A Modeling and Controls Course using Microcontrollers

Hugh Jack, Andy Blauch
Associate Professor / Assistant Professor
Padnos School of Engineering
Grand Valley State University
Grand Rapids, MI
email: jackh@gvsu.edu, blaucha@gvsu.edu

1. Introduction

As with most engineering programs, we offer a dynamic systems modeling and control course (EGR 345) to our students in Mechanical and Manufacturing Engineering [1][2]. This course has evolved since it was originally offered, with the goal of producing graduates who are capable of successfully implementing control systems. Towards this goal, the laboratory component of the course is very important. Originally the laboratories were based on experiments with discrete mechanical components and data acquisition to allow analysis of the results. While this was very effective in improving laboratory skills, it did not help prepare the students to implement actual control systems.

Previously the laboratory made extensive use of Labview for data acquisition and control. This did allow students to quickly build systems, but students rarely saw beyond the graphical interface. The laboratory component of EGR 345 was revised to build upon two previous courses, one in basic programming with C (EGR 261) or Java (CS 162), and a subsequent course in digital systems using 68HC11 microcontrollers (EGR 226). In the laboratory students use 68HC11 single board computers with 32K of memory for data collection and control [3][4]. Programs are written in the C programming language and compiled with the free GCC compiler. The typical laboratory focuses on motor control using a transistor for speed control, or an H-bridge for bidirectional motion. The motor speed or position is read using a potentiometer, tachometer or encoder. The students learn to construct feedback loops by converting block diagrams to C programs, circuits and mechanical systems. Similar efforts have been reported before in mechatronics courses for non-electrical engineering students [6], but dynamic systems modeling courses tend to focus on more traditional approaches [7].

The sequence of the laboratories basically begins with a review of C programming, analog inputs, Pulse Width Modulated (PWM) outputs and interrupt driven subroutines. Through the following laboratories students create and analyze a variety of velocity and position control systems. The laboratory exercise progressively introduces topics such as data collection, deadband compensation, feedback control and measuring parameters of permanent magnet DC motors. Other topics in the laboratories include Scilab (a Matlab clone) [5], Variable Frequency AC Drives, Labview and Simulink.

The normal laboratory structure requires that students prepare programs and do appropriate calculations before the laboratory. During the lab sessions the students build and test a system, and then use it to collect data that they may compare to the theoretical predictions.

The remainder of this paper focuses on the laboratory structure in detail so that others will be able to implement similar laboratories.

2. The Pedagogy

The lecture portion of the course examines the topics listed below. Through the lectures and laboratories, students use Scilab and C programming to solve problems numerically.

Translation
Calculus and differential equations
Numerical methods
Rotation
Input-output equations
Circuits
Feedback controllers
Fourier and root-locus analysis
Converting between analog and digital
Laplace transforms
Sensors
Actuators
Motion control

The laboratory exercises have been designed to emphasize control systems as soon as possible, as shown in the list below. The first week is used to review the use of 68HC11 controllers and C programming, as learned in EGR 226. In the second week the students build a feedback controller for motor speed control. By the end of Lab 6 students are able to design and build a sophisticated motion control systems, including deadband compensation, an interrupt driven feedback loop, setpoint planning and scheduling with a simple user interface. The remainder of the laboratory work expands the basic controller knowledge into areas that are more practical and/or theoretical.

Lab 1 - Analog I/O with the 68HC11 - Review material from EGR 226 and use the 68HC11 to do analog I/O, interrupts and PWM outputs with C programs.
Lab 2 - A Feedback Controller - Use a Darlington coupled NPN transistor to drive a motor, which drives a second motor acting as a velocity feedback. A proportional control program is written in C.
Lab 3 - Numerical Methods with Scilab - Scilab is introduced to support the computational needs of the students later in the labs and lectures.
Lab 4 - Deadband Compensation for Bidirectional Motion - This lab introduced an H-bridge to allow bi-directional motion. The effects of stiction were examined, and compensated for in software.
Lab 5 - Position Control with an Encoder - Quadrature/binary encoders were introduced for measuring motor position. This was used to implement a PI controller
Lab 6 - Motion Control - A motion planning method was used to generate smooth motion profiles. These were then supplied to the system as setpoints.
Lab 7 - Modelling Brushed DC Motors - Permanent magnet DC motors were modeled as differential equations and their parameters measured. This lab also required students to use the 68HC11 board to collect analog data.
Lab 8 - System Modeling - Students measured motor parameters and then used these to compare the actual and theoretical responses of the feedback controllers.
Lab 9 - Variable Frequency Drives (Allen Bradley Ultra 161s) - This lab introduced the students to industrial motor controllers for AC induction motors. During this lab students were able to set PID parameters and see a trapezoidal motion profile.
Lab 10 - IO Using Labview - Labview was introduced so that students were aware of other industrial options for data acquisition and control.
Lab 11 - Torsion - Students built a torsional pendulum, estimated the frequency of oscillation, and then verified it using Labview.
Lab 12 - System Simulation with Simulink - Students were introduced to Simulink for system Simulation.

Labview was used as the primary laboratory programming tool in previous years. This simplified the process of reading inputs and setting outputs. However, students often struggled with concepts such as sampling times and data ranges. They often encountered difficulties when constructing programs that required operational mode changes, or varied data types. However, with Labview now placed after the use of microcontrollers, and only used for two laboratories, students were more comfortable with Labview and able to do more than in previous years.

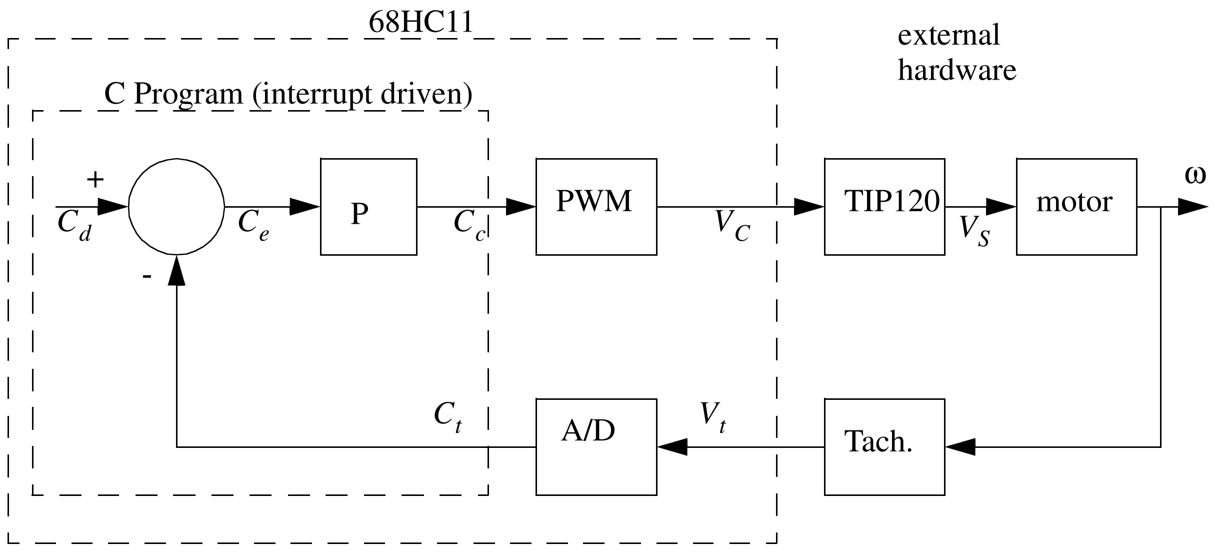
The 68HC11 based laboratories made use of an Axiom development board with 32K of RAM. This made it possible to write control programs in C using integer mathematics. When used properly the boards were reliable, but they occasionally failed when connected improperly. In many cases this only required replacing an inexpensive component. To help overcome this problem capacitors and resistors have been used to limit transient currents. In general the components for the laboratories were very inexpensive. For example the general component costs are,

68HC11 Axiom Board $89
mechanical encoder $2.50
potentiometer $2.50
L293D $0.50
TIP120 $0.50
motor $2.00

Two of the laboratories are described in the following sections.

3. A Velocity Feedback Controller

In the second laboratory students design and build a velocity controller. Although not formally introduced yet, a block diagram is used to present the control system, as shown in Figure 1. Care is taken to present the relationship between the block diagram and actual system components. Aside from the summation block, students find these diagrams to be very intuitive. This block diagram shows a proportional feedback loop implemented in an interrupt subroutine. The A/D input is handled by hardware in the microcontroller, and the PWM output is generated by a provided library. The PWM output drives a power transistor, which in turn drives a motor. Another motor is connected as a tachometer to provide a velocity feedback. Students implement their control software using this diagram, and some provided subroutine examples. As suggested in the variable definitions below all of the math operations are done with integer values. The analog input range is 0 to 5V. The PWM output is at logic levels (0 and 5V) but it switches a transistor to drive heavier loads. (Note: Vs will be proportional to Vc, but they may have different absolute magnitudes.)



Where,

C_d = Desired tach. speed - input via keyboard (0-255)

C_t = Actual tach. speed read via A/D (unsigned char, 0-255)

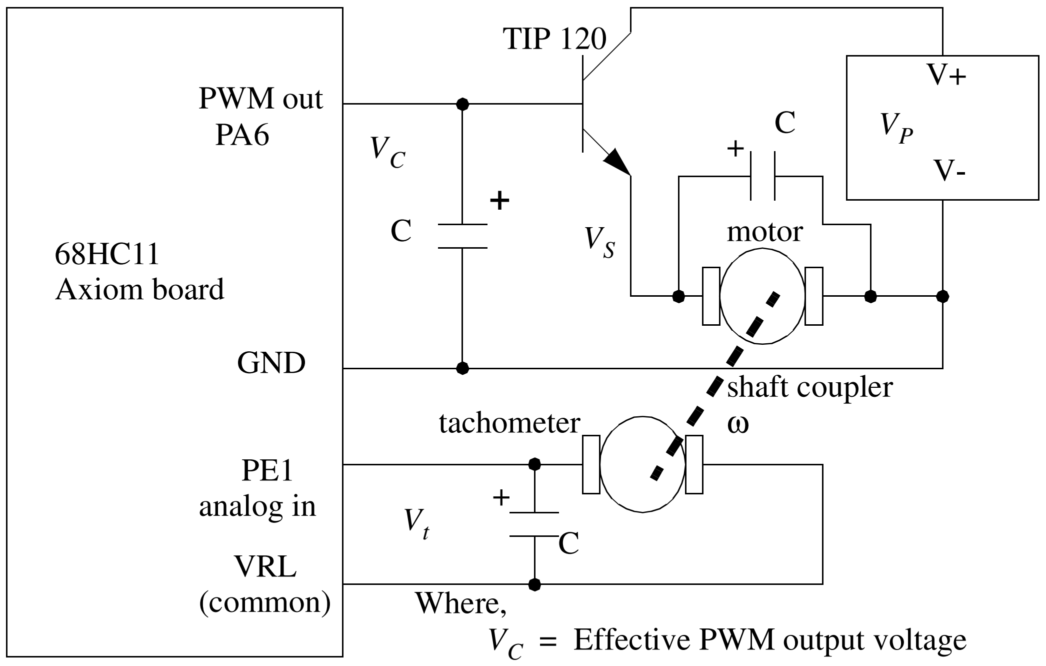
$C_e = C_d - C_t$ = System error (int)

$C_c = PC_e$ = Control value to output via PWM (unsigned char, 0-255)

P = The controller gain (int)

Figure 1 - Block Diagram for a Proportional Velocity Control System

The electrical schematic for the controller was designed to be very simple as shown in Figure 2. The simplicity of the circuit allows students to connect it within a few minutes. Capacitors are used to ‘short’ voltage spikes caused by induction and switching. (Note: it may also be useful to limit currents by putting 10K resistors in series with each of the 68HC11 inputs.) The tachometer can be a simple DC motor. Indeed, the same model motor may be used for the motor and tachometer.



Where,

V_C = Effective PWM output voltage

V_S = Effective motor voltage

V_P = Power supply voltage (3.8V)

V_t = Tachometer voltage

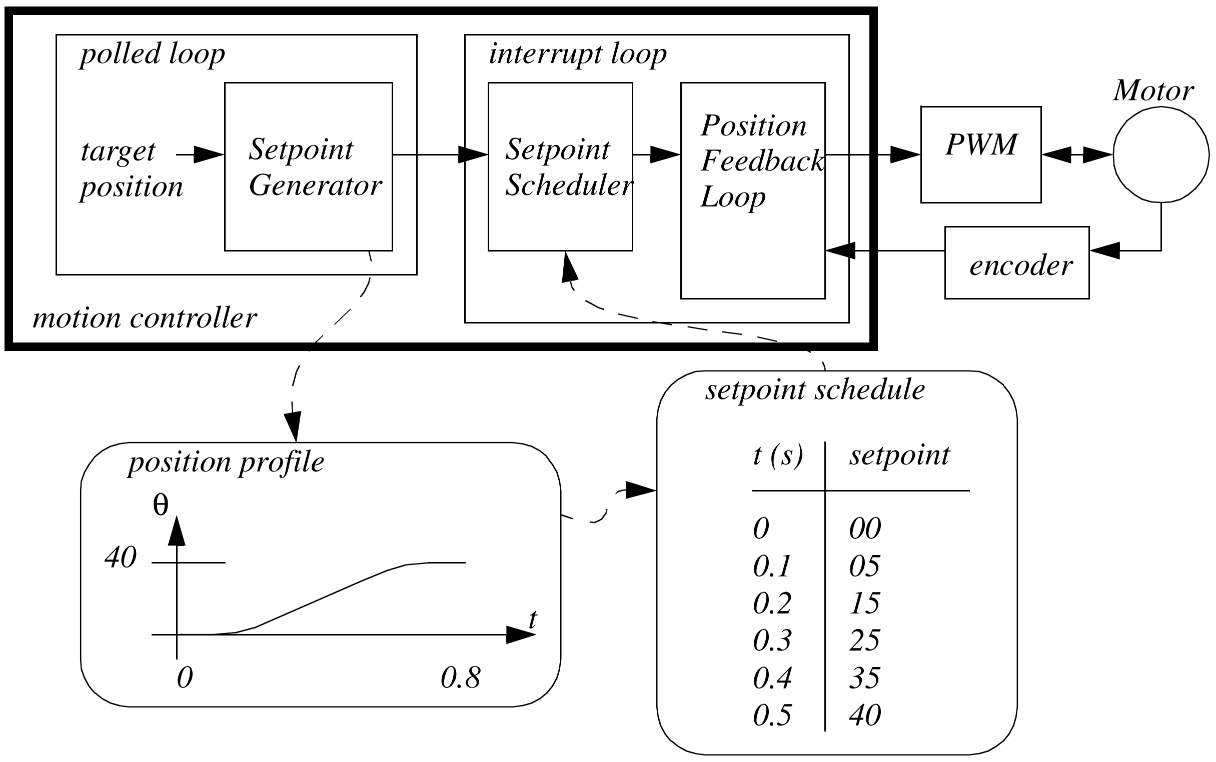
ω = Angular velociy of motor/tachometer

Figure 2 - Electrical Schematic for the Velocity Controller

The average student was able to successfully complete this lab within the 3 hour time block. At the end of the laboratory period they had a fully functional controller that would allow them to control the motor speed. They could vary the setpoint and the proportional gain and observe the effect. Most students were able to notice the motor deadband, and that larger gains made the system more responsive. The limitations of integer mathematics did create some difficulties when students over/underflowed the range. This will be rectified in the following course offerings with additional tutorials and prelaboratory materials.

4. Position Control with Encoder Feedback

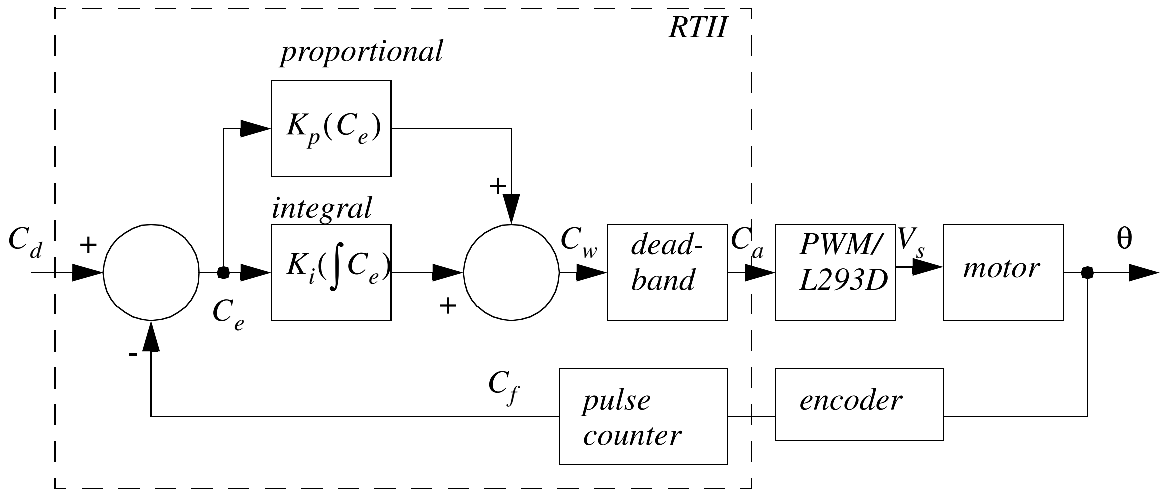
In Laboratory 6 the students developed a motion control system, as shown in Figure 3. Their controllers must allow a desired distance and time to be entered. These are used to generate setpoints along a smooth motion path. A feedback control loop is then used to ‘chase’ these points. A sample program is given in Appendix A and function names are referred between ‘[....]’.



t (s)	setpoint
0	00
0.1	05
0.2	15
0.3	25
0.4	35
0.5	40

Figure 3 - Motion Control System

The feedback control loop for the system is shown in Figure 4 [`void RTI_ISR()`]. The controller includes a proportional and integral component, as well as deadband compensation [`int DeadBand()`]. The deadband subroutine will increase the output PWM values to overcome the stiction of the motor, so that all non-zero voltages result in motion. A subroutine [`void EncoderUpdate()`] is used to decode the quadrature output of the encoder to measure position. In this case the PWM output is still unidirectional, but the use of direction outputs allowed the H-bridge to switch the motor voltage polarity [`void SetOutput(int value)`].



where,

C_f = feedback position (int -32768 to 32767)

C_d = desired position (int -32768 to 32767)

C_e = system error (int -32768 to 32767)

K_i, K_p = controller integral and proportional gain constants

C_w = wanted output (int -32768 to 32767)

C_a = actual PWM output (unsigned char 0 to 255)

Figure 4 - The Feedback Control Loop

This motion control system is bidirectional, and uses an H-Bridge as shown in Figure 5.

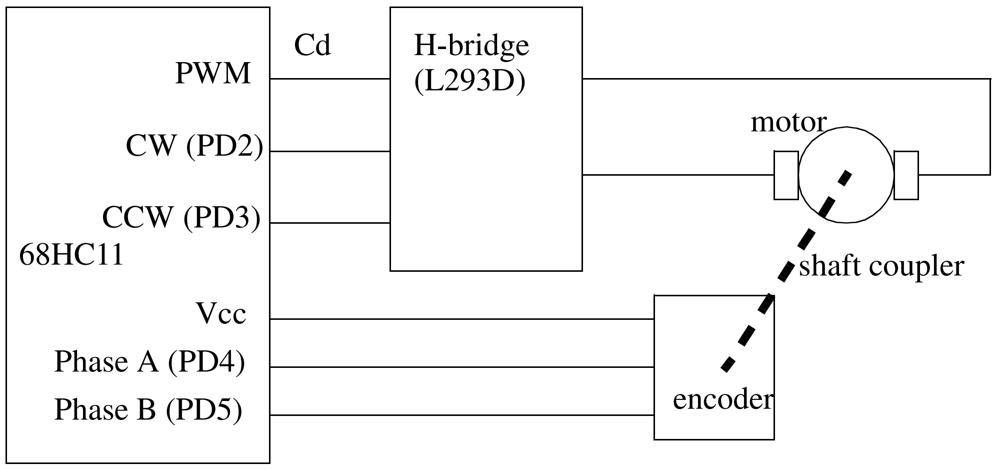


Figure 5 - Bidirectional H-Bridge Schematic

5. Future Issues

The laboratory experiences were complimentary to the lecture material, allowing students to grasp difficult control concepts easily. In the laboratory, students were able to build more control systems with a higher level of complexity in a shorter time than they could using Labview.

At present we are investigating the use of a student owned microcontroller board for use in the laboratories.

References

[1] Jack, H., “Dynamic System Modeling and Control”, http://claymore.engineer.gvsu.edu/~jackh/books
[2] Jack, H., EGR 345 Course homepage, http://claymore.engineer.gvsu.edu/~jackh/eod/egr345.html
[3] Axiom board site, http://www.axman.com
[4] Blauch, A., Sterian, A., A Modified GCC Compiler for the 68HC11, http://claymore.engineer.gvsu.edu/~jackh/eod/courses/egr345/media/gcc-68hc11-installer.exe
[5] Scilab, www.scilab.org
[6] Linde, E., Dolan, D., Batchelder, M., “Mechatronics for Multidisciplinary Teaming”, ASEE Annual Meeting, Nashville, TN, 2003.
[7] Mauer, G., “An Interactive Visual Environment for Scientific Problem Solving”, ASEE Annual Meeting, Nashville, TN, 2003.

Authors

HUGH JACK earned his bachelors degree in electrical engineering, and masters and Ph.D. degrees in mechanical engineering at the University of Western Ontario. He is currently an associate professor at Grand Valley State University and chairs the graduate and manufacturing programs. His research interests include using open source software for industrial control.

ANDY BLAUCH received his B.S. degree in Electrical Engineering from Messiah College, M.S. degree in Electrical and Computer Engineering from Carnegie Mellon University, and Ph.D. degree in Electrical Engineering from the Pennsylvania State University. He is currently an Assistant Professor at Grand Valley State University. Current interests include development of an 68HC11-based general purpose controller and student practice/assessment software.

Appendix A - Source Code for Control Program

```
#include <hc11e9.h>/* 68HC11E9 I/O register defines */
#include <buffalo.h>/* Used for serial communication functions */
#include <interrupt.h>/* Interrupt library functions */
#include <pwm.h>/* PWM library functions */

#define TABLE_SIZE11
int point_master[TABLE_SIZE] = {0,24,95,206,345,500,655,794,905,976,1000};
int point_position[TABLE_SIZE];
int point_time[TABLE_SIZE];
int point_start_time;
int point_index;
int point_current;

int ticks;

void init_table(void)
{
        ticks = 0;
        point_current = 0;
        point_index = TABLE_SIZE;
}

void update_table(int start, int end, unsigned duration_sec)
{
        unsignedi;

        point_time[0] = ticks + 10;
        point_position[0] = start;

        for (i=1; i<TABLE_SIZE; i++) {
                point_time[i] = point_time[0]
                        + (unsigned long)i * duration_sec * 250 / (TABLE_SIZE-1);
                point_position[i] = start + (long int)(end-start)*point_master[i]/1000;
        }
        point_index = 0;
}

void get_setpoint(void)
{
        ticks++;

        if (point_index<TABLE_SIZE) {
                if (point_time[point_index] == ticks) {
                        point_current = point_position[point_index++];
                }
        }
}

int Position = 0;

#define MAX_COUNT 255
void EncoderUpdate(void)
{
        /* Static variables for position calculation */
        static unsigned char state = 0xFF;

        unsigned char new_state;

        /* Read encoder state */
        new_state = (PORTD & (PD5 | PD4)) >> 4;
        /* Update position value */
        if (state!=new_state) {
                switch (state) {
                case 0x00:
                        if (new_state==0x01)
                                Position++;
                        else
                                Position--;
                        break;
                case 0x01:
                        if (new_state==0x03)
                                Position++;
                        else
                                Position--;
                        break;
                case 0x03:
                        if (new_state==0x02)
                                Position++;
                        else
                                Position--;
                        break;
                case 0x02:
                        if (new_state==0x00)
                                Position++;
                        else
                                Position--;
                        break;
                }
                state = new_state;
        }
}

/* Magnnitude values */
#define Cstick_pos     150
#define Cstick_neg     150

int DeadBand(int value)
{
        /* Compensate for deadband */
        if (value>0) {
                if (value>255) value = 255;
                value = Cstick_pos + (unsigned)(value)*(255 - Cstick_pos)/255;
        } else if (value<0) {
                if (value<-255) value = -255;
                value = -Cstick_neg - (unsigned)(-value)*(255 - Cstick_neg)/255;
        }
        return value;
}

void SetOutput(int value)
{
        value = DeadBand(value);

        /* Check sign of output */
        if (value<0) {
                /* Set direction control signals */
                PORTD = (PORTD & ~PD3) | PD2;
                /* Make value positive */
                RefSignal = -value;/* Update PWM value */
        } else {
                /* Set direction control signals */
                PORTD = (PORTD & ~PD2) | PD3;
                RefSignal = value;/* Update PWM value */
        }
}
```

```
int KP=8;

int             e_sum = 0;
#define         T      4       /* define as 4 ms, must divide by 1000 later*/
#define         Ki     1


int     integrate(int e){
        e_sum += e * T;
        return e_sum;
}


/* Interrupt subroutine */
DECLARE_ISR(RTI_ISR);
void RTI_ISR(void)
{
        int Ce,Cc;

        TFLG2 = RTIF;  /* Clear interrupt flag */
        InterruptEnable();/* Unblock interrupts */

        get_setpoint();
        EncoderUpdate();

        Ce = point_current - Position;/* Calculate error */
        Cc = KP * Ce + Ki * integrate(Ce);/* Calculate controller output */
        SetOutput(Cc);
}

int main(void)
{
        char    key;
        int     start, end;

        putstr("Control program with PWM and ADC\n");
        putstr("  Press 'q' to quit\n");

        DDRD |= (DDD2 | DDD3);

        /* Code for initializing PWM */
        PWMInit();     /* Initialize PWM generator */
        PWMEnable();   /* Enable PWM generator */

        /* Code for initializing interrupt */
        PACTL &= ~(RTR1 | RTR0);/* Set RTI for 4.096 msec */
        InterruptDisable();    /* Block interrupts */
        InterruptSetPIV(RTI_ISR,PIV_RTI);/* Set RTI vector */
        TMSK2 |= RTII;                /* Enable RTI interrupt */
        InterruptEnable();     /* Unblock interrupts */

        /* Code for initializing interrupt */
        PACTL &= ~(RTR1 | RTR0);/* Set RTI for 4.096 msec */
        InterruptDisable();    /* Block interrupts */
        InterruptSetPIV(RTI_ISR,PIV_RTI);/* Set RTI vector */
        TMSK2 |= RTII;                /* Enable RTI interrupt */
        InterruptEnable();     /* Unblock interrupts */

        putstr("Profile Generator\n");
        putstr("Q : quit\n");
        putstr("1 : index move -40\n");
        putstr("2 : index move -30\n");
        putstr("3 : index move -20\n");
        putstr("4 : index move -10\n");
        putstr("5 : index move  0\n");
        putstr("6 : index move  10\n");
        putstr("7 : index move  20\n");
        putstr("8 : index move  30\n");
        putstr("9 : index move  40\n");

        init_table();

        do {
                key = getch();
                switch (key) {
                case 0:
                        break;
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9':
                        start = point_current;
                        end = start + (key - '5')*30;
                        update_table(start,end,10);
                        break;
                case 'P':
                case 'p':
                        putstr("   Pos = ");
                        outint16(Position);
                        putch('\n');
                        break;
                case 'R':
                case 'r':
                        putstr("   RefSignal = ");
                        outhex8(RefSignal);
                        putch('\n');
                        break;
                case 'k':
                case 'K':
                        KP*=2;
                        if (KP>128) KP=1;
                        putstr("KP=");
                        outint16(KP);
                        putch('\n');
                        break;
                case 'q':
                case 'Q':
                        key = 'q';
                        break;
                }
        } while (key!='q');

        TMSK2 &= ~RTII;/* Disable RTI interrupt */
        PWMDisable();  /* Disable PWM generator */

        return 0;
}
```