH0, RTL0, RTH1, RTL1; //reload values for Timer_0 and Timer_1
unsigned char LCD_pointer=0;

sbit DB7 = P5^7;
sbit LCD_en = P6^2;
sbit LED = P1^6;
sbit RS = P6^0;
sbit RW = P6^1;
sfr16 ADC0 = 0xBE;
sfr16 RCAP3 = 0xCA;
sfr16 TMR3 = 0xCC;

void ADC_Init(void);
void DISPLAY_ISR(void);
void Display_String(void);
void LCD_Init_ISR(void);
void Oscillator_Init(void);
void PCA_Init(void);
void Port_Init(void);
```

```
void Timer3_Init(void);
void Timer_Init(void);
void Timer_ReInit(void);

//-----------------------------------------------------------------------
// program starts here
//-----------------------------------------------------------------------
void main (void) //calling functions here
{
 WDTCN = 0xde;
 WDTCN = 0xad;
 LCD_Init_ISR();
 Oscillator_Init();
 Timer3_Init();
 Port_Init();
 Timer_Init();
 PCA_Init();
 ADC_Init();
 DISPLAY_ISR();
 Display_String();
 LED = 0;
 while (1);
}

//-----------------------------------------------------------------------
// Oscillator_Init
//
// This routine initializes the system clock to use the precision internal
// oscillator as its clock source.
//-----------------------------------------------------------------------
void Oscillator_Init (void)
{
 char SFRPAGE_SAVE = SFRPAGE; // Save Current SFR page
 SFRPAGE = CONFIG_PAGE; // Set SFR page
 OSCICN = 0x82; //use internal oscillator as system clock 12.25 MHz
 CLKSEL = 0x00; // Select the internal oscillator as SYSCLK source
 SFRPAGE = SFRPAGE_SAVE; // Restore SFR page
}

//-----------------------------------------------------------------------
// Port_Init();
//-----------------------------------------------------------------------
void Port_Init(void)
{
 char SFRPAGE_SAVE = SFRPAGE;
 SFRPAGE = CONFIG_PAGE;
 P1MDOUT = 0x40; //P1.6(LED),
 P6MDOUT = 0x07; //LCD-en (P6.2) RS(P6.0) and RW(P6.1) are set
 P5MDOUT = 0xFF; //DB0-DB7 are set push-pull
 XBR0 = 0x08;
 XBR1   = 0x06;
 XBR2 = 0x40; // Enable crossbar
 SFRPAGE = SFRPAGE_SAVE;
}
```

```
//-----------------------------------------------------------------------
// Initialization of programmable counter array
//-----------------------------------------------------------------------
void PCA_Init()
{
  char SFRPAGE_SAVE = SFRPAGE;
  SFRPAGE   = PCA0_PAGE;
  PCA0CN            = 0x40;
  PCA0CPM0  = 0x42;
  PCA0CPH0  = 0xD0;
  SFRPAGE = SFRPAGE_SAVE;
}

//-----------------------------------------------------------------------
// Initialization of analog to digital conversion
//-----------------------------------------------------------------------
void ADC_Init()
{
  char SFRPAGE_SAVE = SFRPAGE;
  SFRPAGE   = ADC0_PAGE;
  ADC0CN            = 0x85;
  REF0CN = 0x03;
  AMX0SL = 0x00;
  EIE2    |= 0x02;
  SFRPAGE = SFRPAGE_SAVE;
}

//-----------------------------------------------------------------------
// Initialization of Timer_0 and Timer_1 as 16 bit
//-----------------------------------------------------------------------
void Timer_Init(void)
{
  char SFRPAGE_SAVE = SFRPAGE;
  SFRPAGE = TIMER01_PAGE;
  TMOD = 0x11; //Timer_0 and Timer_1 in 16-bit mode
  RTL0 = 0x17; //1000 cycles for about 1 msec
  RTH0 = 0xFC;
  TR0 = 1;
  ET0 = 1;
  RTL1 = 0xEF; //refresh rate of 100Hz (10 msec)
  RTH1 = 0xD8;
  TR1 = 1;
  ET1 = 1;
  EA = 1; //EA and TF0, TF1,and EA are enabled
  SFRPAGE = SFRPAGE_SAVE;
}

//-----------------------------------------------------------------------
// Reinitialization of Timer_0 as a counter
//-----------------------------------------------------------------------
void Timer_ReInit(void)
{
  char SFRPAGE_SAVE = SFRPAGE;
```

```
 SFRPAGE   = TIMER01_PAGE;
 ET0 = 0;
 TMOD = 0x15;
 EA=0;
 TH0 = 0x00;
 TL0 = 0x00;
 EA=1;
 TR0=1;
 SFRPAGE = SFRPAGE_SAVE;
}

//-----------------------------------------------------------------------
// Initialization of Timer_3 as a ADC overflow interrupt
//-----------------------------------------------------------------------
void Timer3_Init(void)
{
 char SFRPAGE_SAVE = SFRPAGE;
 SFRPAGE = TMR3_PAGE;

 TMR3CN = 0x00;  // Stop timer 3, clear TF3
 TMR3CF = 0x08;  // use SYSCLK as time base
 RCAP3H  = 0xEC;
 RCAP3L = 0x7C;
 TMR3   = RCAP3;
 EIE2 &= ~0x01;  //disable timer 3 Interrupts
 TR3 = 1;
 SFRPAGE = SFRPAGE_SAVE;
}

//-----------------------------------------------------------------------
// LCD_Init_ISR
//
// setup Timer_0 overflow interrupt service routine
// to perform LCD Initialization Function
//-----------------------------------------------------------------------
void LCD_Init_ISR(void) interrupt 1
{
 char SFRPAGE_SAVE = SFRPAGE;
 EA = 0;
 TH0 = RTH0;
 TL0 = RTL0;
 EA = 1;
 if (LCD_init_flag == 0 )
 {
  msec_count++;
  SFRPAGE = CONFIG_PAGE;
  switch (msec_count)
  {
   case 5: case 10: case 11: case 12:
   //function set four times
   LCD_en = 1;
   RS = 0;
   RW = 0;
   P5 = 0x3F;
```

```
    // pulse enable
    LCD_en = 0;
    //LED = ~LED;
    break;
    case 13:
    //send display off
    LCD_en = 1;
    RS = 0;
    RW = 0;
    P5 = 0x08;
    // pulse enable
    LCD_en = 0;
    break;
    case 14:
    // send display clear
    LCD_en = 1;
    RS = 0;
    RW = 0;
    P5 = 0x01;
    // pulse enable
    LCD_en = 0;
    break;
    case 16:
    // send entry mode set
    LCD_en = 1;
    RS = 0;
    RW = 0;
    P5 = 0x06;
    // pulse enable
    LCD_en = 0;
    break;
    case 18:
    // send display ON
    LCD_en = 1;
    RS = 0;
    RW = 0;
    P5 = 0x0F;
    // pulse enable
    LCD_en = 0;
    // set the LCD_init_flag
    //LED = 1; //lit the LED to indicate the end of LCD initialization
    LCD_init_flag = 1;
    break;
    default:
    break;
   } //end of switch-case
  }// end of if
  if (LCD_init_flag == 1 )
  {
    Timer_ReInit();
  }//end of if
  SFRPAGE = SFRPAGE_SAVE;
}
```

```
//-----------------------------------------------------------------------
// Display_String()
//
// set up a test display message
//-----------------------------------------------------------------------
void Display_String(void)
{
 // first line dislay data
 LCD_display[0] = 'T';
 LCD_display[1] = 'A';
 LCD_display[2] = 'R';
 LCD_display[3] = 'G';
 LCD_display[4] = 'E';
 LCD_display[5] = 'T';
 LCD_display[6] = ' ';
 LCD_display[7] = 'R';
 LCD_display[8] = 'P';
 LCD_display[9] = 'M';
 LCD_display[10] = '-';
 LCD_display[11] = ' ';
 //generate the rpm number
 LCD_display[12] = '0'+ target_rpm/1000;
 LCD_display[13] = '0'+ (target_rpm%1000)/100;
 LCD_display[14] = '0'+ ((target_rpm%1000)%100)/10;
 LCD_display[15] = '0'+ ((target_rpm%1000)%100)%10;
 //second line display data
 LCD_display[16] = 'A';
 LCD_display[17] = 'C';
 LCD_display[18] = 'T';
 LCD_display[19] = 'U';
 LCD_display[20] = 'A';
 LCD_display[21] = 'L';
 LCD_display[22] = ' ';
 LCD_display[23] = 'R';
 LCD_display[24] = 'P';
 LCD_display[25] = 'M';
 LCD_display[26] = '-';
 //genearte rpm reading for display
 LCD_display[27] = ' ';
 LCD_display[28] = '0'+ actual_rpm/1000;
 LCD_display[29] = '0'+ (actual_rpm%1000)/100;
 LCD_display[30] = '0'+ ((actual_rpm%1000)%100)/10;
 LCD_display[31] = '0'+ ((actual_rpm%1000)%100)%10;
}

//-----------------------------------------------------------------------
// LCD Display Refresh Function
//-----------------------------------------------------------------------
void DISPLAY_ISR(void) interrupt 3
{
 char SFRPAGE_SAVE = SFRPAGE;
 SFRPAGE = CONFIG_PAGE;
 EA = 0;
 TL1 = RTL1; // refresh rate of 50Hz
```

```
 TH1 = RTH1; // about 20-msec interval
 EA = 1;

 if (LCD_init_flag == 1) //&& (LCD_busy_flag()==0))
 {
  switch(LCD_pointer)
  {
   case 16:
   LCD_en = 1;
   RS = 0;
   RW = 0;
   P5 = 0xC0; //set DD RAM address to 40H
   //first char in the 2nd line
   LCD_en = 0;
   for(k=0; k<25; k++); //add delay to settle
   break;
   case 32: //return cursor home
   LCD_en = 1;
   RS = 0;
   RW = 0;
   P5 = 0x02; //move cursor to home position
   LCD_en = 0;
   LCD_pointer = 0;
   for(k=0; k<25; k++); //add delay to settle
   break;
   default:
   break;
  }//end of switch

  //display data on LCD
  LCD_en = 1;
  RS = 1;
  RW = 0;
  P5 = LCD_display[LCD_pointer];
  LCD_en = 0;
  LCD_pointer++;
 }
 SFRPAGE = SFRPAGE_SAVE;
}

//-----------------------------------------------------------------------
// Timer 3 Overflow Interrupt - controls ADC: outputs ADC0H and ADC0L based on 0-5 V reading to be used in
target RPM calc
//-----------------------------------------------------------------------
void AD0_ISR(void) interrupt 15
{
 if (newcount==500)
 {
  SFRPAGE=ADC0_PAGE;
  AD0INT = 0;
  voltage_reading = ADC0;
  target_rpm = ((voltage_reading*max_rpm))/0xFFF0;
  if (target_rpm < min_rpm)
  {
```

```
    target_rpm = min_rpm;
   }
   SFRPAGE = TIMER01_PAGE;

   pulse_count= TH0*0x100+TL0; //first conver TH0 and TL0 into one number (total pulse count)
   actual_rpm = pulse_count*60/80;

   EA=0;
   TH0=0;
   TL0=0;
   EA=1;

   Display_String();

   error = target_rpm-actual_rpm;
   if (error == 0)
   {
   }
   else if (error > 0 && error < 20)
   {
    PCA0CPH0--;
   }
   else if (error < 0 && error > -20)
   {
    PCA0CPH0++;
   }
   else if (actual_rpm <= max_rpm && actual_rpm >= min_rpm)
   {
    error = (int)(target_rpm - actual_rpm)*.1;
    PCA0CPH0 = PCA0CPH0 - error;
   }
   else if (actual_rpm <= min_rpm)
   {
    PCA0CPH0--;
   }
   else if (actual_rpm >= max_rpm)
   {
    PCA0CPH0++;
   }
  }
  newcount=newcount+1;
}
```