IMDL Final Report

Daniel Brickman

12/4/2001

Table of Contents

Abstract

This report will describe the intimate workings of the autonomous robot Spiderman. It will give an introduction to the problem that the robot was designed to solve and then address the parts of each solution devised to conquer the problems. The integrated systems section will describe the overall systems that interact to make the robot work. It will also detail their interactions. Then, the mobile platform will be discussed along with the motivation for its unique design. Next, the actuation utilized in the robot will be discussed. Then the sensors that feed information to the control system of the robot will be described. After each of these members of hardware are established, the paper will move to the behaviors that were created using the controller and software. After concluding the design experience, the paper will include documentation such as code in the appendix.

Introduction

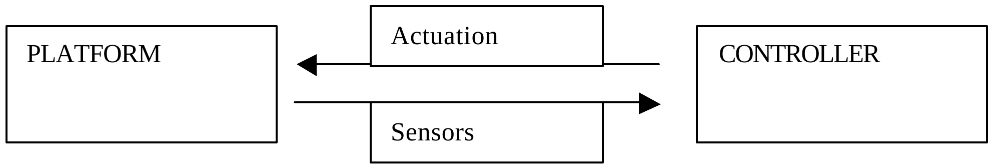
When given an open-ended assignment such as create an autonomous robot of your choice, a student goes through several processes to make a decision. In deciding to make Spiderman, I wanted to break the mold of the simple TJ design. I didn't want to have a two wheeled robot that could to collision avoidance and wall following. Maybe this is a mental problem of mine, but I think I will save that for another report. Ok, so now that I know that I want to do something crazy that has never been doe before and is probably impossible, what should it be? Hmm...What has never been done before and cannot be done. I hate cats so forget that. I should try to leave the ground. OK, how do leave the ground. It took the Wright brothers their entire lives to fly, so forget that. I need to stay attached to hard surfaces. Of course! I will try to climb walls. Now, if you want to do something right, find the person who did it best and try to copy. Alas, now I have a name and an idea. Spiderman.

That was the inspiration for this great big waste of my life. Not that I didn't enjoy making that robot, but it took a little too much of my time. This report will attempt to describe the major parts of Spidey and why they where chosen/designed the way they were. Actually I spent most of my time designed not writing.

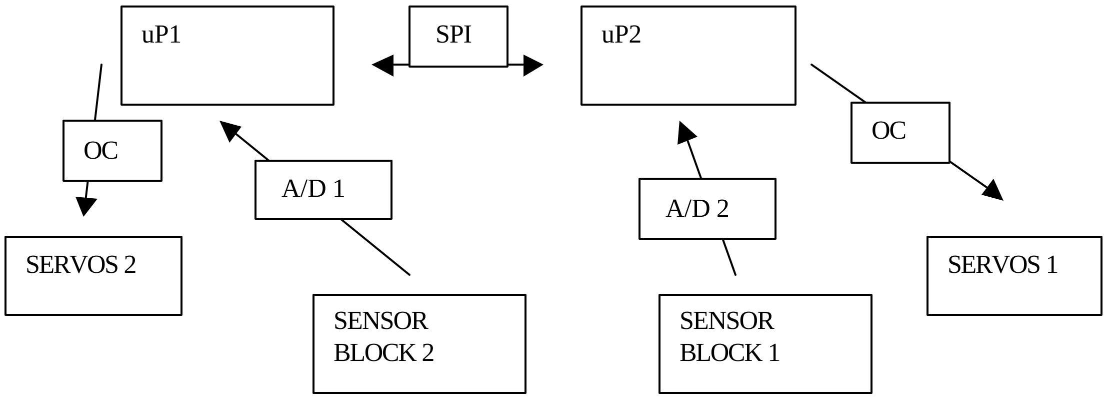
Integrated System

In order to achieve the ultimate goal of climbing walls, a unique integrated system had to be created. This system includes a mobile platform, actuation, sensors, and behaviors created in software. Since each of these will be discussed later individually, this section will describe how these systems interact.

The outer most layer of the system is the platform. It is the way that the robot can interact with its surroundings. The inner most part of the robot is the control hardware. This includes the microprocessor and the software to control it. The interface between the controller and the platform are the sensors. This is the way that the controller can monitor the platform. In order to make changes to the platform, the controller uses actuation. These relationships can be seen in the figure below.

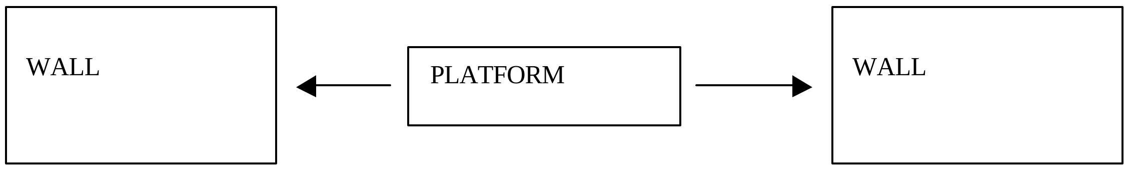


Because of the complexity of the design and the amount of parts it included, I decided to use a dual microprocessor design. The major reason for this was that the design required eight servos. This necessitated eight output compares or the addition of other PWM hardware. In addition, I was planning on using more than eight sensors, which would overflow the A/D port of an HC11. The second microprocessor also allowed the addition of more input capture space and digital I/O lines. The following figure illustrates the interaction of the robot hardware.

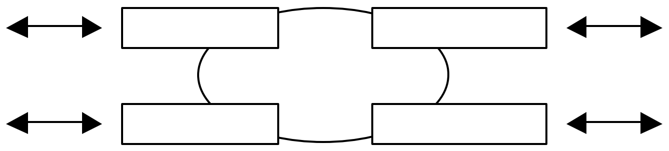


Mobile Platform

In order to complete the task of climbing walls, a unique design for the mobile platform had to be created. The first problem was deciding what type of wall to climb. The solution was in between two walls using a squeeze method. So, how do you create a robot that can squeeze in between two walls? The answer was to create a robot that had independent moving arms. A figure is given below.

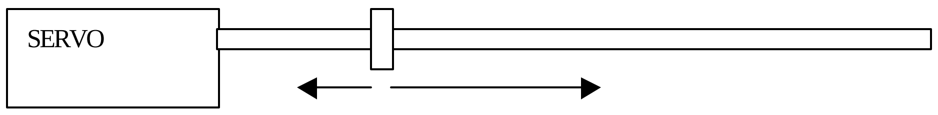


In order to make the body and act of climbing more stable, a design that employed four arms was used. This in conjunction with other sensors would prevent the robot from rolling over in any of the two dimensions. A diagram of the body design is given below.

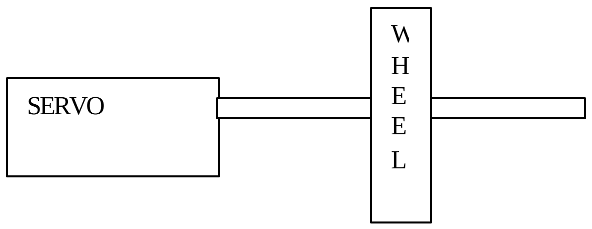


Actuation

There are tow types of actuation required for wall climbing in this particular situation. The first was in the moving are case. The actuation that was needed was linear in the outward direction. Since servos are the cheapest and easiest solution, a way to create a linear motion was required. Incorporating the screw action of a nut and bolt with the spin action of a servo was the answer. If the bolt was fastened to the end of the servo, and the nut held in place, then as the bolt rotated, the nut would travel in a linear fashion. This is shown in the figure below.



The other type of actuation used was another servo used to actually climb the walls. The servo was directly coupled to the drive wheels using a drive shaft. A figure of this setup is given below.



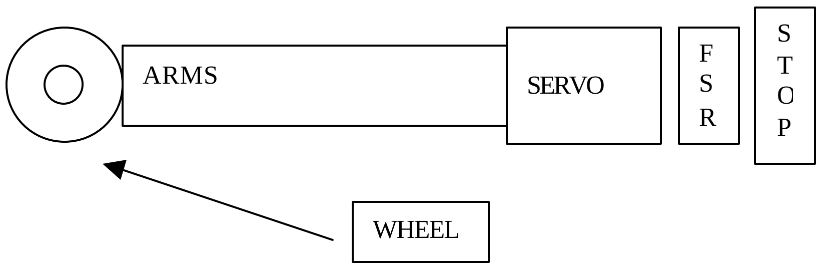
Futaba S3003 Standard	Cirrus CS-80 MG
Torque:	**Torque:**
4.8VDC: 35.5oz-in. (2.56 kg-cm)	4.8VDC: 114.58oz-in.
6.0VDC: 44.4oz-in. (3.2 kg-cm)	6.0VDC: 129.8oz-in.
Speed @ 60 Degrees:	**Speed @ 60 Degrees:**
4.8VDC: 0.25 seconds	4.8VDC: 0.31 seconds
6.0VDC: 0.23 seconds	6.0VDC: 0.25 seconds
Bearing Type:	**Bearing Type:**
None (Case serves as bearing)	Dual Ball Bearings
Case Size:	**Case Size:**
1.59"x 0.78"x 1.42" (40.4 x 19.8 x 36 mm)	1.60" x 1.49" x .79"
Weight: 1.30oz (36.8 g)	**Weight:** 2.01oz (57g)
Wire Length: 12" (Including plug)	**Wire Length:** 12" (Including plug)

Sensors

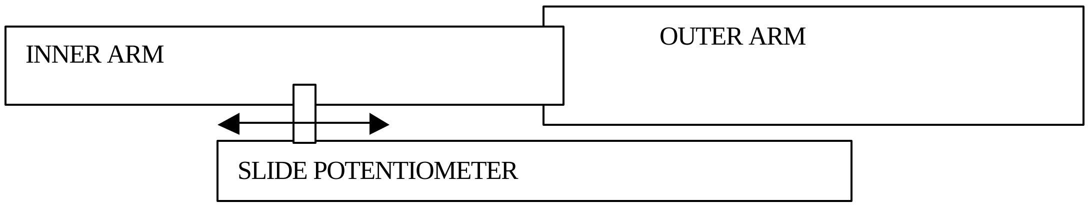
In order for the controller to be able to monitor the operation of the robot at all times, the robot used four types of sensors. Each of them will be described below.

In order to ensure that the robot never flipped, a two-axis accelerometer was used. The ADXL202 is a low cost, low power, complete 2-axis accelerometer with a measurement range of ±2 g. The ADXL202 can measure both dynamic acceleration (e.g., vibration) and static acceleration (e.g., gravity). The outputs are Duty Cycle Modulated (DCM) signals whose duty cycles (ratio of pulsewidth to period) are proportional to the acceleration in each of the 2 sensitive axes. These outputs may be measured directly with a microprocessor counter, requiring no A/D converter or glue logic. The DCM period is adjustable from 0.5 ms to 10 ms via a single resistor (R_{SET}). If an analog output is desired, an analog output proportional to acceleration is available from the X_{FILT} and Y_{FILT} pins, or may be reconstructed by filtering the duty cycle outputs. The bandwidth of the ADXL202 may be set from 0.01 Hz to 6 kHz via capacitors C_X and C_Y. They typical noise floor is 500 µ*g* divided by the square root of Hz allowing signals below 5 mg to be resolved for bandwidths below 60 Hz.

In order to monitor how much force the robot was exerting against the wall during expansion, FSR sensors by Interlink electronics were used. FSR stand for force sensing resistors. The are constructed out of a piezo-resistive material that decreased in resistance proportionally to the amount of force exerted against them. They were implemented using a voltage divider with a 2.2k resistor. The force sensors were used in a threshold manner. After the force broke a threshold, they sent a signal to the controller to stop expansion. The position of the sensors in the robot can be seen in the figure below.

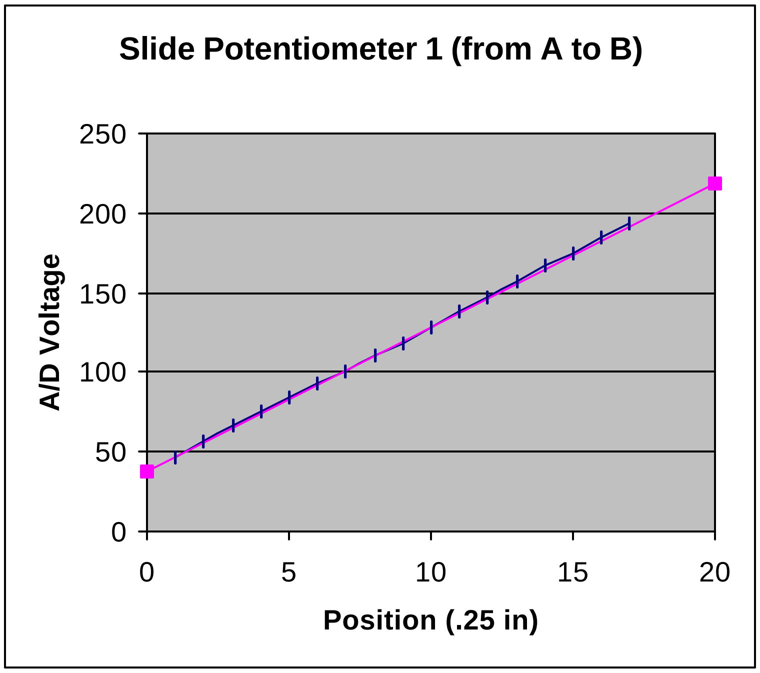


Another sensor that was used was linear slide potentiometers. They were used to track the linear position of the arms. The outside of the pot was attached to the body and the slide was attached to the inner arm. As the arm traveled, it created an internal voltage divider. A figure of the pot is shown below.



Here is some data collected from one of the slide pots.

Position	Value
1	46
2	57
3	66
4	76
5	84
6	93
7	101
8	110
9	118
10	128
11	138
12	147
13	157
14	167
15	175
16	185
17	194
0	38
20	218



Best Fit Line	
Slope	Intercept
9.15	37.73

The last sensor that I used was a CdS cell. The information for these is readily available. I did not use them in a novel way.

Behaviors

Behaviors are implemented on the robot using software and control created from a microprocessor. A Motorola HC11 was used in this case, with the board being an MRC11 from Mekatronix. The MRC11 board, featuring the MC68HC11 microcontroller with up to 64Kbytes of RAM/ROM, provides a versatile package for embedded data acquisition and control systems useful in a wide variety of computer control and measurement applications such as instrumentation, robotics, control, hobby projects, etc. The MRC11's principle features are:

1. Motorola MC68HC11 processor,
2. Two 32KB Memory sockets for either RAM or ROM, 64KB total,
3. 5 Volt regulator,
4. Low voltage inhibit reset circuit,
5. Power on LED,
6. IO Map feature in the upper 32Kbyte Memory using H_MEM_SEL# . External address decode logic can be used to enable or disable H_MEM_SEL# for IO ports assigned in high storage.
7. 60-Pin Male Header serving as a Processor / IO bus.

The behaviors that can be implemented with the hardware previously listed are as follows:

1. Arm expansion and retraction
2. Arm force monitoring
3. Arm position monitoring
4. Body roll and pitch monitoring
5. Wall climbing
6. Line following
7. User interaction using LED's

Conclusion

Well, I guess I have reached the end of a very exciting report on a robot that caused a lot of headaches. What did I learn? Do not do what Id o ever. It is assured to cause roommates to hate you, girlfriends to dump you, and parents to disown you. Did any of that happen to me? Well, again, I think I will save that for another report.

What are m conclusions for the robot? The accelerometer was easy to interface and worked well. The same was true for the force sensors. The slide potentiometers were very inaccurate and had dead spots. I would definitely not use them in future designs. The body design that I put together was very successful for a first draft. I was very pleased with its results. With a few tweaks and successive buildings, it could be perfected and made into a staple of imdl lore.

What did I think of the class? Well, I liked that it was unstructured, but I never quite knew what was due when and what was really important, and what anybody actually wanted from me. Do I think I was successful? Well, that remains to be seen. Still waiting for my report card.

Software

```
///////////////////////////////////////////////////
//          Daniel Brickman           /
//          10-24-2001                          /
//          This program can drive motors 0-3
//          and setup the duty cycles for 4-7
//          forward, stop, and reverse
///////////////////////////////////////////////////

#include <stdio.h>
#include <mil.h>
#include <hc11.h>

   extern void _start(void);

#define DUMMY_ENTRY (void (*)(void))0xFFFF
#define PERIOD 30000
#define STOP 0
#define FASTFORWARD 2700
#define SLOWFORWARD 2900
#define CENTER 3000
#define SLOWREVERSE 3100
#define FASTREVERSE 3300
#define THRESHOLD 170

#pragma interrupt_handler TOC1_isr, TOC2_isr, TOC3_isr, TOC4_isr, TOC5_isr;

   void init_pwm(void);
   void init_var(void);
   void init_ad(void);
   void init_ic(void);
   void wait(void);
   void TOC1_isr(void);
   void TOC2_isr(void);
   void TOC3_isr(void);
   void TOC4_isr(void);
   void TOC5_isr(void);
   //void TIC3_isr(void);
   void motorspeed(void);
   void updatesensor(void);
   void updatesensor1(void);
   void updateic(void);
   void getmotornum(void);
   void menu(void);
   void common(void);

   int duty[8];
   int motornum;
   int motordir[8];
   int speedconv[6];
   int time;
   int i;
   int menuchoice;
   int sensordata[8];
   int firsttime;
   int elapsedtime;
   int isitfirst;

   char clear[]= "\x1b\x5B\x32\x4A\x04";

    void main (void){

      init_var();
      init_pwm();
      init_ad();
      common();
      //init_ic();
   while(1){

      menu();

      if(menuchoice == 0){
         updatesensor();
      }
      else if(menuchoice == 1){
         getmotornum();
         motorspeed();
      }
      else if(menuchoice ==2){
         updateic();
      }
      else{

         int i;
         int j;

         while(1){

            updatesensor1();
            j = 0;

            for(i=0;i<4;i++){

               duty[i]=STOP;

               if(sensordata[i+4] < THRESHOLD ){
                  duty[i] = FASTREVERSE;
                  j++;
               }
            }

            if(j == 0){
               SET_BIT(PORTD, 0x02);
               printf("all stop");
            }

            else
               CLEAR_BIT(PORTD, 0x02);
         }
      }
   }
   //End while(1)
}//End main

 void common(void){

   SET_BIT(DDRD, 0x02);

}

 void init_ad(void){

   SET_BIT(OPTION, 0x80);
   time = 100;
   wait();
}

 void init_pwm(void){
   INTR_OFF();

   SET_BIT(TMSK1, 0x80);                    //Enable OC1

   SET_BIT(TMSK1, 0x40);                    //Enable OC2
   SET_BIT(TCTL1, 0x80);                    //Clear on OC
   CLEAR_BIT(TCTL1, 0x40);
   SET_BIT(OC1M, 0x40);                     //OC1 control
   SET_BIT(OC1D, 0x40);
   SET_BIT(TMSK1, 0x20);                //Enable OC3
   SET_BIT(TCTL1, 0x20);                //Clear on OC
   CLEAR_BIT(TCTL1, 0x10);
   SET_BIT(OC1M, 0x20);                 //OC1 control
   SET_BIT(OC1D, 0x20);

   SET_BIT(TMSK1, 0x10);                //Enable OC4
   SET_BIT(TCTL1, 0x08);                //Clear on OC
   CLEAR_BIT(TCTL1, 0x04);
   SET_BIT(OC1M, 0x10);                 //OC1 control
   SET_BIT(OC1D, 0x10);

   SET_BIT(TMSK1, 0x08);                //Enable OC5
   CLEAR_BIT(PACTL, 0x02);
   SET_BIT(TCTL1, 0x02);                //Clear on OC
   CLEAR_BIT(TCTL1, 0x01);
   SET_BIT(OC1M, 0x08);                 //OC1 control
   SET_BIT(OC1D, 0x08);

   INTR_ON();

}//End init_pwm

 void init_var(void){

   motordir[0] = 0;
   motordir[1] = 0;
   motordir[2] = 0;
   motordir[3] = 0;
   motordir[4] = 0;
   motordir[5] = 0;
   motordir[6] = 0;
   motordir[7] = 0;

   duty[0] = 0;
   duty[1] = 0;
   duty[2] = 0;
   duty[3] = 0;
   duty[4] = 0;
   duty[5] = 0;
   duty[6] = 0;
   duty[7] = 0;

   speedconv[0] = STOP;
   speedconv[1] = FASTFORWARD;
   speedconv[2] = SLOWFORWARD;
   speedconv[3] = CENTER;
   speedconv[4] = SLOWREVERSE;
   speedconv[5] = FASTREVERSE;

   sensordata[0] = 0;
   sensordata[1] = 0;
   sensordata[2] = 0;
   sensordata[3] = 0;
   sensordata[4] = 255;
   sensordata[5] = 255;
   sensordata[6] = 255;
   sensordata[7] = 255;

   isitfirst = 0;
}

 void init_ic(void){

   SET_BIT(TMSK1, 0x01);
   CLEAR_BIT(TCTL2, 0x02);
   SET_BIT(TCTL2, 0x01);
}
```

```
 void wait(void){

   for (i = time; i > 0; i--);
}

 void motorspeed(void){

   printf("What direction for motor %d?\n", motornum);
   printf("(0) Stop\n");
   printf("(1) Fast Forward\n");
   printf("(2) Fast Forward\n");
   printf("(3) Center\n");
   printf("(4) Slow Reverse\n");
   printf("(5) Fast Reverse\n");

   while( (motordir[motornum] < 0x30) || (motordir[motornum] > 0x35))
      motordir[motornum] = getchar();

   motordir[motornum] &= 0x0F;
   duty[motornum] = speedconv[motordir[motornum]];

/*
   switch (motordir[motornum])
   {
      case 1:
         duty[motornum] = FASTFORWARD;
         break;
      case 2:
         duty[motornum] = SLOWFORWARD;
         break;
      case 3:
         duty[motornum] = STOP;
         break;
      case 4:
         duty[motornum] = SLOWREV ERSE;
         break;
      case 5:
         duty[motornum] = FASTREVERSE;
         break;
      default:
         printf("You FUCKED UP!!!!\n");
         break;
   }
*/
}

 void menu(void){

   printf("%s", clear);
   printf("Brickmonster's Main Menu!\n");
   printf("(0) A/D Control\n");
   printf("(1) Motor Control\n");
   printf("(2) Input Capture Control\n");
   printf("(3) Run arm program\n");

   while( (menuchoice < 0x30) || (menuchoice > 0x33))
      menuchoice = getchar();

   menuchoice &= 0x0F;
}

 void updatesensor(void){

   ADCTL = 0x10;

   time = 128;
   wait();

   sensordata[0] = ADR1;
   sensordata[1] = ADR2;
   sensordata[2] = ADR3;
   sensordata[3] = ADR4;

   ADCTL = 0x14;

   time = 128;
   wait();

   sensordata[4] = ADR1;
   sensordata[5] = ADR2;
   sensordata[6] = ADR3;
   sensordata[7] = ADR4;

   printf("Sensor 0: %d, ", sensordata[0]);
   printf("Sensor 1: %d, ", sensordata[1]);
   printf("Sensor 2: %d, ", sensordata[2]);
   printf("Sensor 3: %d,\n", sensordata[3]);
   printf("Sensor 4: %d, ", sensordata[4]);
   printf("Sensor 5: %d, ", sensordata[5]);
   printf("Sensor 6: %d, ", sensordata[6]);
   printf("Sensor 7: %d,\n", sensordata[7]);

   printf("Press any key to continue\n");
   getchar();
}

 void updatesensor1(void){

   ADCTL = 0x10;

   time = 128;
   wait();

   sensordata[0] = ADR1;
   sensordata[1] = ADR2;
   sensordata[2] = ADR3;
   sensordata[3] = ADR4;

   ADCTL = 0x14;

   time = 128;
   wait();

   sensordata[4] = ADR1;
   sensordata[5] = ADR2;
   sensordata[6] = ADR3;
   sensordata[7] = ADR4;

}

 void updateic(void){

   printf("The width is %d\n", elapsedtime);
   printf("Press any key to continue\n");
   getchar();
}

 void getmotornum(void){
   printf("What motor do you want to control?\n");
   printf("Motor 0: %d, ", motordir[0]);
   printf("Motor 1: %d, ", motordir[1]);
   printf("Motor 2: %d, ", motordir[2]);
   printf("Motor 3: %d,\n", motordir[3]);
   printf("Motor 4: %d, ", motordir[4]);
   printf("Motor 5: %d, ", motordir[5]);
   printf("Motor 6: %d, ", motordir[6]);
   printf("Motor 7: %d,\n", motordir[7]);

      /*for (i = 0;i < 8;i++)
      {
            printf("Motor %d:  %d, ", i, motordir[i]);
            if (i == 3)
               printf("\n");
         }*/

      while( (motornum < 0x30) || (motornum > 0x37))
         motornum = getchar();

      motornum &= 0x0F;
   }

    void TOC1_isr(void){

      CLEAR_FLAG(TFLG1, 0x80);

      TOC2 = TOC1 + duty[0];
      TOC3 = TOC1 + duty[1];
      TOC4 = TOC1 + duty[2];
      TOC5 = TOC1 + duty[3];
      TOC1 += PERIOD;

   }//End TOC1

    void TOC2_isr(void){

      CLEAR_FLAG(TFLG1, 0x40);

   }//End TOC2

    void TOC3_isr(void){

      CLEAR_FLAG(TFLG1, 0x20);

   }//End TOC3

    void TOC4_isr(void){

      CLEAR_FLAG(TFLG1, 0x10);

   }//End TOC4

    void TOC5_isr(void){

      CLEAR_FLAG(TFLG1, 0x08);

   }//End TOC5

    /*void TIC3_isr(void){

      CLEAR_FLAG(TFLG1, 0x01);

      if(isitfirst == 0){
         firsttime = TIC3;
         isitfirst = 1;
      }
      else{
         elapsedtime = ( TIC3 - firsttime);
         isitfirst = 0;
      }
   }
*/
#pragma abs_address:0xffd6
/* change the above address if your vector starts elsewhere */

   void (*interrupt_vectors[])(void) =
   {
   /* to cast a constant, say 0xb600, use(void (*)())0xb600 */

   DUMMY_ENTRY,                /* SCI, RS232 protocol */
   DUMMY_ENTRY,                /* SPI, high speed synchronous serial*/
```

```
  DUMMY_ENTRY,              /* Pulse accumulator input edge */
  DUMMY_ENTRY,              /* Pulse accumulator overflow */
  DUMMY_ENTRY,              /* Timer overflow */
  TOC5_isr,                 /* TOC5 */
  TOC4_isr,                 /* TOC4 */
  TOC3_isr,                 /* TOC3 */
  TOC2_isr,                 /* TOC2 */
  TOC1_isr,                 /* TOC1 */
  DUMMY_ENTRY,  /* TIC3 */
  DUMMY_ENTRY,  /* TIC2 */
  DUMMY_ENTRY,  /* TIC1 */
  DUMMY_ENTRY,  /* RTI */
  DUMMY_ENTRY,  /* IRQ */
  DUMMY_ENTRY,  /* XIRQ */
  DUMMY_ENTRY,  /* SWI */
  DUMMY_ENTRY,  /* ILLOP */
  DUMMY_ENTRY,  /* COP */
  DUMMY_ENTRY,  /* CLMON */
  _start      /* RESET */
  };

#pragma end_abs_address
```