C Programming of Microcontrollers for Hobby Robotics

Mehmet Bodur, Asemeh Pousti

Computer Engineering Department, Eastern Mediterranean University
G.Magusa TRNC.

Abstract. This handout supplies the necessary start up information for programming microcontrollers in C language. The examples of the handout is selected on the amateur robotics applications such as line tracking by a mobile platform, on-off, proportional, proportional-integral-derivative control for servomotor control.

Keywords: PIC microcontroller, C for hobby robotics, line tracking, PID controller.

1 Introduction

This article is written for the novice robotics hobbyists who want to have a quick start in microprocessor programming, for the IJRC 2008 Robotics Competition. It is a great opportunity for a hobbyist to enjoy the competition with a home-made robot.

A *microcontroller* (or MCU) [1] is a computer-on-a-chip used to control electronic devices. It is a type of microprocessor emphasizing self-sufficiency and cost-effectiveness, in contrast to a general-purpose microprocessor used in a PC. A typical microcontroller contains all the memory and interfaces needed for a simple application, whereas a general purpose microprocessor requires additional chips to provide the same functions.

The microcontroller applications are mainly categorized into the following fields in an alphabetic listing: Audio; Automotive; Wired communication; Computers and peripherals; Consumer Appliances; Industrial Instrumentation; Imaging and video application; Medical Instrumentation; Military and aerospace instrumentation; Mobile and wireless devices; Motor control; Security Equipments; General Purpose Devices; and Miscellaneous Applications. From these fields, robotics uses the wired and wireless communication, image processing, and motor control extensively.

A microcontroller is a must of a mechatronics device which combines intelligence, mechanical actions, and sensory devices for a goal. Robotics and Hobby Robotics are typical fields of mechatronics. In a robot, microcontrollers are embedded to implement intelligent control algorithms.

A low-end microcontroller can execute most of the robotics control algorithms. Implementation of robotics microcontroller applications requires two main components, a microcontroller and its IC-programming tool, and a convenient programming language to express the algorithms. The main objective of this article is an introduction to these item.

1.1 Why is C preferred over other languages?

Almost 60 percent of the code development in the embedded system industry is performed using C language, although the natural language of a microcontroller is its assembly language. The remaining 40 percent is shared by assembly coding and higher level coding such as using C++, Java, or other object oriented languages. There are many strong reasons for this industrial preference:

1. C is a 'mid-level' coding language, with high-level' features. It supports functions and modules in a well structured form. Yet, it supports all 'low-level' features. It provides access to hardware via pointers in a convenient structure;
2. C is very efficient, popular and well understood programming language;
3. Even desktop developers who have used only Java or C++ can soon understand C syntax;
4. Good, well-proven compilers are available for every embedded processor, including the 8-bit lower end, 16-bit middle-end and 32-bit or higher-end microcontrollers;
5. C programming is common practice for many applications, and it is a subset of C++. Thus, experienced staff are available;
6. Books, training courses, code samples and WWW sites discussing the use of the language are all widely available.

1.3 Well known Microcontroller Families

Intel MCS51, and 8051

Intel is producing MCS51, and 8051, which are popular 8-bit microcontrollers. Typical features of a 8051 are:

- Low-power Idle and Power-down modes.
- Thirty-two input / output lines.
- Internal data (RAM) memory - 256 bytes
- Up to 64 khytes of ROM memory (usually flash)
- Three 16-bit timers / counters
- Nine interrupts (two external) with two priority levels.
- Use Keil IDE for C programming.

For the beginners, Keil IDE and C programming is explained in very fine details in the books "Exploring C for Microcontrollers" [2]. There is a 10 week tutor Programming Embedded Systems by M.J. Pont. and book "Embedded C" by M.J.Pont [3].

Motorola® M68332. Motorola® M68332 is a 32-bit MCU that provides linux operating system and C/C++ programming environment for the embedded system applications. As an example, "Mini RoboMind" MCU board is an ideal platform for beginner, intermediate or advanced amateur robotics with a considerable low price (<$100). The Mini RoboMind is specifically designed for building robots of every kind. It is powerful enough for advanced algorithm development in a small board that fits to most mini robot applications.

The Mini RoboMind contains a MC68332, 512K of Flash, and 32K or 512K of RAM and measures 75 mm x 75 mm. It has A/D, TPU, Port E and Port F headers, an LCD port and 4 SPI ports. The Flash and A/D chips are mounted on the left under the socket for the RAM. The TTL serial port, BDM port and power connector are left to right across the bottom. The LCD port provides two additional chip select pins which allows the LCD port to double as a memory mapped device port.

Eyecon® is another similar controller board for robotics applications. "RoBIOS" (Robot Basic Input Output System) operating system is available in Embedded Robotics by Thomas Bräunl

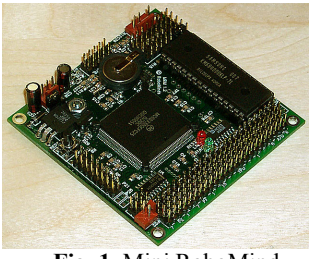


Fig. 1. Mini RoboMind MCU Board.

Microchip PIC18 Family. PIC18 Family processors provide another possibility for robotics applications. PIC18 family is a RISC processor in Harvard Architecture. All PIC18 instructions are 16-bit, and they are in one of the five basic instruction format. The two-operand instructions use W-register (Work Register) for the second operand. All special function registers of PIC18 microcontrollers are in the register range 0xF80 – 0xFFF, which is mapped to Access-Bank. The register address is constructed from BSR (bank-select-register) and 8-bit in-bank address. Indirect addressing is supported through three 16-bit FSRx registers. Access to program memory is provided by Table-read, and table-write instructions through the special function registers. In programming these devices, the main source to be referred is its data sheet [5]. Huge amount of application sheets and sample programs are available in the manufacturers web page "www.microchip.com".

The PIC18 family provides a wide spectrum of microcontroller configuration possibilities. The choice of the correct device requires a comparison of their prices and the availability of the necessary hardware in-chip services. Table 1. may give an opinion about their in-chip facilities.

Table 1. In-chip facilities of well-known PIC18 Family Members.

IC	nr.of pins	program memory	RAM	EEPROM	parallel ports	counter timers	CCP and PWM	serial ports	ADC
18F242	**28 pin**	**16kB 8K-instr**	**768 bytes**	256 bytes	**3 ports**	4	2	2	**5 10-bit**
18F252	**28 pin**	**32kB 16K-instr**	**1536 bytes**	256 bytes	**3 ports**	4	2	2	**5 10-bit**
18F442	**28 pin**	**16kB 8K-instr**	**768 bytes**	256 bytes	**5 ports**	4	2	2	**8 10-bit**
18F452	**28 pin**	**32kB 16K-instr**	**1536 bytes**	256 bytes	**5 ports**	4	2	2	**8 10-bit**

2 Integrated Development Environment and C compilers

For PIC18 family there are 3 main choices of C compilers.

C18. The PIC family manufacturer Microchip Technologies has a C compiler, named C18. The compiler has no student or free version, but it has a 60 day trial version that

contains almost all facilities of its professional version. The download of C18 compiler setup is available in Microchip web page. You can find many sample projects in "Designing Embedded Systems with PIC Microcontrollers" by Tim Wilmshurst [6] .

PICC HiTech compiler. This compiler is an ANSI C compiler, which is an advantage for a novice microcontroller user who knows C, but disadvantage in producing larger codes for the same code compared to C18 and CC8E. Its Lite version is free, but has many restrictions. Many sample projects with PICC are available in the book "Microcontroller Based Applied Digital Control" by Dogan Ibrahim [7].

CC8E. It is a compiler produced by B. Knudsen Data Corporation, which can be accessed from the web page "`http://www.bknd.com/` ". The Compiler Manual is available on web at address "`http://www.bknd.com/cc8e-11.pdf` ". This corporation has also a PIC16 family C compiler, which has proven its code savings by the redesign of a real system originally designed with a 8051 chip. The original 8051 code occupied 32kB, and it contained many low level modules written in assembly, mainly related to a data acquisition system with dynamic memory allocation, data compression, 32-bit arithmetics and more. The CC5X compiler allowed all code to be written in C, and reduced the code to 22kB. At the end of development stage, the renewed and upward compatible system consisted of 5.5 kB code, which is only 3220 instructions, using a PIC16C73. The success is concluded to be a consequence of the efficiency of CC5X in optimizing the code, together with the advanced architecture of PIC16 instruction set over 8051. The free (restricted) version of the compiler is used in CMPE423 Embedded System Design Course, and several Sample Projects available in the course notes.

In the following Sections, we will give microcontroller circuit schematics and C coding examples on the CC8E compiler.

2.1 Integrated Development Environment

An Integrated Development Environment (IDE) provides fast and simple coding and debugging of the microcontrollers. PIC family devices use MPLAB IDE software, distributed free on the Microchip web page. MPLAB supports many other compilers including all C18, PICC, and CC8E compilers. MPLAB has an internal simulator to simulate only the processor. MPLAB-Sim is a very useful tool to measure the time periods of the code loops, and to debug parts of the code. However, it is not sufficient to simulate the overall system together with all peripheral connections. The entire system simulation is managed in a hybrid circuit simulator.

2.2 Circuit Simulators

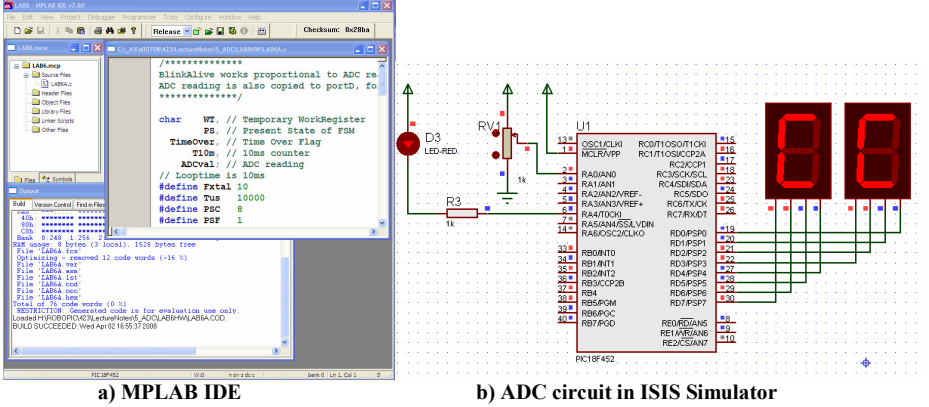
A Circuit Simulator provides the circuits to be tested before their physical implementation. *PROSIS*® is a powerful professional hybrid circuit design tool. It has two parts, *ISIS* for hybrid circuit simulation, and *ARES* for printed circuit board design. ISIS can simulate a controller together with its code. In the following sections, the simulations of the sample circuits will be described in ISIS.

3 Coding Example with ADC

This part is dedicated to four coding examples for PIC18 in C language for CC8E compiler.

3.1 Simple ADC reading and display example

The first example is displaying a potentiometer voltage on a 7-segment Hex display. The circuit is shown in Fig.3, and the C program is listed in App.A.



a) MPLAB IDE **b) ADC circuit in ISIS Simulator**

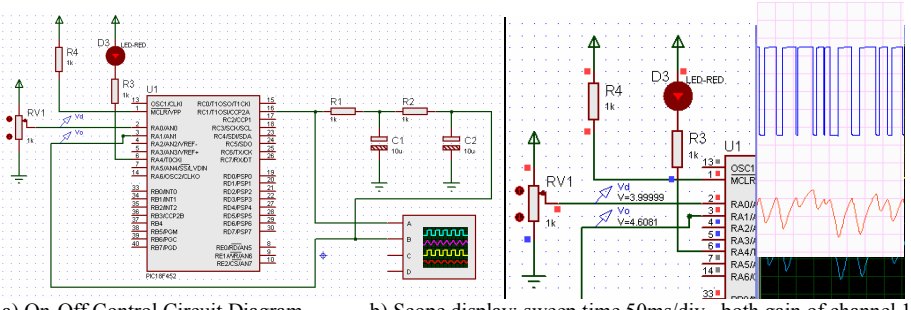
Fig.3 a) Simple ADC Program compiled in MPLAB IDE.
b) PIC18F452 reads analog voltage from RA0, and displays it on RD0-RD7

For simulation, the program shall be coded in MPLAB IDE, as a CC8E project, and shall be compiled to a HEX file (See Fig.3.a). Then, the hex file is linked to the PIC16F452 device in ISIS, and the simulation can be started by clicking the start button. The red (+) and (−) active dots on the potentiometer slides the potentiometer to up and down. The corresponding voltage can be read using a voltage probe.

3.2 An On-Off Control Example

Typical dynamic structure of a robotic component has a second order behavior due to the mass-force relation at the links of the industrial robots, and at the wheels of the

moving robots. A second order system can easily be simulated by RC components R1,C1,R2,C2 as seen in Fig.4.a.



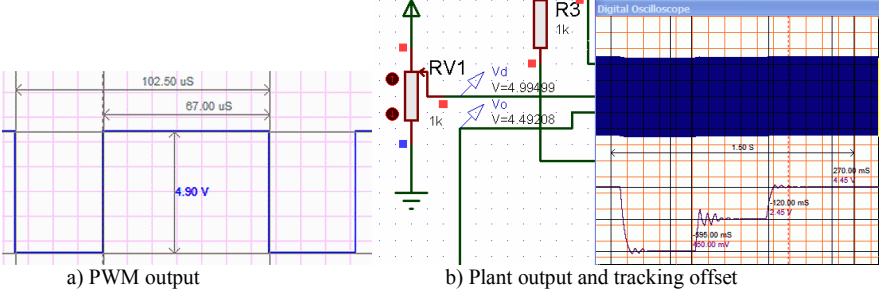
a) On-Off Control Circuit Diagram. b) Scope display: sweep time 50ms/div, both gain of channel 1 (=controller-output) and 2 (=plant-output) are 1V/div.

Fig. 5. On-Off Control of a Second Order Plant made of R1-C1-R2-C2. RV1 sets the desired output voltage, and RC1 gives on-off controller output.

The on-off control of plant output is obtained by converting the voltages from RA0 and RA1 to variable A0 and A1, calculating error e = A0 – A1. Finally RC1 is set to 5V when e is positive, and to 0V when the e is negative. The C code doing this job is given in Appendix B. In Fig.5.b. we observe over 3 volts oscillatory movements of output voltage due to the delay in the second order plant. A proportional control shall stop these deviations as seen in the next section.

3.3 Proportional Control of Second Order Plant

PIC18F452 contains a PWM output to implement analog motor control voltages by chopping the nominal motor voltage. A typical PWM cycle is shown in Fig. 6.a. The PWM cycle time is much higher than the plant time constants, and consequently filtered by the plant to its average value. Thus, the effective voltage applied to the plant changes linearly with the duty cycle of the PWM waveform.



a) PWM output b) Plant output and tracking offset

Fig. 6.a) PWM output with 102.5 µs period and 67 µs duty cycle (=65.3%).
b) Offset between the plant output and the set point is almost 0.5V.

The main drawback of proportional control is a tracking offset as seen in Fig.6.b. The tracking offset can be reduced to zero only when the proportional gain goes to infinity. But, a second order robotic system becomes unstable when the gain is

increased beyond a value. The output voltages at 0.35V, 2.35V and 4.35 V are obtained after setting the desired output voltage RV1 to 0V, 2.5V and 5V. The oscillatory character at the beginning of 2.35 volt section indicates that increasing the gain will put the system into unstable mode.

The tracking error can be reduced to zero by introducing the integral control mode. The unstability caused by the integral control mode can be compensated introducing a derivative control.

3.4 Proportional-Integral-Derivative Control

PID control is applied to the plant using exactly the same circuit, with only improving the controller program by including the derivative and integral terms to the proportional term. The C code is available in Appendix-C. In Fig. 8, we observe that the plant output tracks the set point perfectly with the 10-bit ADC tolerance of about 0.1%. We see also that the stability is improved to an almost critically damped condition. The dark band is the PWM output swinging between 0 and 5V.

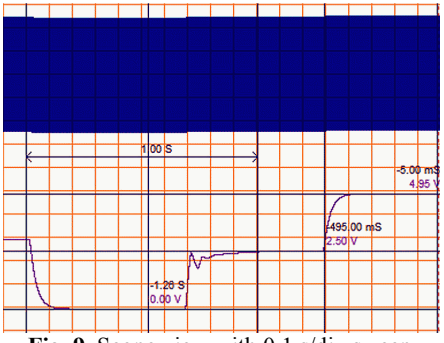


Fig. 9. Scope view with 0.1 s/div sweep setting, and 1V/div vertical setting. The three sections of the waveform are obtained with set point 0V, 2.5V, and 4.95V.

3.5 Line Tracking Robot Control

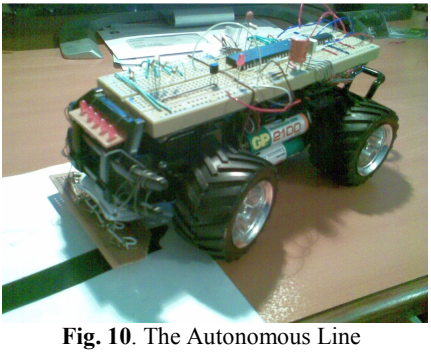


Fig. 10. The Autonomous Line Following Vehicle

The line tracking robot explained in this example was implemented as a design project by senior Computer Engineering students, Yaşar Onur DÜNDAR and Ozan ÇANGO under Dr. Bodur's supervision [8]. The implementation completed in almost two months, and it took around 100hr, including the choice of the toy-car, implementation and test of the sensors, and programming the micro-controller. In the design, a toy-car with two dc-motors is used to move the platform which carries the circuit. The moving robot is made of the following parts:

1- Car with two dc motors. Back-motor drives the car forward and backward. Front motor directs the front wheels to left or to right side.

2- Batteries provide power to the motors, and motor-driver circuit provides control of motor power by the input signals: *forward*, *backward*, *leftturn*, and *rightturn*.

3- The sensor circuit sends infrared-beam, and converts the reflected light intensity into the sensor voltages. Five sensors provide a gradual decision about the deviation of the platform from the black-on-white line that specifies the desired trajectory.

4- The microcontroller which is shown in Fig.11 provides control of the motor drivers to track the desired trajectory.

a) Motor Driver b) Micro Controller c) Optical Sensor d) Sensor Board

Fig. 11. Electrical Components of Line Tracking Control

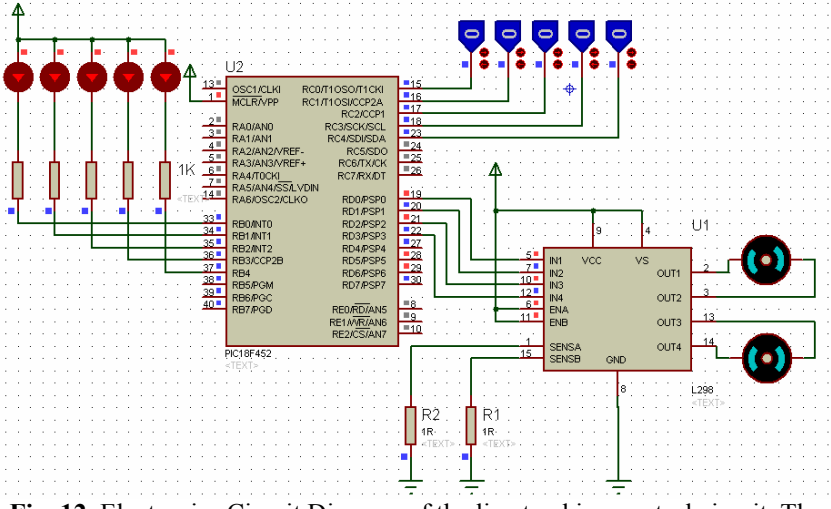


Fig. 12. Electronics Circuit Diagram of the line tracking control circuit. The optical sensors are simulated by logic-input-probes.

The applied control is essentially an on-off control. The control function is a Boolen function of five optical sensor inputs, to decide on four motor control output signals: forward, backward, leftturn, and rightturn. The motor control circuit diagram is shown in Fig. 12. The CC8E code for the controller is displayed at App. D,

4 Conclusion

This paper provides sample C codes for robotics hobbyists, mainly for servo control and optical line tracking purposes. It also shows that amateur robotics can be a fun using today's flash microcontrollers and C compilers.

Acknowledgments. Many thanks to Yaşar Onur DÜNDAR and Ozan ÇANGO who worked on the autonomous line tracking vehicle project.

References

1. Wikipedia
2. Jivan S. Parab, Vinod G. Shelake, Rajanish K. Kamat, Gourish M. Naik, "Exploring C for Microcontrollers. A Hands on Approach", Springer, The Netherlands, 2007.
3. Pont, Michael J. "Embedded C", Addison Wesley - Pearson Education Limited 2002,
4. Thomas Bräunl, "Embedded Robotics, Mobile Robot Design and Applications with Embedded Systems", Second Edition, Springer 2006.
5. Microchip, PIC18FXX2 Data Sheet, High Performance, Enhanced Flash Microcontrollers with 10-Bit A/D. Microchip Technology, 2002
6. Dogan Ibrahim, "Microcontroller Based Applied Digital Control" John Wiley & Sons Ltd, 2006
7. Tim Wilmshurst, "Designing Embedded Systems with PIC Microcontrollers Principles and Applications", Elsevier Ltd, 2007.
8. Yaşar Onur Dundar, Ozan Cango, "Design and Implementation of an Autonomous Line Tracking Robot, BS Graduation Project, EMU-CMPE, Fall-2007.

Appendix-A: Source Code for ADC reading display sample.

In our first example we introduce a CC8E program that reads **RA0** and sets the ToggleAlive period accordingly.

```
// This program sets the ToggleAlive period by a potentiometer.
char    WT, // Temporary WorkRegister
        PS, // Present State of FSM
  TimeOver, // Time Over Flag
      T10m, // 10ms counter
    ADCval,
       MSG; // FSM Message to print
#define Timer0Count (65535-25000+12)
#define TCL (Timer0Count%256)
#define TCH (Timer0Count/256)

// ToggleAlive toggles depending on ADCval.
void ToggleAlive(void){
  --T10m;
  if(T10m ==0 ){ // Blinks at every second
    PORTA.4 ^= 1;
    T10m=ADCval ;
  }
}

void LoopTime(void){
  do{}while( !TMR0IF);
  TMR0IF=0;
  WT = TMR0L +TCL ;
  W = TCH;
  TMR0H = addWFC(TMR0H);
  TMR0L = WT;
  }
```

```
void init(void){
  ADCON1 = 0b10001110;
  ADCON0 = 0b01000001;  // Turn ADC Power On.
  TRISA = 0b11100001;
  PORTA = 0;
  TRISB = 0b00000000; //output
  TRISC = 0b00000000;
  TRISD = 0b00000000; //output
  PORTD = 0;
  TRISE = 0b00000100; //ADC is input
  PORTE = 0;
  T0CON = 0b10001000; // no prescaler
  T10m = 100;
  PS=0;
  }

void ADCRA0(void){
   ADCON1 = 0b01001110;
   ADCON0 = 0b01000101;  // started
   do{}while(ADCON0.2);
   ADCval = ADRESH;
   PORTD=ADCval;}

void main(void){
 init();
 do{
   ADCRA0();
   ToggleAlive();
   LoopTime();
   }while(1);
 }
```

The `main()` procedure contains only the initialization, and a list of processes in an endless loop. This is typical super-loop structure of an embedded program. `ADCRA0()` reads `RA0` into a global char `ADCval` variable, and sends it to `PORTD`. `ToggleAlive()` toggles the `RA4` pin at every `ADCval` * 10ms time period. `LoopTime()` waits until the timer signals that exactly 10ms is passed since the previous call.

You may observe the ADC readings on the display when you set RV1 (by clicking to the active dots) to generate a higher or a lower voltage.

Appendix-B: Source Code for On-Off control .

```
/**************
2008 ( c ) - Mehmet Bodur
On-Off control.
**************/

unsigned char
      WT  , // temp for Timer0
      T10m; // 10ms counter

int16  err, // Error between A0(=set-point) and A1(=output)

        A0, // ADC reading 1
       A0p, // 10ms past ADC0 reading
        A1, // ADC reading 1
       A1p; // 10ms past ADC0 reading

// Timer0 counts 10ms Looptime:
#define Fxtal 10
#define Tus   10000
#define PSC   8
#define PSF   1
#define TCL  (65536-(long)Tus/4/PSF*Fxtal)%256
#define TCH  (65536-(long)Tus/4/PSF*Fxtal)/256

#define ToggleCnt 50

// ADC configuration
#define ADCRA0 0b11000101
#define ADCRA1 0b11001101
#define ADCONF 0b10000100
//   ADCON1 = 0b11000100;
//   ADCON0 = 0b01000101;  // started

void ToggleAlive(void){
// ToggleAlive toggles RA4 at every 500ms.
  --T10m;
  if(T10m ==0 ){ // Blink depends on ADCval
```

```
    PORTA.4 ^= 1;
    T10m=ToggleCnt ; // counts for 50x10ms
  }
}
void LoopTime(void){
// LoopTime waits timeover, and sets Timer0
  do{}while( !TMR0IF);
  TMR0IF=0;
  WT = TMR0L +TCL;
  W = TCH;
  TMR0H = addWFC(TMR0H);
  TMR0L = WT;
  }

void init(void){
// Initialize ADC from RA0 and RA1
  ADCON1 = ADCONF;
  ADCON0 = ADCRA0;      // ADC powered up
  TRISA = 0b11100011;   // ADC input is configured as inp.pin
  PORTA = 0;
  TRISB = 0b00000000;   //output to display ADCval
  TRISC = 0b00000000;
  TRISD = 0b00000000;   //output
  PORTD = 0;
  TRISE = 0b00000100;   //ADC is input
  PORTE = 0;
  T0CON = 0b10001000;   // no prescaler
  T10m  = ToggleCnt ;
// PWM related
//  PR2  = 200 ;    // PWM cycle time
//  CCPR2L = 50;    // PWM duty cycle
//  T2CON  = 0x04 ; // TMR2 on, prescaler=1, postscaler=1;
//  CCP1CON=0x0c; CCP2CON=0x0c; // PWM mode
  }

void ADCA0(void){
   ADCON1  = ADCONF;
   ADCON0  = ADCRA0;  // started
   do{}while(ADCON0.2);
   A0.low8 = ADRESL;
   A0.high8= ADRESH;
   }
void ADCA1(void){
   ADCON1  = ADCONF;
   ADCON0  = ADCRA1;  // started
   do{}while(ADCON0.2);
   A1.low8 = ADRESL;
   A1.high8= ADRESH;
   }


void main(void){
 init();
// This loop passes at every 10ms
 do{ ADCA0();   // read setpoint
     ADCA1();   // read feedback
     err=A0-A1; // error
     PORTB = (char)err;

     if(err>0) PORTC.1=1;
     else PORTC.1=0;
     //PORTC.1 ^=1;

     ToggleAlive();
     LoopTime();
     }while(1);
 }
```

Appendix-C: Source Code for Proportional control .

```
/**************
2008 (c) Mehmet Bodur
Proportional Controller
**************/
#include "math16.h"
unsigned char
      WT  , // temp for Timer0
      T10m; // 10ms counter

int16  err, // Error between A0(=set-point) and A1(=output)
        A0, // ADC reading 1
        A1, // ADC reading 1
        co; // controller output

// Timer0 counts 10ms Looptime:
```

```
#define Fxtal 10
#define Tus   10000
#define PSC   8
#define PSF   1
#define TCL  (65536-(long)Tus/4/PSF*Fxtal)%256
#define TCH  (65536-(long)Tus/4/PSF*Fxtal)/256

#define ToggleCnt 50

// ADC configuration
#define ADCRA0 0b11000101
#define ADCRA1 0b11001101
#define ADCONF 0b10000100

void ToggleAlive(void){
// ToggleAlive toggles RA4 at every 500ms.
  --T10m;
  if(T10m ==0 ){ // Blink depends on ADCval
    PORTA.4 ^= 1;
    T10m=ToggleCnt ; // counts for 50x10ms
  }
}
void LoopTime(void){
// LoopTime waits timeover, and sets Timer0
  do{}while( !TMR0IF);
  TMR0IF=0;
  WT = TMR0L +TCL;
  W = TCH;
  TMR0H = addWFC(TMR0H);
  TMR0L = WT;
  }

void init(void){
// Initialize ADC from RA0 and RA1
  ADCON1 = ADCONF;
  ADCON0 = ADCRA0;      // ADC powered up
  TRISA = 0b11100111;   // ADC input is configured as inp.pin
  PORTA = 0;  TRISB = 0;  TRISC = 0;
  TRISD = 0;  PORTD = 0;
  TRISE = 0;  PORTE = 0;
  T0CON = 0b10001000;   // no prescaler
  T1Om  = ToggleCnt ;
// PWM related
  PR2    = 255 ;  // PWM cycle time
  CCPR2L = 127 ;    // PWM duty cycle
  T2CON  = 0x04 ; // TMR2 on, prescaler=1, postscaler=1;
//  CCP1CON=0x0c;
  CCP2CON=0x0c;   // PWM mode
  }

void ADCA0(void){
   ADCON1  = ADCONF;
   ADCON0  = ADCRA0;  // started
   do{}while(ADCON0.2);
   A0.low8 = ADRESL;
   A0.high8= ADRESH;
   }
void ADCA1(void){
   ADCON1  = ADCONF;
   ADCON0  = ADCRA1;  // started
   do{}while(ADCON0.2);
   A1.low8 = ADRESL;
   A1.high8= ADRESH;
   }


void main(void){
 int16   Kc=10, // x10 Proportional gain
      n100=100; // 100 to implement fractions
 init();
 do{
// This loop passes at every 10ms
   ADCA0();   // read setpoint
   ADCA1();   // read feedback
   err=A0-A1; // error
   co =  err * Kc /10;
   co += 127; // shift to mid dutycyc
   if (co > 254) co = 254; //max duty
   if (co < 1) co = 1;     //min duty
   CCPR2L = co ;   // PWM duty cycle
   PORTB  = err ;  // for debug
   ToggleAlive();
   LoopTime();
   }while(1);
 }
```

Appendix-D: Source Code for PID control .

```
/**************
2008 ( c) Mehmet Bodur
PID control with PIC18F452
   ADC inputs: RA0 set point, RA1 plant output
   Control output: RC1 in PWM format.
**************/
#include "math16.h"
unsigned char
      WT  , // temp for Timer0
      T10m; // 10ms counter

int16  err, // Error between A0(=set-point) and A1(=output)
       erp, // past error
        co, // controller output
        wn, // temp1
      n100, // 100
      i100, // integer of error x100
      d100, // deriv. of error x100
        Kp, // Prop.gain
                  Kd, // Deriv.gain
        Ki, // Int.gain
        A0, // ADC reading 1
       A0p, // 10ms past ADC0 reading
        A1, // ADC reading 1
       A1p; // 10ms past ADC0 reading

// Timer0 counts 10ms Looptime:
#define Fxtal 10
#define Tus   10000
#define PSC   8
#define PSF   1
#define TCL  (65536-(long)Tus/4/PSF*Fxtal)%256
#define TCH  (65536-(long)Tus/4/PSF*Fxtal)/256

#define ToggleCnt 50

// ADC configuration
#define ADCRA0 0b11000101
#define ADCRA1 0b11001101
#define ADCONF 0b10000100

void ToggleAlive(void){
// ToggleAlive toggles RA4 at every 500ms.
  --T10m;
  if(T10m ==0 ){ // Blink depends on ADCval
    PORTA.4 ^= 1;
    T10m=ToggleCnt ; // counts for 50x10ms
  }
}
void LoopTime(void){
// LoopTime waits timeover, and sets Timer0
  do{}while( !TMR0IF);
  TMR0IF=0;
  WT = TMR0L +TCL;
  W = TCH;
  TMR0H = addWFC(TMR0H);
  TMR0L = WT;
  }

void init(void){
// Initialize ADC from RA0 and RA1
  ADCON1 = ADCONF;
  ADCON0 = ADCRA0;      // ADC powered up
  TRISA = 0b11100011;   // ADC input is configured as inp.pin
  PORTA = 0;
  TRISB = 0b00000000;   //output to display ADCval
  TRISC = 0b00000000;
  TRISD = 0b00000000;   //output
  PORTD = 0;
  TRISE = 0b00000100;   //ADC is input
  PORTE = 0;
  T0CON = 0b10001000;   // no prescaler
  T10m  = ToggleCnt ;
// PWM related
  PR2    = 255 ;  // PWM cycle time
  CCPR2L = 127 ;    // PWM duty cycle
  T2CON  = 0x04 ; // TMR2 on, prescaler=1, postscaler=1;
//  CCP1CON=0x0c;
  CCP2CON=0x0c;   // PWM mode
  }

void ADCA0(void){
   A0p = A0;
   ADCON1  = ADCONF;
   ADCON0  = ADCRA0;  // started
   do{}while(ADCON0.2);
```

```
   A0.low8 = ADRESL;
   A0.high8= ADRESH;
   }
void ADCA1(void){
   A1p = A1;
   ADCON1  = ADCONF;
   ADCON0  = ADCRA1;  // started
   do{}while(ADCON0.2);
   A1.low8 = ADRESL;
   A1.high8= ADRESH;
   }


void main(void){
 init();
 Kp = 50;
 Ki = 10;
 Kd = 10;
 n100 = 100;
 do{
// This loop passes at every 10ms
   ADCA0();   // read setpoint
   ADCA1();   // read feedback
   err=A0-A1; // error
   erp=A0p-A1p; // past error
   d100 = err - erp; // derivative error x100

   co =  err * Kp /n100;  // Proportional Control

   co += d100 * Kd /n100; // Derivative Control

   i100 = i100+ err;      // 100x integral of error
   co += i100 * Ki /n100; // Integral Control

   co += 127;
   if (co > 254) co = 254;
   if (co < 1) co = 1;
   CCPR2L = co ;          // PWM duty cycle
   PORTB = err ;
   ToggleAlive();
   LoopTime();
   }while(1);
}
```

Appendix-E: Source Code for Autonomous Line Tracking Vehicle control .

```
   /***************************************
   2008 ( c) Mehmet Bodur
   Line Following Control
   for an autonomous vehicle.
   Boolean output function is obtained
   by using four look-up tables.
   ***************************************/
   const char F[]= { 0, 1, 1, 1, 1, 1, 1, 1,
                     1, 1, 1, 1, 1, 1, 1, 1,
                     1, 1, 1, 1, 1, 1, 1, 1,
                     1, 1, 1, 1, 1, 1, 1, 0 },
             B[]= { 1, 0, 0, 0, 0, 0, 0, 0,
                    0, 0, 0, 0, 0, 0, 0, 0,
                    0, 0, 0, 0, 0, 0, 0, 0,
                    0, 0, 0, 0, 0, 0, 0, 0 },
             R[]= { 0, 1, 1, 1, 0, 1, 0, 1,
                    0, 0, 0, 0, 0, 0, 0, 0,
                    0, 0, 0, 0, 0, 0, 0, 0,
                    0, 0, 0, 0, 0, 0, 0, 0 },
             L[]= { 0, 0, 0, 0, 0, 0, 0, 0,
                    1, 0, 0, 0, 0, 0, 0, 0,
                    1, 0, 0, 0, 1, 0, 0, 0,
                    1, 0, 0, 0, 1, 0, 0, 0 };
   void main(){
         char PC;
         TRISC=0xFF;
         TRISB=0x00;
         TRISD=0x00;
         PORTB=0xFF;
         do {  PC=PORTC & 0b00011111; PC^=0x1F; PORTB=PC;
               if( F[PC]==1 ) PORTD.3=1; else PORTD.3=0;
               if( B[PC]==1 ) PORTD.2=1; else PORTD.2=0;
               if( R[PC]==1 ) PORTD.1=1; else PORTD.1=0;
               if( L[PC]==1 ) PORTD.0=1; else PORTD.0=0;
           }while(1);}
```