CHAPTER 5

# Simple PIC18 Projects

## Chapter Outline



**PIC Microcontroller Projects in C.** http://dx.doi.org/10.1016/B978-0-08-099924-1.00005-8

In this chapter, we will be looking at the design of simple PIC18 microcontroller-based projects, with the idea of becoming familiar with basic interfacing techniques and learning how to use the various microcontroller peripheral registers. We will look at the design of projects using light emitting diodes (LEDs), push-button switches, keyboards, LED arrays, liquid crystal displays (LCDs), sound devices, and so on. We will be developing programs in C language using both the *mikroC Pro for PIC* and

the *MPLAB XC8* compilers. The fully tested and working code will be given for both compilers. All the projects given in this chapter can easily be built on a simple breadboard, but we will be using the low-cost and highly popular development board EasyPIC V7. The required board jumper settings will be given where necessary. We will start with very simple projects and then proceed to more complex ones. It is recommended that the reader moves through the projects in the given order to benefit the most.

The following are provided for each project:

- Project name,
- Description of the project,
- Block diagram of the project (where necessary),
- Circuit diagram of the project,
- Description of the hardware,
- Algorithm description (in PDL),
- Program listing (mikroC pro for PIC and MPLAB XC8),
- Suggestions for further development.

## Project 5.1—Chasing LEDs

### Project Description

In this project, eight LEDs are connected to PORTC of a PIC18F45K22-type microcontroller, and the microcontroller is operated from an 8-MHz crystal. When the power is applied to the microcontroller (or when the microcontroller is reset), the LEDs turn ON alternately in an anticlockwise manner where only one LED is ON at any time. A 1-s delay is used between each output so that the LEDs can be seen turning ON and OFF.

An LED can be connected to a microcontroller output port in two different modes: the *current-sinking* mode and the *current-sourcing* mode.

### Current Sinking

As shown in Figure 5.1, in the current-sinking mode, the anode leg of the LED is connected to the +5-V supply, and the cathode leg is connected to the microcontroller output port through a current limiting resistor.

The voltage drop across an LED varies between 1.4 and 2.5 V, with a typical value of 2 V. The brightness of the LED depends on the current through the LED, and this current can vary between 8 and 16 mA, with a typical value of 10 mA.

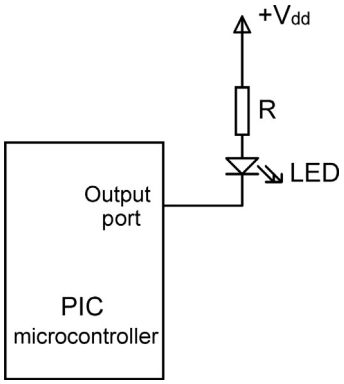


**Figure 5.1: LED Connected in the Current-Sinking Mode.**

The LED is turned ON when the output of the microcontroller is at logic 0 so that current flows through the LED. Assuming that the microcontroller output voltage is about 0.4 V when the output is low, we can calculate the value of the required resistor as follows:

$$R = \frac{V_S - V_{LED} - V_L}{I_{LED}}, \tag{5.1}$$

where

$V_S$ is the supply voltage (5 V),
$V_{LED}$ is the voltage drop across the LED (2 V),
$V_L$ is the maximum output voltage when the output port is low (0.4 V),
$I_{LED}$ is the current through the LED (10 mA).

By substituting the values into Eqn (5.1), we get $R = \frac{5-2-0.4}{10} = 260\ \Omega$. The nearest physical resistor is 270 Ω.

### *Current Sourcing*

As shown in Figure 5.2, in the current-sourcing mode, the anode leg of the LED is connected to the microcontroller output port, and the cathode leg is connected to the ground through a current limiting resistor.

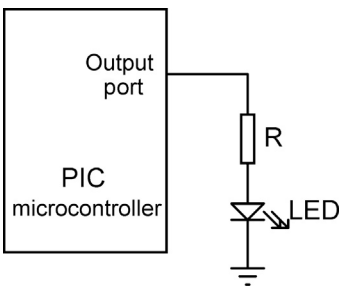


**Figure 5.2: LED Connected in the Current-Sourcing Mode.**

In this mode, the LED is turned ON when the microcontroller output port is at logic 1, that is, +5 V. In practice, the output voltage is about 4.85 V, and the value of the resistor can be determined as follows:

$$R = \frac{V_O - V_{LED}}{I_{LED}}, \tag{5.2}$$

where

$V_O$ is the output voltage of the microcontroller port when at logic 1 (+4.85 V).

Thus, the value of the required resistor is $R = \frac{4.85-2}{10} = 285\ \Omega$. The nearest physical resistor is 290 Ω.

### Project Hardware

The circuit diagram of the project is shown in Figure 5.3. LEDs are connected to PORTC in the current-sourcing mode with eight 290-Ω resistors. An 8-MHz crystal is

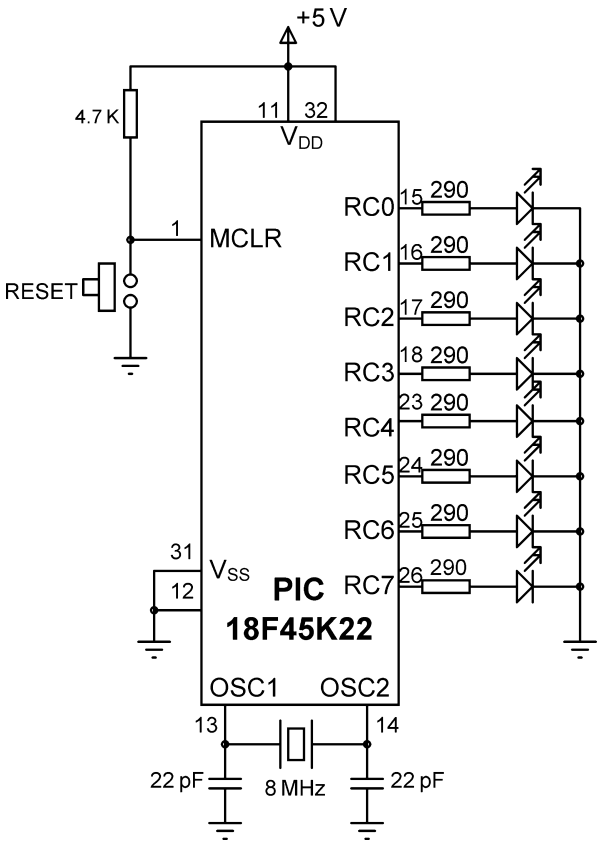


**Figure 5.3: Circuit Diagram of the Project.**

```
BEGIN
    Configure PORTC pins as digital output
    Initialize J = 1
    DO FOREVER
        Set PORTC = J
        Shift left J by 1 digit
        IF J = 0 THEN
            J = 1
        ENDIF
        Wait 1 s
    ENDDO
END
```

**Figure 5.4: PDL of the Project.**

connected between the OSC1 and OSC2 pins of the microcontroller. Also, an external reset push button is connected to the Master Clear (MCLR) input to reset the microcontroller when required.

If you are using the EasyPIC V7 development board, then make sure that the following jumper is configured:

DIP switch SW3: PORTC ON

### *Project PDL*

The PDL of this project is very simple and is given in Figure 5.4.

### *Project Program*

#### *mikroC Pro for PIC*

The mikroC Pro for PIC program is named MIKROC-LED1.C, and the program listing is given in Figure 5.5. At the beginning of the program, PORTC pins are configured as outputs by setting TRISC = 0. Then, an endless **for** loop is formed, and the LEDs are turned ON alternately in an anticlockwise manner to give the chasing effect. The program checks continuously so that when LED 7 is turned ON the next LED to be turned ON is LED 0.

The program is compiled using the mikroC compiler. Project settings should be configured to an 8-MHz clock, XT crystal mode, and WDT OFF. The HEX file (MIKROC-LED1.HEX) should be loaded to the PIC18F45K22 microcontroller using an in-circuit debugger, a programming device, or the EasyPIC V7 development board.

When using the mikroC Pro for PIC compiler, the configuration fuses can be modified from the Edit Project window that is entered by clicking Project → Edit Project.

```
/*************************************************************************
                                CHASING LEDS
                                ============

In this project 8 LEDs are connected to PORTC of a PIC18F45K22 microcontroller and the
microcontroller is operated from an 8 MHz crystal.

The program turns ON the LEDs in an anticlockwise manner with 1 s delay between
each output. The net result is that LEDs seem to be chasing each other.


Author:          Dogan Ibrahim
Date:            August 2013
File:            MIKROC-LED1.C
*************************************************************************/

void main()
{
   unsigned char J = 1;

   ANSELC = 0;                          // Configure PORTC as digital
   TRISC = 0;                           // Configure PORTC as output
   for(;;)                              // Endless loop
   {
       PORTC = J;                       // Send J to PORTC
       Delay_ms(1000);                  // Delay 1 s
       J = J << 1;                      // Shift left J
       if(J == 0) J = 1;                // If last LED, move to first one
   }
}
```

**Figure 5.5: mikroC Pro for PIC Program Listing.**

### *MPLAB XC8*

The MPLAB XC8 program is named XC8-LED1.C, and the program listing is given in Figure 5.6. The program is basically the same as in Figure 5.5, except that here a 1-s delay is created using the basic XC8 __delay_ms function in a loop as it is not possible to create large delays using the __delay_ms function. Function Delay_Seconds creates delay in seconds where the amount of delay is specified by the argument of the function. Note also that the header file <xc.h> must be included at the beginning of the program. Also, the MPLAB IDE must be configured for the PIC18F45K22 type microcontroller and In-Circuit Debugger (ICD) 3 device (hardware tool). The ICD 3 device should be connected to the ICD socket on the EasyPIC V7 development board (top middle part, labeled as EXT ICD). The generated code can then be loaded to the target microcontroller using the MPLAB IDE (see Chapter 5 for more details).

When using the MPLAB XC8 compiler, the configuration fuses can be modified by specifying the "#pragma config" statements at the beginning of the program. It is important to note that different PIC microcontrollers have different sets of configuration fuses. Appendix A gives a list of the valid configuration fuses for the PIC18F45K22 microcontroller.

```
/*************************************************************************
                                CHASING LEDS
                                ============

In this project 8 LEDs are connected to PORTC of a PIC18F45K22 microcontroller and the
microcontroller is operated from an 8 MHz crystal.

The program turns ON the LEDs in an anticlockwise manner with 2 s delay between
each output. The net result is that LEDs seem to be chasing each other.


Author:          Dogan Ibrahim
Date:            August 2013
File:            XC8-LED1.C
*************************************************************************/
#include <xc.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

//
// This function creates seconds delay. The argument specifies the delay time in seconds.
//
void Delay_Seconds(unsigned char s)
{
  unsigned char i,j;

  for(j = 0; j < s; j++)
  {
    for(i = 0; I <  100; i++)__delay_ms(10);
  }
}


void main()
{
   unsigned char J = 1;

   ANSELC = 0;                          // Configure PORTC as digital
   TRISC = 0;                           // Configure PORTC as output

   for(;;)                              // Endless loop
   {
       PORTC = J;                       // Send J to PORTC
       Delay_Seconds(1);                // Delay 1 s
       J = J << 1;                      // Shift left J
       if(J == 0) J = 1;                // If last LED, move to first one
   }
}
```

**Figure 5.6: MPLAB XC8 Program Listing.**

### Further Development

The project can be modified such that the LEDs chase each other in both directions. For example, while moving in an anticlockwise direction, when LED RB7 is ON, the direction can be changed so that the next LED to turn ON is RB6, RB5, and so on.

## Project 5.2—Complex Flashing LED

### Project Description

In this project, one LED is connected to port pin RC0 (PORTC bit 0) of a PIC18F45K22-type microcontroller, and the microcontroller is operated from an 8-MHz crystal. The LED flashes continuously with the following pattern:

3 flashes with 200 ms delay between each flash
2 s delay

### Project Hardware

The circuit diagram of the project is shown in Figure 5.7. An LED is connected to port pin RC0 in the current-sourcing mode with eight 290-Ω resistors. An 8-MHz crystal is connected between the OSC1 and OSC2 pins of the microcontroller. Also, an external reset push button is connected to the MCLR input to reset the microcontroller when required.

If you are using the EasyPIC V7 development board, then make sure that the following jumper is configured:

DIP switch SW3: PORTC ON

### Project PDL

The PDL of this project is very simple and is given in Figure 5.8.

### Project Program

#### *mikroC Pro for PIC*

The mikroC Pro for PIC program is named MIKROC-LED2.C, and the program listing is given in Figure 5.9. Using a *for* loop, the LED is flashed three times with a 200-ms delay between each flash. Then, this process is repeated after 2 s of delay.

#### *MPLAB XC8*

The MPLAB XC8 program is named XC8-LED2.C, and the program listing is given in Figure 5.10. The program uses a function called DelayMs to create milliseconds of delay

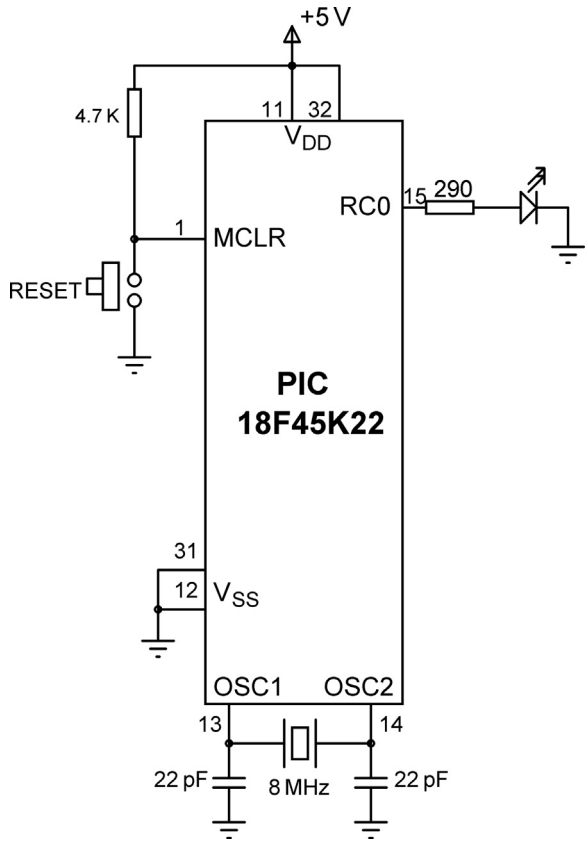


**Figure 5.7: Circuit Diagram of the Project.**

```
BEGIN
    Configure PORTC pins as digital output
    DO FOREVER
        DO 3 times
            Turn ON LED
            Wait 200 ms
            Turn OFF LED
            Wait 200 ms
        ENDDO
        Wait 2 s
    ENDDO
END
```

**Figure 5.8: PDL of the Project.**

```
/*************************************************************************
                        COMPLEX FLASHING LED
                        ====================

In this project an LEDs is connected to port pin RC0 of a PIC18F45K22 microcontroller and the
microcontroller is operated from an 8 MHz crystal.

The program flashes the LED continuously with the following pattern:

                3 flashes with 200 ms delay between each flash
                2 s delay

Author:      Dogan Ibrahim
Date:        August 2013
File:        MIKROC-LED2.C
*************************************************************************/

void main()
{
   unsigned char i;

   ANSELC = 0;                          // Configure PORTC as digital
   TRISC = 0;                           // Configure PORTC as output
   for(;;)                              // Endless loop
   {
     for(i = 0; i < 3; i++)             // Do 3 times
     {
        PORTC.RC0 = 1;                  // LED ON
        Delay_ms(200);                  // 200 ms delay
        PORTC.RC0 = 0;                  // LED OFF
        Delay_ms(200);                  // 200 ms delay
     }
     Delay_ms(2000);                    // 2 s delay
   }
}
```

**Figure 5.9: mikroC Pro for PIC Program Listing.**

from 1 to 65535 ms (maximum value of an unsigned integer) where the required delay is passed in the function argument.

## *Project 5.3—Random Flashing LEDs*

### *Project Description*

In this project, eight LEDs are connected to PORTC of a PIC18F45K22-type microcontroller, and the microcontroller is operated from an 8-MHz crystal. An integer random number is generated between 1 and 255 every second, and the LEDs are turned ON to indicate this number in binary. The net result is that the LEDs flash in a random fashion, and it is interesting to watch them flashing.

```
/*********************************************************************************
                        COMPLEX FLASHING LED
                        ====================

In this project an LEDs is connected to port pin RC0 of a PIC18F45K22 microcontroller and the
the microcontroller is operated from an 8 MHz crystal.

The program flashes the LED continuously with the following pattern:

   3 flashes with 200 ms delay between each flash
   2 s delay

Author:      Dogan Ibrahim
Date:        August 2013
File:        XC8-LED2.C
*********************************************************************************/
#include <xc.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

//
// This function creates milliseconds delay. The argument specifies the delay time.
// The delay can be 1 to 65535 ms
//
void DelayMs(unsigned int ms)
{
   unsigned int j;

   for(j = 0; j < ms; j++)__delay_ms(1);
}


void main()
{
    unsigned char i;

    ANSELC = 0;                          // Configure PORTC as digital
    TRISC = 0;                           // Configure PORTC as output
    for(;;)                              // Endless loop
    {
      for(i = 0; i < 3; i++)             // Do 3 times
      {
         PORTCbits.RC0 = 1;              // LED ON
         DelayMs(200);                   // 200 ms delay
         PORTCbits.RC0 = 0;              // LED OFF
         DelayMs(200);                   // 200 ms delay
      }
      DelayMs(2000);
    }
}
```

**Figure 5.10: MPLAB XC8 Program Listing.**

### Project Hardware

The circuit diagram of the project is as shown in Figure 5.3.

If you are using the EasyPIC V7 development board, then make sure that the following jumper is configured:

DIP switch SW3: PORTC ON

### Project PDL

The PDL of this project is very simple and is given in Figure 5.11.

### Project Program

#### *mikroC Pro for PIC*

The mikroC Pro for PIC program is named MIKROC-LED3.C and the program listing is given in Figure 5.12. The random number is generated using the built-in library function rand. The function should be initialized once by calling function srand with an integer argument before it is used (any integer number can be used). Then, every time rand is called, it generates a pseudorandom number between 0 and 32,767. To generate a number between 1 and 255, we can divide the generated number by 128. Although the generated number is not exactly random, it is good enough for our application.

The random number seed is initialized by calling function srand with integer 10. Then, an endless *for* loop is established, a random number is generated in variable *p*, divided by 128, and sent to PORTC. This process is repeated with a 1-s delay between each output.

#### *MPLAB XC8*

The MPLAB XC8 program is named XC8-LED3.C, and the program listing is given in Figure 5.13. As in Figure 5.6, the required 1-s delay is created using a function called Delay_Seconds. Random numbers are generated as in the mikroC Pro for PIC version of the program. Note that the header file <stdlib.h> must be included at the beginning of the program.

```
BEGIN
    Configure PORTC pins as digital output
    Initialize random number seed
    DO FOREVER
        Get a random number between 1 and 255
        Send the random number to PORTC
        Wait 1 s
    ENDDO
END
```

**Figure 5.11: PDL of the Project.**

```
/*************************************************************************
                        RANDOM FLASHING LEDS
                        ====================

In this project 8 LEDs are connected to PORTC of a PIC18F45K22 microcontroller and the
microcontroller is operated from an 8 MHz crystal.

The program uses a pseudorandom number generator to generate a number between 0 and
32767. This number is then divided by 128 to limit it between 1 and 255. The resultant number
is sent to PORTC of the microcontroller. This process is repeated every second.


Author:       Dogan Ibrahim
Date:         August 2013
File:         MIKROC-LED3.C
*************************************************************************/

void main()
{
   unsigned int p;

   ANSELC = 0;                          // Configure PORTC as digital
   TRISC = 0;                           // Configure PORTC as output
   srand(10);                           // Initialize random number seed

   for(;;)                              // Endless loop
   {
     p = rand();                        // Generate a random number
     p = p / 128;                       // Number between 1 and 255
     PORTC = p;                         // Send to PORTC
     Delay_ms(1000);                    // 1 s delay
   }
}
```

**Figure 5.12: mikroC Pro for PIC Program Listing.**

## *Project 5.4—Logic Probe*

### *Project Description*

This project is a simple logic probe. A logic probe is used to indicate the logic status of an unknown digital signal. In a typical application, a test lead (probe) is used to detect the unknown signal, and two different color LEDs are used to indicate the logic status. If, for example, the signal is logic 0, then the RED color LED is turned ON. If on the other hand the signal is logic 1, then the GREEN LED is turned ON.

### *Project Hardware*

The circuit diagram of the project is as shown in Figure 5.14. Port pin RC0 is used as the probe input. Port pins RB6 and RB7 are connected to RED LED and GREEN LED, respectively.

```
/*************************************************************************
                        RANDOM FLASHING LEDS
                        ====================

In this project 8 LEDs are connected to PORTC of a PIC18F45K22 microcontroller and the
microcontroller is operated from an 8 MHz crystal.

The program uses a pseudorandom number generator to generate a number between 0 and
32767. This number is then divided by 128 to limit it between 1 and 255. The resultant number
is sent to PORTC of the microcontroller. This process is repeated every second.


Author:      Dogan Ibrahim
Date:        August 2013
File:        XC8-LED3.C
*************************************************************************/
#include <xc.h>
#include <stdlib.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

//
// This function creates seconds delay. The argument specifies the delay time in seconds.
//
void Delay_Seconds(unsigned char s)
{
  unsigned char i,j;

  for(j = 0; j < s; j++)
  {
    for(i = 0; i <  100; i++)__delay_ms(10);
  }
}


void main()
{
   unsigned int p;

   ANSELC = 0;                          // Configure PORTC as digital
   TRISC = 0;                           // Configure PORTC as output
   srand(10);                           // Initialize random number seed

   for(;;)                              // Endless loop
   {
     p = rand();                        // Generate a random number
     p = p/128;                         // Number between 1 and 255
     PORTC = p;                         // Send to PORTC
     Delay_Seconds(1);                  // 1 s delay
   }
}
```

**Figure 5.13: MPLAB XC8 Program Listing.**

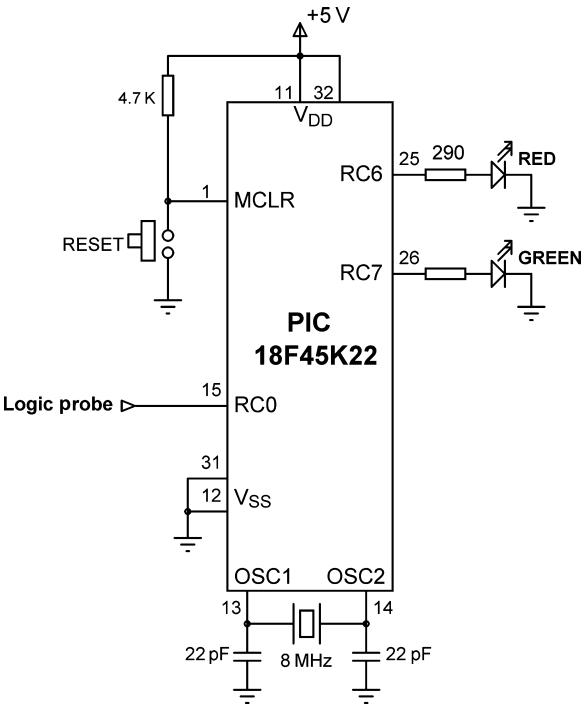


**Figure 5.14: Circuit Diagram of the Project.**

If you are using the EasyPIC V7 development board, then make sure that the following jumper is configured:

DIP switch SW3: PORTC ON

### *Project PDL*

The PDL of this project is very simple and is given in Figure 5.15.

```
BEGIN
        Configure RC0 as digital input
        Configure RC6, RC7 as digital outputs
        DO FOREVER
                IF RC0 is logic 0 THEN
                        Turn OFF GREEN LED
                        Turn on RED LED
                ELSE
                        Turn OFF RED LED
                        Turn ON GREEN LED
                ENDIF
        ENDDO
END
```

**Figure 5.15: PDL of the Project.**

## Project Program

### mikroC Pro for PIC

The mikroC Pro for PIC program is named MIKROC-LED4.C, and the program listing is given in Figure 5.16. The operation of the program is very simple. An endless loop is established using a *for* statement and inside this loop port pin RC0 is checked. If RC0 is at logic 0, then the RED LED is turned ON; otherwise, the GREEN LED is turned ON. Note

```
/*************************************************************************
                                LOGIC PROBE
                                ===========

This is a logic probe project. In this project 2 colored LEDs are connected to PORTC pins
RC6 (RED) and RC7 (GREEN). In addition, RC0 is used as the probe input.

If the logic probe is at logic 0 then the RED LED is turned ON. Otherwise, the GREEN LED is
turned ON.

Author:  Dogan Ibrahim
Date:    August 2013
File:    MIKROC-LED4.C
*************************************************************************/
#define PROBE PORTC.RC0
#define RED_LED PORTC.RC6
#define GREEN_LED PORTC.RC7

void main()
{
   ANSELC = 0;                          // Configure PORTC as digital
   TRISC0_bit = 1;                      // Configure RC0 as input
   TRISC6_bit = 0;                      // Configure RC6 as output
   TRISC7_bit = 0;                      // Configure RC7 as output

   for(;;)                              // Endless loop
   {
     if(PROBE == 0)                     // If the signal is LOW
     {
      GREEN_LED = 0;                    // Turn OFF GREEN LED
      RED_LED = 1;                      // Torn ON RED LED
     }
     else
     {
      RED_LED = 0;                      // Turn OFF RED LED
      GREEN_LED = 1;                    // Turn ON GREEN LED
     }
   }
}
```

**Figure 5.16: mikroC Pro for PIC Program Listing.**

in the program that the individual PORTC pins are configured as input or outputs. We could instead set TRISC = 1 to configure RC0 as the input and others (RC1:RC7) as outputs.

*MPLAB XC8*

The MPLAB XC8 program is named XC8-LED4.C, and the program listing is given in Figure 5.17.

### Further Development

The problem with this logic probe is that one of the LEDs is always ON even if the probe is not connected to any digital signal, or if the signal is at high-impedance state (i.e. tristate). We can develop the project further so that the high-impedance state can also be detected, and none of the LEDs are turned ON.

Figure 5.18 shows the modified circuit diagram. Note here that a transistor (BC108) is used at the front end of the circuit. The operation of the circuit is as follows.

The transistor is configured as an emitter–follower stage with the base controlled from output pin RC4 of the microcontroller through a 100-K resistor. The external signal is also applied to the base of the transistor through a 10-K resistor. The emitter of the transistor is connected to input pin RC3 of the microcontroller through a 100-K resistor. Pin RC4 applies logic levels to the base of the transistor, and then pin RC3 determines the state of the external signal (probe) as shown in Table 5.1. For example, if after setting RC4 = 1, we detect RC3 = 1 and also after setting RC4 = 0 we again detect RC3 = 1 then the probe must be at logic 1.

Figure 5.19 shows the mikroC Pro for the PIC program listing of the modified project. The program is named MIKROC-LED4-1.C. At the beginning of the program, symbols are given to the used I/O pins. Then, these I/O pins are configured as either inputs or outputs. The program executes in an endless *for* loop where the logic in Table 5.1 is implemented. A small delay is used (1 ms, although a few microseconds should be enough) after the transistor base signal is changed to allow the transistor to settle down.

Figure 5.20 shows the equivalent MPLAB XC8 program named XC8-LED4-1.C.

## Project 5.5—LED Dice

### Project Description

This is a simple dice project based on LEDs, a push-button switch, and a PIC18F45K22 microcontroller operating with an 8-MHz crystal. The block diagram of the project is shown in Figure 5.21.

```
/*************************************************************************
                                LOGIC PROBE
                                ===========

This is a logic probe project. In this project 2 colored LEDs are connected to PORTC pins RC6
(RED) and RC7 (GREEN). In addition, RC0 is used as the probe input.

If the logic probe is at logic 0 then the RED LED is turned ON. Otherwise, the GREEN LED is
turned ON.

Author:     Dogan Ibrahim
Date:       August 2013
File:       XC8-LED4.C
*************************************************************************/
#include <xc.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

#define PROBE PORTCbits.RC0
#define RED_LED PORTCbits.RC6
#define GREEN_LED PORTCbits.RC7


void main()
{
   ANSELC = 0;                          // Configure PORTC as digital
   TRISCbits.RC0 = 1;                   // Configure RC0 as input
   TRISCbits.RC6 = 0;                   // Configure RC6 as output
   TRISCbits.RC7 = 0;                   // Configure RC7 as output

   for(;;)                              // Endless loop
   {
     if(PROBE == 0)                     // If the signal is LOW
     {
      GREEN_LED = 0;                     // Turn OFF GREEN LED
      RED_LED = 1;                      // Turn ON RED LED
     }
     else
     {
      RED_LED = 0;                      // Turn OFF RED LED
      GREEN_LED = 1;                    // Turn ON GREEN LED
     }
   }
}
```

**Figure 5.17: MPLAB XC8 Program Listing.**

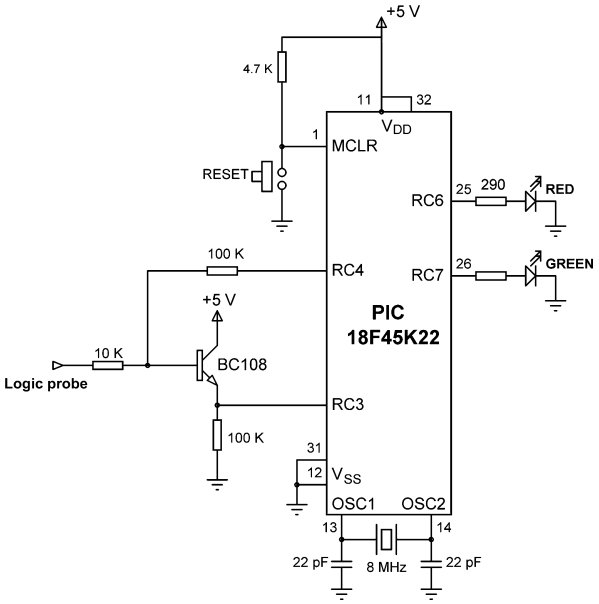


**Figure 5.18: Modified Circuit Diagram.**

As shown in Figure 5.22, the LEDs are organized such that when they turn ON, they indicate the numbers as in a real dice. Operation of the project is as follows: Normally, the LEDs are all OFF to indicate that the system is ready to generate a new dice number. Pressing the switch generates a random dice number between 1 and 6 and displays on the LEDs for 3 s. After 3 s, the LEDs turn OFF again.

### *Project Hardware*

The circuit diagram of the project is shown in Figure 5.23. Seven LEDs representing the faces of a dice are connected to PORTC of a PIC18F45K22 microcontroller in current-sourcing

**Table 5.1: Applied and Detected Logic Levels for Figure 5.18**

| Probe State | Applied to RC4 | Detected at RC3 |
|---|---|---|
| Probe at high impedance | 1 | 1 |
| | 0 | 0 |
| Probe at logic 1 | 1 | 1 |
| | 0 | 1 |
| Probe at logic 0 | 1 | 0 |
| | 0 | 0 |

```
/*************************************************************************
                                LOGIC PROBE
                                ===========

This is a logic probe project. In this project 2 colored LEDs are connected to PORTC  pins RC6
(RED) and RC7 (GREEN). A transistor is used at the front end of the project. Pin RC4 is
connected to the base of the transistor. The emitter of the transistor is connected to pin RC3
of the microcontroller. Table 7.1 in the text explains how the logic state of the probe signal is
detected.

If the logic probe is at logic 0 then the RED LED is turned ON. If the logic level is 1 the GREEN
LED is turned ON. If the probe is not connected to any logic signal or if the signal is at
high-impedance state then none of the LEDs turn on.


Author:    Dogan Ibrahim
Date:      August 2013
File:      MIKROC-LED4-1.C
*************************************************************************/
#define RED_LED PORTC.RC6
#define GREEN_LED PORTC.RC7
#define TSTOUT PORTC.RC4
#define TSTIN PORTC.RC3

void main()
{
   ANSELC = 0;                          // Configure PORTC as digital
   TRISC6_bit = 0;                      // Configure RC6 as output
   TRISC7_bit = 0;                      // Configure RC7 as output
   TRISC4_bit = 0;                      // Configure RC4 as output
   TRISC3_bit = 1;                      // Configure RC3 as input

   for(;;)                              // Endless loop
   {
     TSTOUT = 1;                        // Set RC4 = 1
     Delay_ms(1);                       // Small delay
     if(TSTIN == 1)                      // Check RC3. If RC3 = 1
     {
       TSTOUT = 0;                      // Set RC4 = 0
       Delay_ms(1);                     // Small delay
       if(TSTIN == 0)                   // Check RC3. If RC3 = 0
       {
         GREEN_LED = 0;                 // High-impedance state
         RED_LED = 0;                   // High-impedance state
       }
       else
       {
         RED_LED = 0;
         GREEN_LED = 1;                 // Probe at logic 1
       }
     }
     else
     {
       TSTOUT = 0;
       Delay_ms(1);                     // Small delay
       if(TSTIN == 0)
       {
         GREEN_LED = 0;
         RED_LED = 1;                   // Probe at logic 0
       }
     }
   }
}
```

**Figure 5.19: mikroC Pro for PIC Program Listing.**

```
/*********************************************************************************
                                LOGIC PROBE
                                ===========

This is a logic probe project. In this project 2 colored LEDs are connected to PORTC pins RC6
(RED) and RC7 (GREEN). A transistor is used at the front end of the project. Pin RC4 is
connected to the base of the transistor. The emitter of the transistor is connected to pin RC3
of the microcontroller. Table 7.1 in the text explains how the logic state of the probe signal is
detected.

If the logic probe is at logic 0 then the RED LED is turned ON. If the logic level is 1 the GREEN
LED is turned ON. If the probe is not connected to any logic signal or if the signal is at
high-impedance state then none of the LEDs turn on.


Author:    Dogan Ibrahim
Date:      August 2013
File:      XC8-LED4-1.C
*********************************************************************************/

#include <xc.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

#define RED_LED PORTCbits.RC6
#define GREEN_LED PORTCbits.RC7
#define TSTOUT PORTCbits.RC4
#define TSTIN PORTCbits.RC3


void main()
{
   ANSELC = 0;                    // Configure PORTC as digital
   TRISCbits.RC6 = 0;             // Configure RC6 as output
   TRISCbits.RC7 = 0;             // Configure RC7 as output
   TRISCbits.RC4 = 0;             // Configure RC4 as output
   TRISCbits.RC3 = 1;             // Configure RC3 as input

   for(;;)                        // Endless loop
   {
     TSTOUT = 1;                  // Set RC4 = 1
     __delay_ms(1);               // Small delay
     if(TSTIN == 1)               // Check RC3. If RC3 = 1
     {
       TSTOUT = 0;                // Set RC4 = 0
       __delay_ms(1);             // Small delay
       if(TSTIN == 0)             // Check RC3. If RC3 = 0
       {
         GREEN_LED = 0;           // High-impedance state
         RED_LED = 0;
       }
```

**Figure 5.20: MPLAB XC8 Program Listing.**

```
        else
        {
         RED_LED = 0;
         GREEN_LED = 1;                  // Probe at logic 1
        }
       }
       else
       {
        TSTOUT = 0;                      // Set RC4 = 0
        __delay_ms(1);                   // Small delay
        if(TSTIN == 0)                   // Check RC3. If RC3 = 0
        {
         GREEN_LED = 0;
         RED_LED = 1;                    // Probe at logic 0
        }
       }
      }
}
```

**Figure 5.20**
cont'd

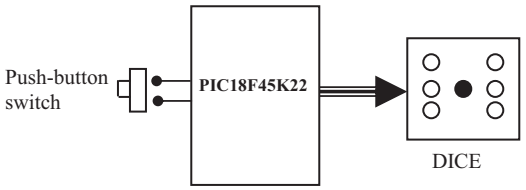


**Figure 5.21: Block Diagram of the Project.**

mode using 290-Ω current limiting resistors. A push-button switch is connected to bit 0 of PORTB (RB0) using a pull-up resistor. The microcontroller is operated from an 8-MHz crystal connected between pins OSC1 and OSC2.

If you are using the EasyPIC V7 development board, then make sure that the following jumpers are configured. Push-button switch RB0 on the board can be pressed to generate a dice number:

DIP switch SW3: PORTC ON
Jumper J17 (Button Press Level): GND

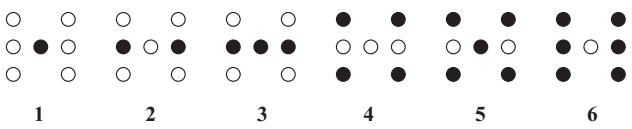


**Figure 5.22: LED Dice.**

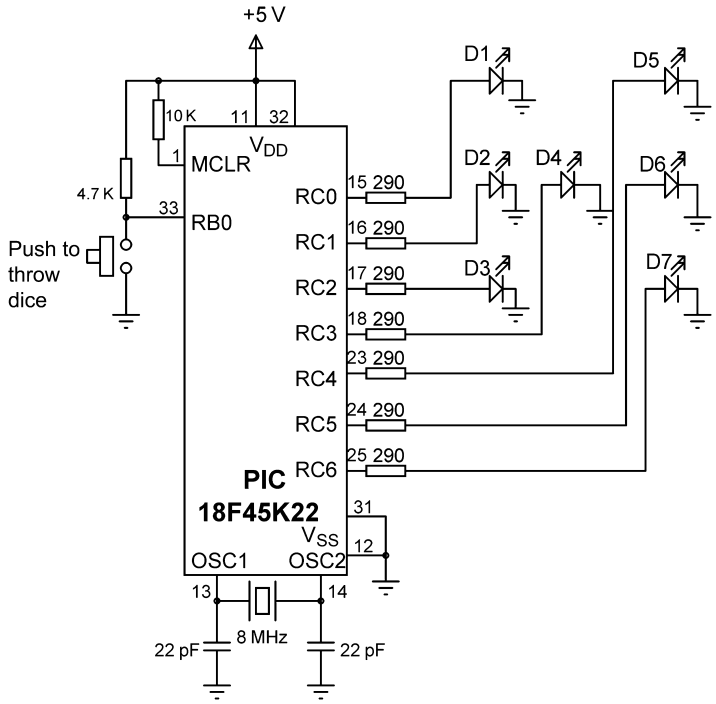


**Figure 5.23: Circuit Diagram of the Project.**

### *Project PDL*

The operation of the project is described in the PDL given in Figure 5.24. At the beginning of the program, PORTC pins are configured as outputs, and bit 0 of PORTB (RB0) is configured as input. The program then executes in a loop continuously and increments a variable between 1 and 6. The state of the push-button switch is checked, and when the switch is pressed (switch output at logic 0), the current number is sent to the LEDs. A simple array is used to find out the LEDs to be turned ON corresponding to the generated dice number.

Table 5.2 gives the relationship between a dice number and the corresponding LEDs to be turned ON to imitate the faces of a real dice. For example, to display number 1 (i.e. only the middle LED is ON), we have to turn on D4. Similarly, to display number 4, we have to turn ON D1, D3, D5, and D7.

The relationship between the required number and the data to be sent to PORTC to turn on the correct LEDs is given in Table 5.3. For example, to display dice number 2, we have to send hexadecimal 0x22 to PORTC. Similarly, to display number 5, we have to send hexadecimal 0x5D to PORTC and so on.

```
BEGIN
        Create DICE table
        Configure PORTC as outputs
        Configure RB0 as input
        Set J = 1
        DO FOREVER
                IF button pressed THEN
                        Get LED pattern from DICE table
                        Turn ON required LEDs
                        Wait 3 s
                        Set J = 0
                        Turn OFF all LEDs
                ENDIF
                Increment J
                IF J = 7 THEN
                        Set J = 1
                ENDIF
        ENDDO
END
```

**Figure 5.24: PDL of the Project.**

**Table 5.2: Dice Number and LEDs to be Turned ON**

| Required Number | LEDs to be Turned on |
|---|---|
| 1 | D4 |
| 2 | D2, D6 |
| 3 | D2, D4, D6 |
| 4 | D1, D3, D5, D7 |
| 5 | D1, D3, D4, D5, D7 |
| 6 | D1, D2, D3, D5, D6, D7 |

## *Project Program*

### *mikroC Pro for PIC*

The mikroC Pro for PIC program is called MIKROC-LED5.C and the program listing is given in Figure 5.25. At the beginning of the program, **Switch** is defined as bit 0 of PORTB, and **Pressed** is defined as 0. The relationship between the dice numbers and the LEDs to be turned on are stored in an array called **DICE**. Variable **J** is used as the dice

**Table 5.3: Required Number and PORTC Data**

| Required Number | PORTC Data (Hex) |
|---|---|
| 1 | 0 x 08 |
| 2 | 0 x 22 |
| 3 | 0 x 2A |
| 4 | 0 x 55 |
| 5 | 0 x 5D |
| 6 | 0 x 77 |

```
/*************************************************************************************
                                    SIMPLE DICE
                                    ==========

In this project 7 LEDs are connected to PORTC of a PIC18F45K22 microcontroller and the
microcontroller is operated from an 8 MHz crystal. The LEDs are organized as the faces
of a real dice. When a push-button switch connected to RB0 is pressed a dice pattern is
displayed on the LEDs. The display remains in this state for 3 s and after this period
the LEDs all turn OFF to indicate that the system is ready for the button to be pressed again.


Author:         Dogan Ibrahim
Date:           August 2013
File:           MIKROC-LED5.C
*************************************************************************************/

#define Switch PORTB.RB0
#define Pressed 0

void main()
{
   unsigned char J = 1;
   unsigned char Pattern;
   unsigned char DICE[] = {0,0x08,0x22,0x2A,0x55,0x5D,0x77};

   ANSELC = 0;                          // Configure PORTC as digital
   ANSELB = 0;                          // Configure PORTB as digital
   TRISC = 0;                           // Configure PORTC as outputs
   TRISB = 1;                           // Configure RB0 as input
   PORTC = 0;                           // Turn OFF all LEDs

   for(;;)                              // Endless loop
   {
      if(Switch == Pressed)             // Is switch pressed ?
      {
       Pattern = DICE[J];               // Get LED pattern
       PORTC = Pattern;                 // Turn on LEDs
       Delay_ms(3000);                  // Delay 3 s
       PORTC = 0;                       // Turn OFF all LEDs
       J = 0;                           // Initialize J
       }
       J++;                             // Increment J
       if(J == 7) J = 1;                // Back to 1 if  >6
   }
}
```

**Figure 5.25: mikroC Pro for PIC Program Listing.**

number. Variable **Pattern** is the data sent to the LEDs. The program then enters an endless **for** loop where the value of variable **J** is incremented very fast between 1 and 6. When the push-button switch is pressed, the LED pattern corresponding to the current value of **J** is read from the array and sent to the LEDs. The LEDs remain at this state for 3 s (using function Delay_ms with the argument set to 3000 ms), and after this time, they all turn OFF to indicate that the system is ready to generate a new dice number.

*MPLAB XC8*

The MPLAB XC8 program is named XC8-LED5.C, and the program listing is given in Figure 5.26. The 3 -s delay is created using function Delay_Seconds as before.

### *Using a Random Number Generator*

In the above project, the value of variable **J** changes very fast between 1 and 6, and when the push-button switch is pressed, the current value of this variable is taken and used as the dice number. Because the values of **J** are changing very fast, we can say that the numbers generated are random, that is, new numbers do not depend on the previous numbers.

In this section, we shall see how a pseudorandom number generator function can be used to generate the dice numbers. The modified program listing is shown in Figure 5.27 (MIKROC-LED5-1.C). In this program, a function called **Number** generates the dice numbers. Here, we could have used the built-in *rand* function as in Project 5.3, but a pseudorandom number generator function has been created instead to show how such a function works. The function receives the upper limit of the numbers to be generated (6 in this example), and also a seed value that defines the number set to be generated. In this example, the seed is set to 1. Every time the function is called, a number will be generated between 1 and 6.

The operation of the program is basically the same as in Figure 5.25. When the push-button switch is pressed, function **Number** is called to generate a new dice number between 1 and 6, and this number is used as an index in array **DICE** to find the bit pattern to be sent to the LEDs.

Figure 5.28 shows the MPLAB XC8 version of the program (XC8-LED5-1.C). In this version, the random number generator function is used. The function is divided by 6553 and then 1 is added to generate a number between 1 and 6.

## *Project 5.6—Two-Dice Project*

### *Project Description*

This project is similar to the previous project, but here a pair of dice are used instead of one. In many dice games (e.g. backgammon), a pair of dice are thrown together and then the player takes action based on the result.

The circuit given in Figure 5.23 can be modified by adding another set of seven LEDs for the second dice. For example, the first set of LEDs can be driven from PORTC, the second set from PORTD, and the push-button switch can be connected to RB0 as before. Such a design will require the use of 14 output ports just for the LEDs. Later on, we will see how

```
/*************************************************************************************
                                    SIMPLE DICE
                                    ==========

In this project 7 LEDs are connected to PORTC of a PIC18F45K22 microcontroller and the
microcontroller is operated from an 8 MHz crystal. The LEDs are organized as the faces
of a real dice. When a push-button switch connected to RB0 is pressed a dice pattern is
displayed on the LEDs. The display remains in this state for 3 s and after this period
the LEDs all turn OFF to indicate that the system is ready for the button to be pressed again.


Author:         Dogan Ibrahim
Date:           August 2013
File:           XC8-LED5.C
*************************************************************************************/

#include <xc.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

#define Switch PORTBbits.RB0
#define Pressed 0

//
// This function creates seconds delay. The argument specifies the delay time in seconds.
//
void Delay_Seconds(unsigned char s)
{
  unsigned char i,j;

  for(j = 0; j < s; j++)
  {
    for(i = 0; i <  100; i++)__delay_ms(10);
  }
}


void main()
{
   unsigned char J = 1;
   unsigned char Pattern;
   unsigned char DICE[] = {0,0x08,0x22,0x2A,0x55,0x5D,0x77};

   ANSELC = 0;                          // Configure PORTC as digital
   ANSELB = 0;                          // Configure PORTB as digital
   TRISC = 0;                           // Configure PORTC as outputs
   TRISB = 1;                           // Configure RB0 as input
   PORTC = 0;                           // Turn OFF all LEDs

   for(;;)                              // Endless loop
   {
```

**Figure 5.26: MPLAB XC8 Program Listing.**

```
        if(Switch == Pressed)                  // Is switch pressed ?
        {
         Pattern = DICE[J];                    // Get LED pattern
         PORTC = Pattern;                      // Turn on LEDs
         Delay_Seconds(3);                     // Delay 3 s
         PORTC = 0;                            // Turn OFF all LEDs
         J = 0;                                // Initialize J
        }
        J++;                                   // Increment J
        if(J == 7) J = 1;                      // Back to 1 if >6
    }
}
```

**Figure 5.26**
cont'd

the LEDs can be combined to reduce the I/O requirements. Figure 5.29 shows the block diagram of the project.

### *Project Hardware*

The circuit diagram of the project is shown in Figure 5.30. The circuit is basically the same as in Figure 5.23, with the addition of another set of LEDs, connected to PORTD.

If you are using the EasyPIC V7 development board, then make sure that the following jumpers are configured. Push-button switch RB0 on the board can be pressed to generate a dice number:

DIP switch SW3: PORTC ON
DIP switch SW3: PORTD ON
Jumper J17 (Button Press Level): GND

### *Project PDL*

The operation of the project is very similar to that in the previous project. Figure 5.31 shows the PDL for this project. At the beginning of the program, PORTC and PORTD pins are configured as outputs, and bit 0 of PORTB (RB0) is configured as input. The program then executes in a loop continuously and checks the state of the push-button switch. When the switch is pressed, two pseudorandom numbers are generated between 1 and 6, and these numbers are sent to PORTC and PORTD. The LEDs remain at this state for 3 s, and after this time, all the LEDs are turned OFF to indicate that the push button can be pressed again for the next pair of numbers.

```
/*************************************************************************
                                    SIMPLE DICE
                                    ===========

In this project 7 LEDs are connected to PORTC of a PIC18F45K22 microcontroller and the
microcontroller is operated from an 8 MHz crystal. The LEDs are organized as the faces of a real
dice. When a push-button switch connected to RB0 is pressed a dice pattern is displayed on the
LEDs. The display remains in this state for 3 s and after this period the LEDs all turn OFF to
indicate that the system is ready for the button to be pressed again.

In this program a pseudorandom number generator function is used to generate the
dice numbers between 1 and 6.

Author:      Dogan Ibrahim
Date:        August 2013
File:        MIKROC-LED5-1.C
*************************************************************************/

#define Switch PORTB.RB0
#define Pressed 0

//
// This function generates a pseudorandom integer number
// between 1 and Lim
//
unsigned char Number(int Lim, int Y)
{
  unsigned char Result;
  static unsigned int Y;

  Y = (Y * 32719 + 3) % 32749;
  Result = ((Y % Lim) + 1);
  return Result;
}


void main()
{
   unsigned char J, Pattern, Seed = 1;
   unsigned char DICE[] = {0,0x08,0x22,0x2A,0x55,0x5D,0x77};

   ANSELC = 0;                          // Configure PORTC as digital
   ANSELB = 0;                          // Configure PORTB as digital
   TRISC = 0;                           // Configure PORTC as outputs
   TRISB = 1;                           // Configure RB0 as input
   PORTC = 0;                           // Turn OFF all LEDs

   for(;;)                              // Endless loop
   {
      if(Switch == Pressed)             // Is switch pressed ?
      {
```

**Figure 5.27: Modified mikroC Pro for the PIC Dice Program.**

```
            J = Number(6, seed);          // Generate a Number 1 to 6
            Pattern = DICE[J];            // Get LED pattern
            PORTC = Pattern;              // Turn on LEDs
            Delay_ms(3000);               // Delay 3 s
            PORTC = 0;                    // Turn OFF all LED
           }
        }
    }
```

**Figure 5.27**
cont'd

### *Project Program*

#### *mikroC Pro for PIC*

The program is called MIKROC-LED6.C, and the program listing is given in Figure 5.32. At the beginning of the program, **Switch** is defined as bit 0 of PORTB, and **Pressed** is defined as 0. The relationship between the dice numbers and the LEDs to be turned on are stored in an array called **DICE** as in the previous project. Variable **Pattern** is the data sent to the LEDs. The program enters an endless **for** loop where the state of the push-button switch is checked continuously. When the switch is pressed, two random numbers are generated by calling function **Numbers**. The bit pattern to be sent to the LEDs are then determined and sent to PORTC and PORTD. The program then repeats inside the endless loop checking the state of push-button switch.

#### *MPLAB XC8*

The MPLAB X8 version of the program is shown in Figure 5.33. Here again, the built-in random number generator function rand is used to generate the dice numbers.

## *Project 5.7—Two-Dice Project Using Fewer I/O Pins*

### *Project Description*

This project is similar to Project 5.6, but here, LEDs are shared, which uses fewer input/output pins.

The LEDs in Table 5.2 can be grouped as shown in Table 5.4. Looking at this Table we can say that

- D4 can appear on its own,
- D2 and D6 are always together,
- D1 and D3 are always together,
- D5 and D7 are always together.

Thus, we can drive D4 on its own, and then drive the D2, D6 pair together in series, D1, D3 pair together in series, and also D5, D7 pair together in series (actually, we could share

```
/*********************************************************************************
                                SIMPLE DICE
                                ==========

In this project 7 LEDs are connected to PORTC of a PIC18F45K22 microcontroller and the
microcontroller is operated from an 8 MHz crystal. The LEDs are organized as the faces of a
real dice. When a push-button switch connected to RB0 is pressed a dice pattern is displayed
on the LEDs. The display remains in this state for 3 s and after this period the LEDs all
turn OFF to indicate that the system is ready for the button to be pressed again.

 The random number generator function is used in this program.
 *
Author:         Dogan Ibrahim
Date:           August 2013
File:           XC8-LED5-1.C
*********************************************************************************/

#include <xc.h>
#include <stdlib.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

#define Switch PORTBbits.RB0
#define Pressed 0

//
// This function creates seconds delay. The argument specifies the delay time in seconds.
//
void Delay_Seconds(unsigned char s)
{
  unsigned char i,j;

  for(j = 0; j < s; j++)
  {
    for(i = 0; i <  100; i++)__delay_ms(10);
  }
}


void main()
{
   unsigned char J, Pattern, Seed = 1;
   unsigned char DICE[] = {0,0x08,0x22,0x2A,0x55,0x5D,0x77};

   ANSELC = 0;                          // Configure PORTC as digital
   ANSELB = 0;                          // Configure PORTB as digital
   TRISC = 0;                           // Configure PORTC as outputs
   TRISB = 1;                           // Configure RB0 as input
   PORTC = 0;                           // Turn OFF all LEDs
   srand(1);                            // Initialize rand function
```

**Figure 5.28: Modified MPLAB XC8 Dice Program.**

```
    for(;;)                                  // Endless loop
    {
       if(Switch == Pressed)                 // Is switch pressed ?
       {
        J = rand() / 6553 + 1;               // Generate a number 1 to 6
        Pattern = DICE[J];                   // Get LED pattern
        PORTC = Pattern;                     // Turn on LEDs
        Delay_Seconds(3);                    // Delay 3 s
        PORTC = 0;                           // Turn OFF all LEDs
       }
    }
}
```

**Figure 5.28**
cont'd

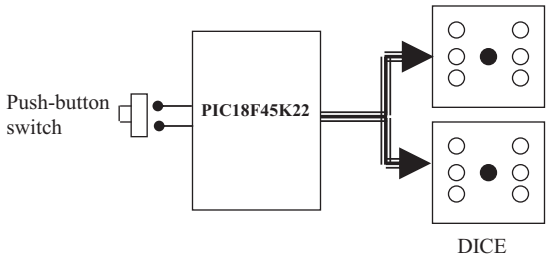


**Figure 5.29: Block Diagram of the Project.**

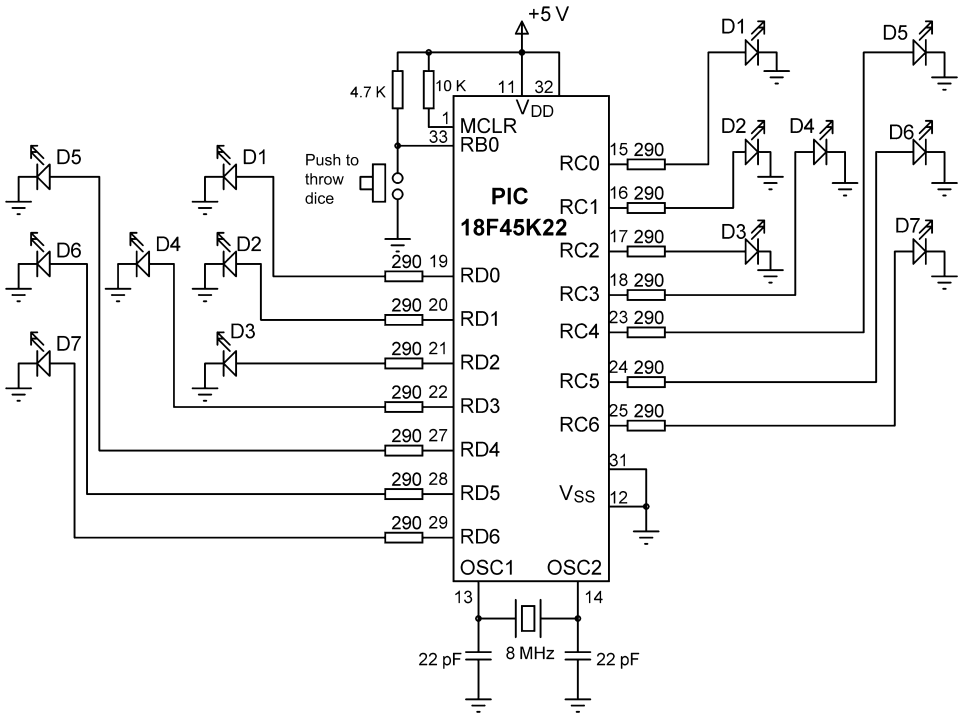


**Figure 5.30: Circuit Diagram of the Project.**

```
BEGIN
    Create DICE table
    Configure PORTC as outputs
    Configure PORTD as outputs
    Configure RB0 as input
    DO FOREVER
        IF button pressed THEN
            Get a random number between 1 and 6
            Find bit pattern
            Turn ON LEDs on PORTC
            Get second random number between 1 and 6
            Find bit pattern
            Turn on LEDs on PORTD
            Wait 3 s
            Turn OFF all LEDs
        ENDIF
    ENDDO
END
```

**Figure 5.31: PDL of the Project.**

D1, D3, D5, D7, but this would require 8 V to drive if the LEDs are connected in series. Connecting these LEDs in parallel will require excessive current and a driver IC will be required). Altogether four lines will be required to drive seven LEDs of a dice. Similarly, four lines will be required to drive the second dice. Thus, a pair of dice can easily be driven from an 8-bit output port.

### *Project Hardware*

The circuit diagram of the project is shown in Figure 5.34. PORTC of a PIC18F45K22 microcontroller is used to drive the LEDs as follows:

- RC0 drives D2, D6 of the first dice,
- RC1 drives D1, D3 of the first dice,
- RC2 drives D5, D7 of the first dice,
- RC3 drives D4 of the first dice,
- RC4 drives D2, D6 of the second dice,
- RC5 drives D1, D3 of the second dice,
- RC6 drives D5, D7 of the second dice,
- RC7 drives D4 of the second dice.

Since we are driving two LEDs on some outputs, we can calculate the required value of the current limiting resistors. Assuming that the voltage drop across each LED is 2 V, the current through the LED is 10 mA, and the output high voltage of the microcontroller is 4.85 V, the required resistors are $R = \frac{4.85-2-2}{10} = 85\ \Omega$. We will choose 100-Ω resistors.

We now need to find the relationship between the dice numbers and the bit pattern to be sent to the LEDs for each dice. Table 5.5 shows the relationship between the first dice

```
/*************************************************************************
                                    TWO DICE
                                    ========

In this project 7 LEDs are connected to PORTC of a PIC18F452 microcontroller and 7 LEDs to
PORTD. The microcontroller is operated from an 8 MHz crystal. The LEDs are organized as the
faces of a real dice. When a push-button switch connected to RB0 is pressed a dice pattern is
displayed on the LEDs. The display remains in this state for 3 s and after this period the
LEDs all turn OFF to indicate that the system is ready for the button to be pressed again.

In this program a pseudorandom number generator function is used to generate the dice
numbers between 1 and 6.

Author:         Dogan Ibrahim
Date:           August 2013
File:           MIKROC-LED6.C
*************************************************************************/

#define Switch PORTB.F0
#define Pressed 0

//
// This function generates a pseudorandom integer number between 1 and Lim.
//
unsigned char Number(int Lim, int Y)
{
  unsigned char Result;
  static unsigned int Y;

  Y = (Y * 32719 + 3) % 32749;
  Result = ((Y % Lim) + 1);
  return Result;
}


//
// Start of MAIN program
//
void main()
{
   unsigned char J,Pattern,Seed = 1;
   unsigned char DICE[] = {0,0x08,0x22,0x2A,0x55,0x5D,0x77};

   ANSELC = 0;                          // Configure PORTC as digital
   ANSELD = 0;                          // Configure PORTD as digital
   ANSELB = 0;                          // Configure PORTB as digital
   TRISC = 0;                           // Configure PORT as outputs
   TRISD = 0;                           // Configure PORTD as outputs
   TRISB = 1;                           // Configure RB0 as input
   PORTC = 0;                           // Turn OFF all LEDs
   PORTD = 0;                           // Turn OFF all LEDs
```

**Figure 5.32: mikroC Pro for PIC Program Listing.**

```
    for(;;)                                  // Endless loop
    {
       if(Switch == Pressed)                 // Is switch pressed ?
       {
        J = Number(6,seed);                  // Generate first dice number
        Pattern = DICE[J];                   // Get LED pattern
        PORTC = Pattern;                     // Turn on LEDs for first dice
        J = Number(6,seed);                  // Generate second dice number
        Pattern = DICE[J];                   // Get LED pattern
        PORTD = Pattern;                     // Turn on LEDs for second dice
        Delay_ms(3000);                      // Delay 3 seconds
        PORTC = 0;                           // Turn OFF all LEDs
        PORTD = 0;                           // Turn OFF all LEDS
       }
    }
}
```

**Figure 5.32**
cont'd

numbers and the bit pattern to be sent to port pins RC0–RC3. Similarly, Table 5.6 shows the relationship between the second dice numbers and the bit pattern to be sent to port pins RC4–RC7.

We can now find the 8-bit number to be sent to PORTC to display both dice numbers as follows:

- Get the first number from the number generator, call this P.
- Index DICE table to find the bit pattern for a low nibble, that is, L = DICE[P].
- Get the second number from the number generator, call this P.
- Index DICE table to find the bit pattern for a high nibble, that is, U = DICE[P].
- Multiply the high nibble with 16 and add a low nibble to find the number to be sent to PORTC, that is, R = 16 * U + L where R is the 8-bit number to be sent to PORTC to display both dice values.

If you are using the EasyPIC V7 development board, then make sure that the following jumpers are configured. Push-button switch RB0 on the board can be pressed to generate a dice number:

DIP switch SW3: PORTC ON
DIP switch SW3: PORTD ON
Jumper J17 (Button Press Level): GND

### *Project PDL*

Figure 5.35 shows the PDL for this project. At the beginning of the program, PORTC pins are configured as outputs, and bit 0 of PORTB (RB0) is configured as the input. The

```
/*************************************************************************
                                    TWO DICE
                                    ========

In this project 7 LEDs are connected to PORTC of a PIC18F452 microcontroller and 7 LEDs to
PORTD. The microcontroller is operated from an 8 MHz crystal. The LEDs are organized as the
faces of a real dice. When a push-button switch connected to RB0 is pressed a dice pattern is
displayed on the LEDs. The display remains in this state for 3 s and after this period the
LEDs all turn OFF to indicate that the system is ready for the button to be pressed again.

In this program a pseudorandom number generator function is used to generate the dice numbers
between 1 and 6.

Author:         Dogan Ibrahim
Date:           August 2013
File:           XC8-LED6.C
*************************************************************************/

#include <xc.h>
#include <stdlib.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

#define Switch PORTBbits.RB0
#define Pressed 0

//
// This function creates seconds delay. The argument specifies the delay time in seconds.
//
void Delay_Seconds(unsigned char s)
{
  unsigned char i,j;

  for(j = 0; j < s; j++)
  {
    for(i = 0; i <  100; i++)__delay_ms(10);
  }
}


void main()
{
   unsigned char J, Pattern, Seed = 1;
   unsigned char DICE[] = {0,0x08,0x22,0x2A,0x55,0x5D,0x77};

   ANSELC = 0;                          // Configure PORTC as digital
   ANSELD = 0;                          // Configure PORTD as digital
   ANSELB = 0;                          // Configure PORTB as digital
   TRISC = 0;                           // Configure PORT as outputs
   TRISD = 0;                           // Configure PORTD as outputs
   TRISB = 1;                           // Configure RB0 as input
```

**Figure 5.33: MPLAB XC8 Program Listing.**

```
    PORTC = 0;                          // Turn OFF all LEDs
    PORTD = 0;                          // Turn OFF all LEDs
    srand(1);                           // Initialize rand function

    for(;;)                             // Endless loop
    {
       if(Switch == Pressed)            // Is switch pressed ?
       {
        J = rand() / 6553 + 1;          // Generate a number 1 to 6
        Pattern = DICE[J];              // Get LED pattern
        PORTC = Pattern;                // Turn on PORTC LEDs
        J = rand() / 6553 + 1;          // Generate a number 1 to 6
        Pattern = DICE[J];              // Get LED pattern
        PORTD = Pattern;                // Turn on PORTD LEDs
        Delay_Seconds(3);               // Delay 3 s
        PORTC = 0;                      // Turn OFF PORTC LEDs
        PORTD = 0;                      // Turn OFF PORTD LEDs
       }
    }
}
```

**Figure 5.33**
cont'd

program then executes in a loop continuously and checks the state of the push-button switch. When the switch is pressed, two pseudorandom numbers are generated between 1 and 6, and the bit pattern to be sent to PORTC is found using the method described above. This bit pattern is then sent to PORTC to display both dice numbers at the same time. The display shows the dice numbers for 3 s, and then, all the LEDs turn OFF to indicate that the system is waiting for the push button to be pressed again to display next set of numbers.

## *Project Program*

### *mikroC Pro for PIC*

The mikroC Pro for the PIC program is called MIKROC-LED7.C, and the program listing is given in Figure 5.36. At the beginning of the program, **Switch** is defined as bit 0 of

**Table 5.4: Grouping the LEDs**

| Required Number | LEDs to be Turned on |
|---|---|
| 1 | D4 |
| 2 | D2, D6 |
| 3 | D2, D6, D4 |
| 4 | D1, D3, D5, D7 |
| 5 | D1, D3, D5, D7, D4 |
| 6 | D2, D6, D1, D3, D5, D7 |

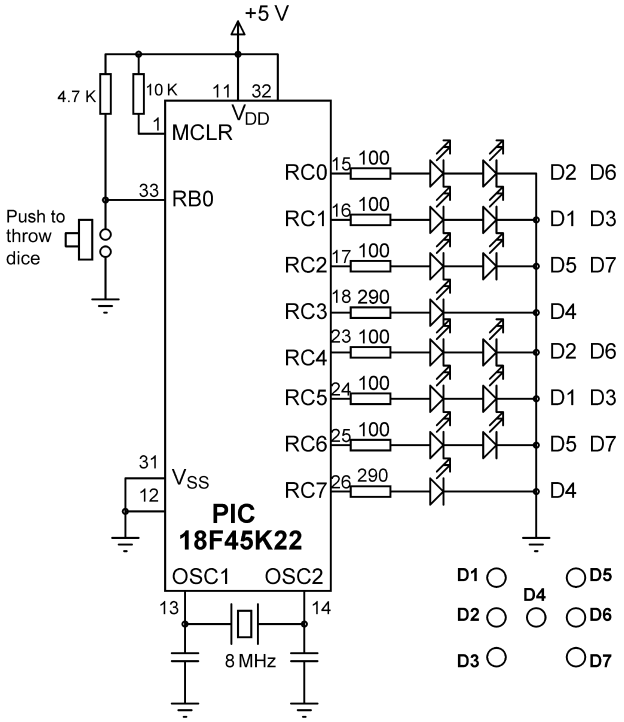


**Figure 5.34: Circuit Diagram of the Project.**

PORTB, and **Pressed** is defined as 0. The relationship between the dice numbers and the LEDs to be turned on are stored in an array called **DICE** as in the previous project. Variable **Pattern** is the data sent to the LEDs. The program enters an endless **for** loop where the state of the push-button switch is checked continuously. When the switch is pressed, two random numbers are generated by calling function **Numbers**. Variables **L** and **U** store the lower and the higher nibbles of the bit pattern to be sent to PORTC. The bit pattern to be sent to PORTC is then determined using the method described in the Project hardware section and stored in variable **R**. This bit pattern is then sent to PORTC to display both dice numbers at the same time. The dice is displayed for 3 s, and after this period, the LEDs are turned OFF to indicate that the system is ready.

**Table 5.5: First Dice Bit Patterns**

| Dice Number | RC3 | RC2 | RC1 | RC0 | Hex Value |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 8 |
| 2 | 0 | 0 | 0 | 1 | 1 |
| 3 | 1 | 0 | 0 | 1 | 9 |
| 4 | 0 | 1 | 1 | 0 | 6 |
| 5 | 1 | 1 | 1 | 0 | E |
| 6 | 0 | 1 | 1 | 1 | 7 |

**Table 5.6: Second Dice Bit Patterns**

| Dice Number | RC7 | RC6 | RC5 | RC4 | Hex Value |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 8 |
| 2 | 0 | 0 | 0 | 1 | 1 |
| 3 | 1 | 0 | 0 | 1 | 9 |
| 4 | 0 | 1 | 1 | 0 | 6 |
| 5 | 1 | 1 | 1 | 0 | E |
| 6 | 0 | 1 | 1 | 1 | 7 |

*MPLAB XC8*

The MPLAB X8 version of the program is shown in Figure 5.37 (XC8-LED7.C). Here, again the built-in random number generator function rand is used to generate the dice numbers.

## *Modifying the Program*

The program given in Figure 5.36 can be modified and made more efficient by combining the two dice nibbles into a single table value. The new program is described in this section.

There are 36 possible combinations of two dice values. Referring to Tables 5.5 and 5.6 and Figure 5.34, we can create Table 5.7 to show all the possible two dice values and the corresponding numbers to be sent to PORTC.

The modified program (program name MIKROC-LED7-1.C) is given in Figure 5.38. In this program, array **DICE** contains the 36 possible dice values. Program enters an endless **for** loop, and inside this loop, the state of the push-button switch is checked. Also, a variable is incremented from 1 to 36, and when the button is pressed, the value of this variable is used as an index to array **DICE** to determine the bit pattern to be sent to

```
BEGIN
    Create DICE table
    Configure PORTC as outputs
    Configure RB0 as input
    DO FOREVER
        IF button pressed THEN
            Get a random number between 1 and 6
            Find low nibble bit pattern
            Get second random number between 1 and 6
            High high nibble bit pattern
            Calculate data to be sent to PORTC
            Wait 3 s
            Turn OFF all LEDs
        ENDIF
    ENDDO
END
```

**Figure 5.35: PDL of the Project.**

```
/*************************************************************************
                        TWO DICE - FEWER I/O COUNT
                        ==========================

In this project LEDs are connected to PORTC of a PIC18F45K22 microcontroller and
the microcontroller is operated from an 8 MHz crystal. The LEDs are organized as the
faces of a real dice. When a push-button switch connected to RB0 is pressed a dice
pattern is displayed on the LEDs. The display remains in this state for 3 s and
after this period the LEDs all turn OFF to indicate that the system is ready for the
button to be pressed again.

In this program a pseudorandom number generator function is used to generate
the dice numbers between 1 and 6.

Author:         Dogan Ibrahim
Date:           August 2013
File:           MIKROC-LED7.C
*************************************************************************/

#define Switch PORTB.RB0
#define Pressed 0

//
// This function generates a pseudorandom integer Number between 1 and Lim.
//
unsigned char Number(int Lim, int Y)
{
        unsigned char Result;
        static unsigned int Y;

        Y = (Y * 32719 + 3) % 32749;
        Result = ((Y % Lim) + 1);
        return Result;
}


//
// Start of MAIN program
//
void main()
{
   unsigned char J,L,U,R,Seed = 1;
   unsigned char DICE[] = {0,0x08,0x01,0x09,0x06,0x0E,0x07};

   ANSELC = 0;                          // Configure PORTC as digital
   ANSELB = 0;                          // Configure PORTB as digital
   TRISC = 0;                           // Configure PORTC as outputs
   TRISB = 1;                           // Configure RB0 as input
   PORTC = 0;                           // Turn OFF all LEDs

   for(;;)                              // Endless loop
```

**Figure 5.36: mikroC Pro for PIC Program Listing.**

```
    {
      if(Switch == Pressed)                  // Is switch pressed ?
      {
        J = Number(6,seed);                  // Generate first dice number
        L = DICE[J];                         // Get LED pattern
        J = Number(6,seed);                  // Generate second dice number
        U = DICE[J];                         // Get LED pattern
        R = 16 * U + L;                      // Bit pattern to send to PORTC
              PORTC = R;                     // Turn on LEDs for both dice
        Delay_ms(3000);                      // Delay 3 s
        PORTC = 0;                           // Turn OFF all LEDs
      }
    }
}
```

**Figure 5.36**
cont'd

PORTC. As before, the program displays the dice numbers for 3 s and then turns OFF all LEDs to indicate that it is ready.

The MPLAB XC8 version of the modified program is shown in Figure 5.39 (XC8-LED7-1.C).

## *Project 5.8—7-Segment LED Counter*

### *Project Description*

This project describes the design of a 7-segment LED based counter that counts from 0 to 9 continuously with a 1-s delay between each count. The project shows how a 7-segment LED can be interfaced and used in a PIC microcontroller project.

Seven-segment displays are used frequently in electronic circuits to show numeric or alphanumeric values. As shown in Figure 5.40, a 7-segment display basically consists of seven LEDs connected such that numbers from 0 to 9 and some letters can be displayed. Segments are identified by letters from **a** to **g**, and Figure 5.41 shows the segment names of a typical 7-segment display.

Figure 5.42 shows how numbers from 0 to 9 can be obtained by turning ON different segments of the display.

Seven-segment displays are available in two different configurations: **common cathode** and **common anode**. As shown in Figure 5.43, in common-cathode configuration, all the cathodes of all segment LEDs are connected together to the ground. The segments are turned ON by applying a logic 1 to the required segment LED via current limiting resistors. In the common-cathode configuration, the 7-segment LED is connected to the microcontroller in the current-sourcing mode.

```
/*********************************************************************************
                        TWO DICE - FEWER I/O COUNT
                        ==========================

In this project LEDs are connected to PORTC of a PIC18F45K22 microcontroller and the
microcontroller is operated from an 8 MHz crystal. The LEDs are organized as the faces of a
real dice. When a push-button switch connected to RB0 is pressed a dice pattern is displayed
on the LEDs. The display remains in this state for 3 s and after this period the LEDs all
turn OFF to indicate that the system is ready for the button to be pressed again.

In this program a pseudorandom number generator function is used to generate the dice
numbers between 1 and 6.

Author:         Dogan Ibrahim
Date:           August 2013
File:           XC8-LED7.C
*********************************************************************************/

#include <xc.h>
#include <stdlib.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

#define Switch PORTBbits.RB0
#define Pressed 0

//
// This function creates seconds delay. The argument specifies the delay time in seconds.
//
void Delay_Seconds(unsigned char s)
{
  unsigned char i,j;

  for(j = 0; j < s; j++)
  {
    for(i = 0; i <  100; i++)__delay_ms(10);
  }
}

void main()
{
   unsigned char J,L,U,R;
   unsigned char DICE[] = {0,0x08,0x22,0x2A,0x55,0x5D,0x77};

   ANSELC = 0;                          // Configure PORTC as digital
   ANSELB = 0;                          // Configure PORTB as digital
   TRISC = 0;                           // Configure PORTC as outputs
   TRISB = 1;                           // Configure RB0 as input
   PORTC = 0;                           // Turn OFF all LEDs
   srand(1);                            // Initialize rand function
```

**Figure 5.37: MPLAB XC8 Program Listing.**

```
    for(;;)                                  // Endless loop
    {
       if(Switch == Pressed)                 // Is switch pressed ?
       {
        J = rand() / 6553 + 1;               // Generate a number 1 to 6
        L = DICE[J];                         // Get dice pattern
        J = rand() / 6553 + 1;               // Generate another number
        U = DICE[J];                         // Get dice pattern
        R = 16 * U + L;                      // Bit pattern to send to PORTC
        PORTC = R;                           // Send bit pattern to PORTC
        Delay_Seconds(3);                    // Delay 3 s
        PORTC = 0;                           // Turn OFF PORTC LEDs
       }
    }
}
```

**Figure 5.37**
cont'd

In a common anode configuration, the anode terminals of all the LEDs are connected together as shown in Figure 5.44. This common point is then normally connected to the supply voltage. A segment is turned ON by connecting its cathode terminal to logic 0 via a current limiting resistor. In the common anode configuration, the 7-segment LED is connected to the microcontroller in the current-sinking mode.

In this project, a *Kingbright SA52-11* model red common anode 7-segment display is used. This is a 13-mm (0.52-in) display with 10 pins, and it also has a segment LED for the decimal point. Table 5.8 shows the pin configuration of this display.

**Table 5.7: Two Dice Combinations and Number to be Sent to PORTC**

| Dice Numbers | PORTC Value | Dice Numbers | PORTC Value |
|---|---|---|---|
| 1,1 | 0 x 88 | 4,1 | 0 x 86 |
| 1,2 | 0 x 18 | 4,2 | 0 x 16 |
| 1,3 | 0 x 98 | 4,3 | 0 x 96 |
| 1,4 | 0 x 68 | 4,4 | 0 x 66 |
| 1,5 | 0 x E8 | 4,5 | 0 x E6 |
| 1,6 | 0 x 78 | 4,6 | 0 x 76 |
| 2,1 | 0 x 81 | 5,1 | 0 x 8E |
| 2,2 | 0 x 11 | 5,2 | 0 x 1E |
| 2,3 | 0 x 91 | 5,3 | 0 x 9E |
| 2,4 | 0 x 61 | 5,4 | 0 x 6E |
| 2,5 | 0 x E1 | 5,5 | 0 x EE |
| 2,6 | 0 x 71 | 5,6 | 0 x 7E |
| 3,1 | 0 x 89 | 6,1 | 0 x 87 |
| 3,2 | 0 x 19 | 6,2 | 0 x 17 |
| 3,3 | 0 x 99 | 6,3 | 0 x 97 |
| 3,4 | 0 x 69 | 6,4 | 0 x 67 |
| 3,5 | 0 x E9 | 6,5 | 0 x E7 |
| 3,6 | 0 x 79 | 6,6 | 0 x 77 |

```
/*************************************************************************
                                TWO DICE - LESS I/O COUNT
                                =========================

In this project LEDs are connected to PORTC of a PIC18F452 microcontroller and the
microcontroller is operated from an 8 MHz crystal. The LEDs are organized as the faces
of a real dice. When a push-button switch connected to RB0 is pressed a dice pattern
is displayed on the LEDs. The display remains in this state for 3 s and after this
period the LEDs all turn OFF to indicate that the system is ready for the button to be
pressed again.

In this program a pseudorandom number generator function is used to generate the dice
numbers between 1 and 6.

Author:         Dogan Ibrahim
Date:           August 2013
File:           MIKROC-LED7-1.C
*************************************************************/

#define Switch PORTB.F0
#define Pressed 0

//
// Start of MAIN program
//
void main()
{
   unsigned char Pattern, J = 1;
   unsigned char DICE[] = {0,0x88,0x18,0x98,0x68,0xE8,0x78,
                           0x81,0x11,0x91,0x61,0xE1,0x71,
                           0x89,0x19,0x99,0x69,0xE9,0x79,
                           0x86,0x16,0x96,0x66,0xE6,0x76,
                           0x8E,0x1E,0x9E,0x6E,0xEE,0x7E,
                           0x87,0x17,0x97,0x67,0xE7,0x77};

   ANSELC = 0;                          // Configure PORTC as digital
   ANSELB = 0;                          // Configure PORTB as digital
   TRISC = 0;                           // Configure PORTC as outputs
   TRISB = 1;                           // Configure RB0 as input
   PORTC = 0;                           // Turn OFF all LEDs

   for(;;)                              // Endless loop
   {
      if(Switch == Pressed)             // Is switch pressed ?
      {
        Pattern = DICE[J];              // Number to send to PORTC
        PORTC = Pattern;                // Send to PORTC
        Delay_ms(3000);                 // 3 seconds delay
        PORTC = 0;                      // Clear PORTC
      }
        J++;                            // Increment J

      if(J == 37) J = 1;                // If J = 37, reset to 1
   }
}
```

**Figure 5.38: Modified mikroC pro for PIC Program.**

```
/*********************************************************************
                        TWO DICE - LESS I/O COUNT
                        =========================

In this project LEDs are connected to PORTC of a PIC18F452 microcontroller and the
microcontroller is operated from an 8 MHz crystal. The LEDs are organized as the faces
of a real dice. When a push-button switch connected to RB0 is pressed a dice pattern
is displayed on the LEDs. The display remains in this state for 3 s and after this
period the LEDs all turn OFF to indicate that the system is ready for the button to be
pressed again.

In this program a pseudorandom number generator function is used to generate
the dice numbers between 1 and 6.

Author:         Dogan Ibrahim
Date:           August 2013
File:           XC8-LED7-1.C
*********************************************************************/

#include <xc.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

#define Switch PORTBbits.RB0
#define Pressed 0

//
// This function creates seconds delay. The argument specifies the delay time
// in seconds.
//
void Delay_Seconds(unsigned char s)
{
   unsigned char i,j;

   for(j = 0; j < s; j++)
   {
      for(i = 0; i <  100; i++)__delay_ms(10);
   }
}


void main()
{
    unsigned char Pattern, J = 1;
   unsigned char DICE[] = {0,0x88,0x18,0x98,0x68,0xE8,0x78,
                           0x81,0x11,0x91,0x61,0xE1,0x71,
                           0x89,0x19,0x99,0x69,0xE9,0x79,
                           0x86,0x16,0x96,0x66,0xE6,0x76,
                           0x8E,0x1E,0x9E,0x6E,0xEE,0x7E,
                           0x87,0x17,0x97,0x67,0xE7,0x77};
```

**Figure 5.39: Modified MPLAB XC8 Program.**

```
    ANSELC = 0;                          // Configure PORTC as digital
    ANSELB = 0;                          // Configure PORTB as digital
    TRISC = 0;                           // Configure PORTC as outputs
    TRISB = 1;                           // Configure RB0 as input
    PORTC = 0;                           // Turn OFF all LEDs

    for(;;)                              // Endless loop
    {
       if(Switch == Pressed)             // Is switch pressed ?
       {
        Pattern = DICE[J];               // Number to send to PORTC
        PORTC = Pattern;                 // Send to PORTC
        Delay_Seconds(3);                // 3 s delay
        PORTC = 0;                       // Clear PORTC
       }
        J++;                             // Increment J
        if(J == 37) J = 1;               // If J = 37, reset to 1
    }

}
```

**Figure 5.39**
cont'd

## *Project Hardware*

The circuit diagram of the project is shown in Figure 5.44. A PIC18F45K22-type microcontroller is used with an 8-MHz crystal. Segments **a**–**g** of the display are connected to PORTC of the microcontroller through 290-Ω current limiting resistors. Before driving the display, we have to know the relationship between the numbers to be displayed and the corresponding segments to be turned ON, and this is shown in Table 5.9. For example, to

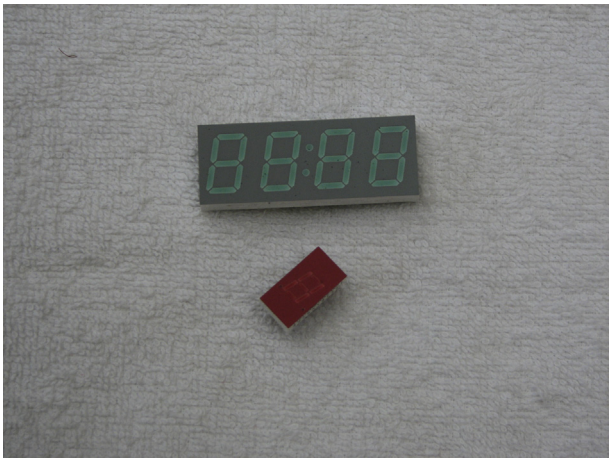

**Figure 5.40: Some 7-Segment Displays. (For color version of this figure, the reader is referred to the online version of this book.)**

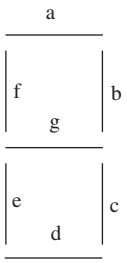


**Figure 5.41: Segment Names of a 7-Segment Display.**

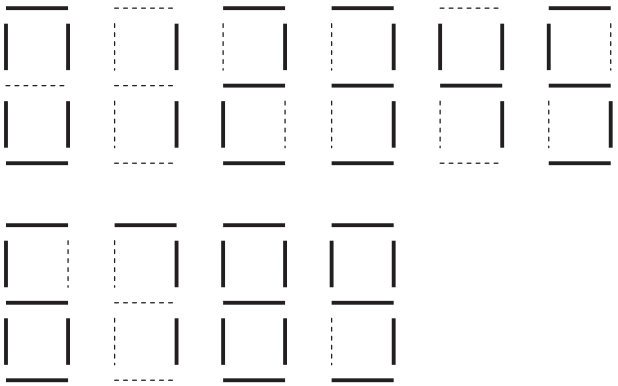

**Figure 5.42: Displaying Numbers 0–9.**

display number 3, we have to send the hexadecimal number 0x4F to PORTC which turns ON segments **a**, **b**, **c**, **d**, and **g**. Similarly, to display number 9, we have to send the hexadecimal number 0x6F to PORTC, which turns ON segments **a**, **b**, **c**, **d**, **f**, and **g**.

### *Project PDL*

The operation of the project is shown in Figure 5.45 with a PDL. At the beginning of the program, an array called SEGMENT is declared and filled with the relationship between the

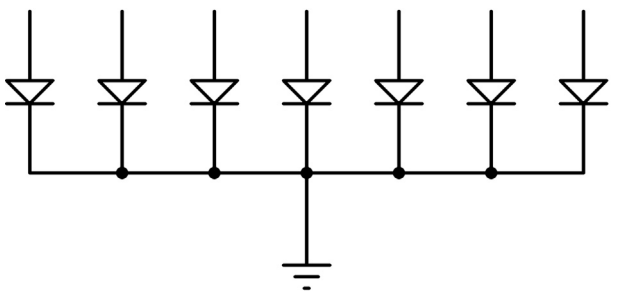

**Figure 5.43: Common-Cathode 7-Segment Display.**

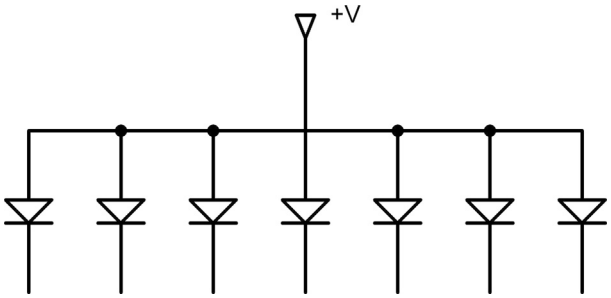


Figure 5.44: Circuit Diagram of the Project.

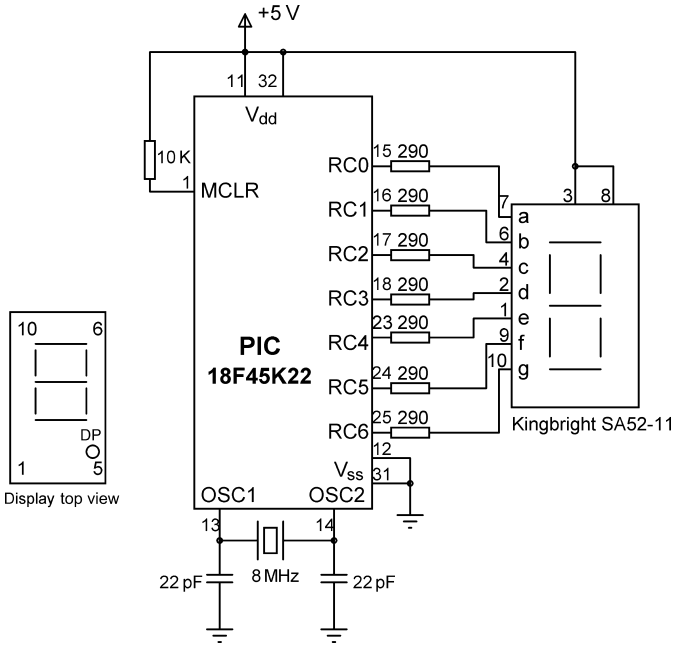


Figure 5.44
cont'd

numbers 0–9 and the data to be sent to PORTC. PORTC pins are then configured as outputs, and a variable is initialized to 0. The program then enters an endless loop where the variable is incremented between 0 and 9, and the corresponding bit pattern to turn ON the appropriate segments is sent to PORTC continuously with a 1-s delay between each output.

### *Project Program*

#### *mikroC Pro for PIC*

The mikroC Pro for PIC program is called MIKROC-LED8.C, and the program listing is given in Figure 5.46. At the beginning of the program, character variables **Pattern** and

**Table 5.8: The SA52-11 Pin Configuration**

| Pin Number | Segment |
|---|---|
| 1 | e |
| 2 | d |
| 3 | Common anode |
| 4 | c |
| 5 | Decimal point |
| 6 | b |
| 7 | a |
| 8 | Common anode |
| 9 | f |
| 10 | g |

**Table 5.9: Displayed Number and Data Sent to PORTC**

| Number | x | g | f | e | d | c | b | a | PORTC Data |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 x 3F |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 x 06 |
| 2 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 x 5B |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 x 4F |
| 4 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 x 66 |
| 5 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 x 6D |
| 6 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 x 7D |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 x 07 |
| 8 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 x 7F |
| 9 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 x 6F |

**x** is not used, taken as 0.

**Cnt** are declared, and **Cnt** is cleared to 0. Then, Table 5.9 is implemented using array **SEGMENT**. After configuring PORTC pins as outputs, the program enters an endless loop using the **for** statement. Inside the loop, the bit pattern corresponding to the contents of **Cnt** is found and stored in variable **Pattern**. Because we are using a common anode

```
BEGIN
    Create SEGMENT table
    Configure PORTC as digital outputs
    Initialize CNT to 0
    DO FOREVER
        Get bit pattern from SEGMENT corresponding to CNT
        Send this bit pattern to PORTC
        Increment CNT between 0 and 9
        Wait 1 s
    ENDDO
END
```

**Figure 5.45: PDL of the Project.**

```
/*************************************************************************
                              7-SEGMENT DISPLAY
                              =================

In this project a common anode 7-segment LED display is connected to PORTC of a PIC18F45K22
microcontroller and the microcontroller is operated from an 8 MHz crystal. The program displays
numbers 0 to 9 on the display with a 1 s delay between each output.

Author:         Dogan Ibrahim
Date:           August 2013
File:           MIKROC-LED8.C
*************************************************************************/

void main()
{
   unsigned char Pattern, Cnt = 0;
   unsigned char SEGMENT[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,
     0x7D,0x07,0x7F,0x6F};

   ANSELC = 0;                          // Configure PORTC as digital
   TRISC = 0;                           // Configure PORTC as outputs

   for(;;)                              // Endless loop
   {
     Pattern = SEGMENT[Cnt];            // Number to send to PORTC
     Pattern = ~Pattern;                // Invert bit pattern
     PORTC = Pattern;                   // Send to PORTC
     Cnt++;                             // Increment Cnt
     if(Cnt == 10) Cnt = 0;             // Cnt is between 0 and 9
     Delay_ms(1000);                    // 1 s delay
   }
}
```

**Figure 5.46: mikroC Pro for PIC Program Listing.**

display, a segment is turned ON when it is at logic 0, and thus, the bit pattern is inverted before sending to PORTC. The value of **Cnt** is then incremented between 0 and 9, after which the program waits for a second before repeating the above sequence.

*MPLAB XC8*

The MPLAB X8 version of the program is shown in Figure 5.47 (XC8-LED8.C). The operation of the program is as in mikroC Pro for PIC.

### *Modified Program*

Note that the program can be made more readable if we create a function to display the required number and then call this function from the main program.

```
/*********************************************************************************
                                7-SEGMENT DISPLAY
                                =================

In this project a common anode 7-segment LED display is connected to PORTC of a PIC18F45K22
microcontroller and the microcontroller is operated from an 8 MHz crystal. The program displays
numbers 0 to 9 on the display with a 1 s delay between each output.

Author:         Dogan Ibrahim
Date:           August 2013
File:           XC8-LED8.C
*********************************************************************************/

#include <xc.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000


//
// This function creates seconds delay. The argument specifies the delay time
// in seconds.
//
void Delay_Seconds(unsigned char s)
{
  unsigned char i,j;

  for(j = 0; j < s; j++)
  {
    for(i = 0; i <  100; i++)__delay_ms(10);
  }
}


void main()
{
  unsigned char Pattern, Cnt = 0;
   unsigned char SEGMENT[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};

   ANSELC = 0;                                  // Configure PORTC as digital
   TRISC = 0;                                   // Configure PORTC as outputs

   for(;;)                                      // Endless loop
   {
     Pattern = SEGMENT[Cnt];                     // Number to send to PORTC
     Pattern = ~Pattern;                        // Invert bit pattern
     PORTC = Pattern;                           // Send to PORTC
     Cnt++;                                     // Increment Cnt
     if(Cnt == 10) Cnt = 0;                     // Cnt is between 0 and 9
     Delay_Seconds(1);                          // 1 s delay
   }
}
```

**Figure 5.47: MPLAB XC8 Program Listing.**

## Project 5.9—Two-Digit Multiplexed 7-Segment LED

### Project Description

This project is similar to the previous project, but here, multiplexed two digits are used instead of just one digit, and a fixed number is displayed. In this project, number 25 is displayed as an example. In multiplexed LED applications (Figure 5.48), the LED segments of all the digits are tied together, and the common pins of each digit is turned ON separately by the microcontroller. By displaying each digit for several milliseconds, the eye cannot differentiate that the digits are not ON all the time. In this way we can multiplex any number of 7-segment displays together. For example, to display number 53, we have to send 5 to the first digit and enable its common pin. After a few milliseconds, number 3 is sent to the second digit, and the common point of the second digit is enabled. When this process is repeated continuously, the user sees as if both displays are ON continuously.

Some manufacturers provide multiplexed multidigit displays in single packages. For example, we can purchase two-, four-, or eight-digit multiplexed displays in a single package. The display used in this project is the DC56-11EWA which is a red 0.56-in height common-cathode two digit display having 18 pins and the pin configuration as shown in Table 5.10. This display can be controlled from the microcontroller as follows:

- Send the segment bit pattern for digit 1 to segments **a**–**g**.
- Enable digit 1.
- Wait for a few milliseconds.
- Disable digit 1.
- Send the segment bit patter for digit 2 to segments **a**–**g**.
- Enable digit 2.
- Wait for a few milliseconds.
- Disable digit 2.
- Repeat the above process continuously.

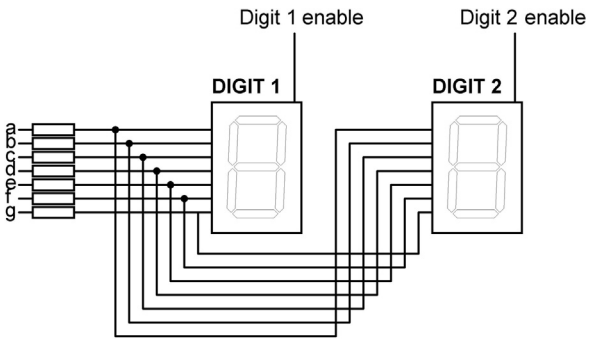


**Figure 5.48: Two Multiplexed 7-Segment Displays.**

**Table 5.10: Pin Configuration of the DC56-11EWA Dual Display**

| Pin Number | Segment |
|---|---|
| 1,5 | e |
| 2,6 | d |
| 3,8 | c |
| 14 | Digit 1 enable |
| 17,7 | g |
| 15,10 | b |
| 16,11 | a |
| 18,12 | f |
| 13 | Digit 2 enable |
| 4 | Decimal point 1 |
| 9 | Decimal point 2 |

The segment configuration of the DC56-11EWA display is shown in Figure 5.49. In a multiplexed display application, the segment pins of the corresponding segments are connected together. For example, pins 11 and 16 are connected as the common **a** segment. Similarly, pins 15 and 10 are connected as the common **b** segment, and so on.

### *Project Hardware*

The block diagram of this project is shown in Figure 5.50. The circuit diagram is given in Figure 5.51. The segments of the display are connected to PORTD of a PIC18F45K22-type microcontroller, operated with an 8-MHz crystal. Current limiting resistors are used on each segment of the display. Each digit is enabled using a BC108-type transistor switch connected to RA0 and RA1 port pins of the microcontroller. A segment is turned on when a logic 1 is applied to the base of the corresponding segment transistor.

If you are using the EasyPIC V7 development board, then make sure that the following jumpers are configured to enable two digits of the 7-segment display:

DIP switch SW4: DIS0 and DIS1 enabled to RA0 and RA1, respectively

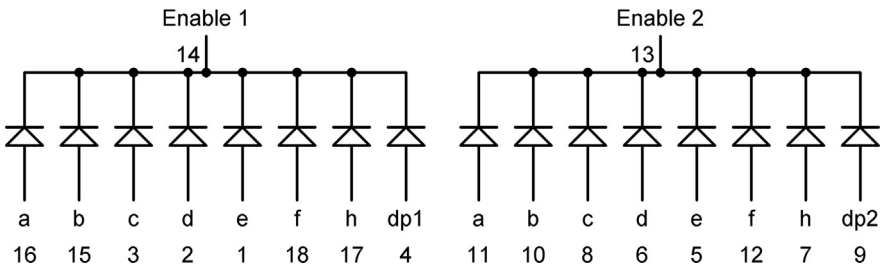


**Figure 5.49: The DC56-11EWA Display Segment Configuration.**

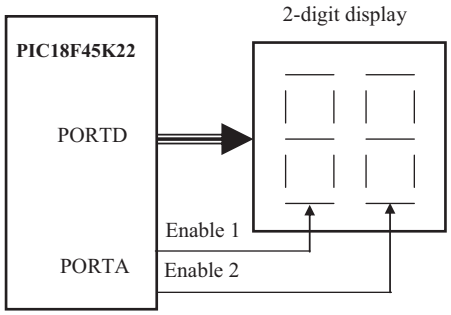


**Figure 5.50: Block Diagram of the Project.**

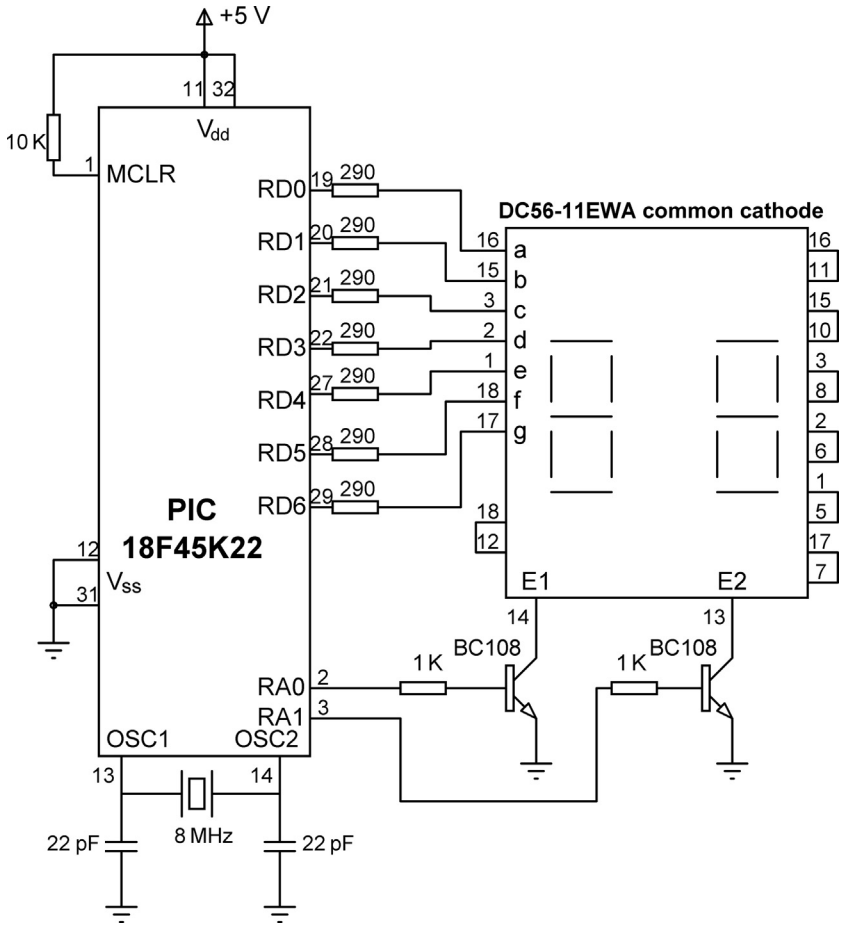


**Figure 5.51: Circuit Diagram of the Project.**

```
BEGIN
    Create SEGMENT table
    Configure PORTA as digital outputs
    Configure PORTD as digital outputs
    Initialize CNT to 25
    DO FOREVER
        Find MSD digit
        Get bit pattern from SEGMENT
        Enable digit 2
        Wait for a while
        Disable digit 2
        Find LSD digit
        Get bit pattern from SEGMENT
        Enable digit 1
        Wait for a while
        Disable digit 1
    ENDDO
END
```

**Figure 5.52: PDL of the Project.**

### *Project PDL*

At the beginning of the program, PORTA and PORTD pins are configured as outputs. The program then enters an endless loop where first of all the most significant digit (MSD) of the number is calculated, function **Display** is called to find the bit pattern, and then sent to the display and digit 1 is enabled. Then, after a small delay digit 1 is disabled, the LSD of the number is calculated, function **Display** is called to find the bit pattern and then sent to the display, and digit 2 is enabled. Then, again after a small delay, digit 2 is disabled and the above process repeats indefinitely. Figure 5.52 shows the PDL of the project.

### *Project Program*

#### *mikroC Pro for PIC*

The mikroC Pro for the PIC program is called MIKROC-LED9.C, and the program listing is given in Figure 5.53. **DIGIT1** and **DIGIT2** are defined to be equal to bit 0 and bit 1 of PORTA, respectively. The value to be displayed (number 25) is stored in variable **Cnt**. An endless loop is formed using a **for** statement. Inside the loop, the MSD of the number is calculated by dividing the number by 10. Function **Display** is then called to find the bit pattern to send to PORTD. Then, digit 2 is enabled by setting DIGIT2 = 1, and the program waits for 10 ms. After this, digit 2 is disabled, and the LSD of the number is calculated using the mod operator ("**%**") and sent to PORTD. At the same time, digit 1 is enabled by setting DIGIT1 = 1, and the program waits for 10 ms. After this time, digit 1 is disabled, and the program repeats forever.

```
/*************************************************************************
                              Dual 7-SEGMENT DISPLAY
                              ======================

In this project two common cathode 7-segment LED displays are connected to PORTD of a
PIC18F45K22 microcontroller and the microcontroller is operated from an 8 MHz crystal.
Digit 1 (right digit) enable pin is connected to port pin RA0 and digit 2 (left digit) enable pin is
connected to port pin RA1 of the microcontroller. The program displays number 25 on the displays.

Author:         Dogan Ibrahim
Date:           August 2013
File:           MIKROC-LED9.C
*************************************************************************/
#define DIGIT1 PORTA.RA0
#define DIGIT2 PORTA.RA1

//
// This function finds the bit pattern to be sent to the port to display a number
// on the 7-segment LED. The number is passed in the argument list of the function.
//
unsigned char Display(unsigned char no)
{
  unsigned char Pattern;
  unsigned char SEGMENT[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,
                    0x7D,0x07,0x7F,0x6F};

  Pattern = SEGMENT[no];                    // Pattern to return
  return (Pattern);
}


//
// Start of MAIN Program
//
void main()
{
   unsigned char Msd, Lsd, Cnt = 25;

   ANSELA = 0;                              // Configure PORTA as digital
   ANSELD = 0;                              // Configure PORTD as digital
   TRISA = 0;                               // Configure PORTA as outputs
   TRISD = 0;                               // Configure PORTD as outputs

   DIGIT1 = 0;                              // Disable digit 1
   DIGIT2 = 0;                              // Disable digit 2

   for(;;)                                  // Endless loop
   {
     Msd = Cnt/10;                          // MSD digit
     PORTD = Display(Msd);                  // Send to PORTD
     DIGIT2 = 1;                            // Enable digit 2
```

**Figure 5.53: mikroC Pro for PIC Program Listing.**

```
        Delay_Ms(10);                   // Wait a while

        DIGIT2 = 0;                     // Disable digit 2
        Lsd = Cnt % 10;                 // LSD digit
        PORTD = Display(Lsd);           // Send to PORTD
        DIGIT1 = 1;                     // Enable digit 1
        Delay_Ms(10);                   // Wait a while
        DIGIT1 = 0;                     // Disable digit 1
    }
}
```

**Figure 5.53**
cont'd

*MPLAB XC8*

The MPLAB X8 version of the program is shown in Figure 5.54 (XC8-LED9.C). The operation of the program is as in mikroC Pro for PIC.

## Project 5.10—Four-Digit Multiplexed 7-Segment LED

### Project Description

This project is similar to that in the previous project, but here, multiplexed four digits are used instead of two. The display digits are enabled and disabled as in the two-digit example.

### Project Hardware

The block diagram of this project is shown in Figure 5.55. The circuit diagram is given in Figure 5.56. The segments of the display are connected to PORTD of a PIC18F45K22 type microcontroller, operated with an 8-MHz crystal. Current limiting resistors are used on each segment of the display. Each digit is enabled using a BC108 type transistor switch connected to RA0:RA3 port pins of the microcontroller. A segment is turned on when logic 1 is applied to the base of the corresponding segment transistor.

If you are using the EasyPIC V7 development board, then make sure that the following jumpers are configured to enable four digits of the 7-segment display:

DIP switch SW4: DIS0:DIS3 enabled to RA0:RA3 respectively

### Project PDL

Figure 5.57 shows the PDL of the project. At the beginning of the program, DIGIT1−DIGIT4 connections are defined. Then, PORTA and PORTD are configured as digital outputs, and all the digits are disabled. The program executes in an endless *for*

```
/*************************************************************************
                              Dual 7-SEGMENT DISPLAY
                              ======================

In this project two common cathode 7-segment LED displays are connected to PORTD of a
PIC18F45K22 microcontroller and the microcontroller is operated from an 8 MHz crystal
Digit 1 (right digit) enable pin is connected to port pin RA0 and digit 2 (left digit) enable pin
is connected to port pin RA1 of the microcontroller. The program displays number 25 on the
displays.

Author:         Dogan Ibrahim
Date:           August 2013
File:           XC8-LED9.C
*************************************************************************/

#include <xc.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

#define DIGIT1 PORTAbits.RA0
#define DIGIT2 PORTAbits.RA1

//
// This function finds the bit pattern to be sent to the port to display a number on the
// 7-segment LED. The number is passed in the argument list of the function.
//
unsigned char Display(unsigned char no)
{
  unsigned char Pattern;
  unsigned char SEGMENT[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,
                    0x7D,0x07,0x7F,0x6F};

  Pattern = SEGMENT[no];                    // Pattern to return
  return (Pattern);
}


void main()
{
   unsigned char Msd, Lsd, Cnt = 25;

   ANSELA = 0;                              // Configure PORTA as digital
   ANSELD = 0;                              // Configure PORTD as digital
   TRISA = 0;                               // Configure PORTA as outputs
   TRISD = 0;                               // Configure PORTD as outputs

   DIGIT1 = 0;                              // Disable digit 1
   DIGIT2 = 0;                              // Disable digit 2

   for(;;)                                  // Endless loop
   {
```

**Figure 5.54: MPLAB XC8 Program Listing.**

```
        Msd = Cnt/10;                    // MSD digit
        PORTD = Display(Msd);            // Send to PORTD
        DIGIT2 = 1;                      // Enable digit 2
        __delay_ms(10);                  // Wait a while

        DIGIT2 = 0;                      // Disable digit 2
        Lsd = Cnt % 10;                  // LSD digit
        PORTD = Display(Lsd);            // Send to PORTD
        DIGIT1 = 1;                      // Enable digit 1
        __delay_ms(10);                  // Wait a while
        DIGIT1 = 0;                      // Disable digit 1
    }
}
```

**Figure 5.54**
cont'd

loop. Inside this loop, the digits of the number to be displayed (1234) are extracted, sent to PORTD, and the corresponding digits are enabled.

Function *Display* extracts the bit pattern corresponding to a number. Each digit is displayed for 5 ms.

### *Project Program*

#### *mikroC Pro for PIC*

The mikroC Pro for the PIC program is called MIKROC-LED10.C and the program listing is given in Figure 5.58. **DIGIT1–DIGIT4** are defined to be equal to bit 0–bit 3 of PORTA, respectively. The value to be displayed (number 1234 in this example) is stored in variable **Cnt**. An endless loop is formed using a **for** statement. Inside this loop, a digit is

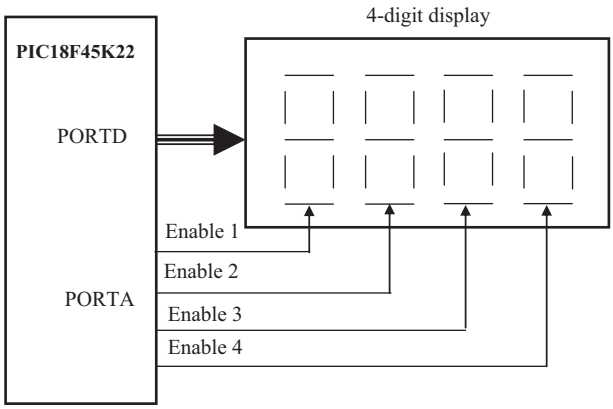


**Figure 5.55: Block Diagram of the Project.**

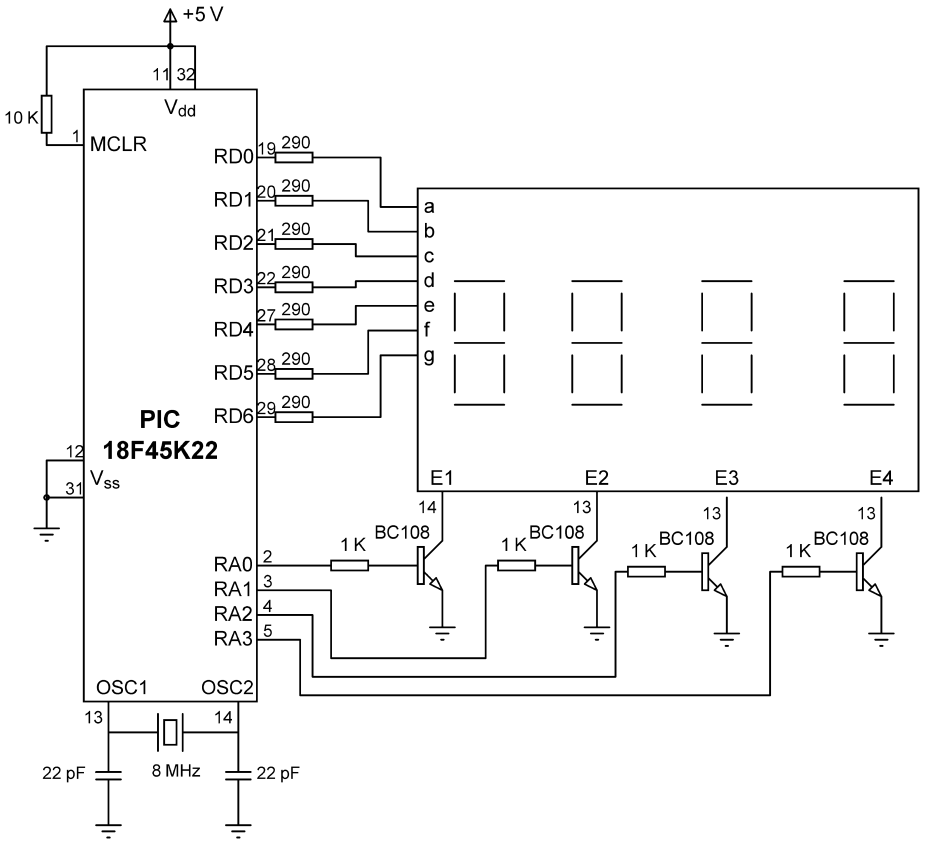


**Figure 5.56: Circuit Diagram of the Project.**

extracted, sent to the display, and then the corresponding digit is enabled. After a 5-ms delay, the digit is disabled, and the next digit is processed.

The digits are stored in the following variables:

| | |
|---|---|
| D1 | 1000s digit |
| D3 | 100s digit |
| D5 | 10s digit |
| D6 | 1s digit |

### *MPLAB XC8*

The MPLAB X8 version of the program is shown in Figure 5.59 (XC8-LED10.C). The operation of the program is as in mikroC Pro for PIC.

```
BEGIN
        Define digits
        Configure PORTA as digital outputs
        Configure PORTD as digital outputs
        Initialize CNT to 1234
        DO FOREVER
                Find 1000s digit
                CALL Display to get bit pattern
                Enable digit 4
                Wait for a while
                Disable digit 4
                Find 100s digit
                CALL Display to get bit pattern
                Enable digit 3
                Wait for a while
                Disable digit 3
                Find 10s digit
                CALL Display to get bit pattern
                Enable digit 2
                Wait for a while
                Disable digit 2
                Find 1s digit
                CALL Display to get bit pattern
                Enable bit 1
                Wait for a while
                Disable digit 1
        ENDDO
END


BEGIN/Display
        Create Segment table
        Find the bit pattern corresponding to the number
        Return the bit pattern
END/Display
```

**Figure 5.57: PDL of the Project.**

## *Project 5.11—LED Voltmeter*

### *Project Description*

In this project, a voltmeter with an LED display is designed. The voltmeter can be used to measure voltages 0−5 V. The voltage to be measured is applied to one of the analog inputs of a PIC18F45K22 microcontroller. The microcontroller reads the analog voltage, converts into digital, formats it, and then turns ON one of the five LEDs to indicate the applied voltage range.

Figure 5.60 shows the block diagram of the project.

### *Project Hardware*

The circuit diagram is given in Figure 5.61. The LEDs are connected to PORTD, and analog input RA0 (AN0, channel 0) is used to read the input voltage.

```
/*************************************************************************
                        4-DIGIT 7-SEGMENT DISPLAY
                        =========================

In this project four common cathode 7-segment LED displays are connected to PORTD of a
PIC18F45K22 microcontroller and the microcontroller is operated from an 8 MHz crystal.
Four PORTA pins are used to enable/disable the LEDs. The program displays number 1234.


Author:        Dogan Ibrahim
Date:          August 2013
File:          MIKROC-LED10.C
*************************************************************************/
#define DIGIT1 PORTA.RA0
#define DIGIT2 PORTA.RA1
#define DIGIT3 PORTA.RA2
#define DIGIT4 PORTA.RA3

//
// This function finds the bit pattern to be sent to the port to display a number
// on the 7-segment LED. The number is passed in the argument list of the function.
//
unsigned char Display(unsigned char no)
{
  unsigned char Pattern;
  unsigned char SEGMENT[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};

  Pattern = SEGMENT[no];                          // Pattern to return
  return (Pattern);
}


//
// Start of MAIN Program
//
void main()
{
   unsigned int Cnt = 1234;
   unsigned int D1,D2,D3,D4,D5,D6;
   ANSELA = 0;                                    // Configure PORTA as digital
   ANSELD = 0;                                    // Configure PORTD as digital
   TRISA = 0;                                     // Configure PORTA as outputs
   TRISD = 0;                                     // Configure PORTD as outputs

   DIGIT1 = 0;                                    // Disable digit 1
   DIGIT2 = 0;                                    // Disable digit 2
   DIGIT3 = 0;                                    // Disable digit 3
   DIGIT4 = 0;                                    // Disable digit 4

   for(;;)                                        // Endless loop
   {
```

**Figure 5.58: mikroC Pro for PIC Program Listing.**

```
        D1 = Cnt/1000;                          // 1000s  digit
        PORTD = Display(D1);                    // Send to PORTD
        DIGIT4 = 1;                             // Enable digit 4
        Delay_Ms(5);                            // Wait a while
        DIGIT4 = 0;                             // Disable digit 4

        D2 = Cnt % 1000;
        D3 = D2/100;                            // 100s digit
        PORTD = Display(D3);                    // Send to PORTD
        DIGIT3 = 1;                             // Enable digit 3
        Delay_Ms(5);                            // Wait a while
        DIGIT3 = 0;                             // Disable digit 3

        D4 = D2 % 100;
        D5 = D4/10;                             // 10s digit
        PORTD = Display(D5);                    // Send to PORTD
        DIGIT2 = 1;                             // Enable digit 2
        Delay_Ms(5);                            // Wait a while
        DIGIT2 = 0;                             // Disable digit 2

        D6 = D4 % 10;
        PORTD = Display(D6);                    // Send to PORTD
        DIGIT1 = 1;                             // Enable digit 1
        Delay_Ms(5);                            // Wait a while
        DIGIT1 = 0;                 /           / Disable digit 1

    }
}
```

**Figure 5.58**
cont'd

If you are using the EasyPIC V7 development board, then make sure that the following jumper is connected so that the potentiometer can be enabled at the RA0 input:

Jumper J15 (ADC INPUT): Connect RA0 inputs

### *Project PDL*

Figure 5.62 shows the PDL of the project. At the beginning of the program, PORTD is configured as a digital output, and RA0 is configured as analog input. Then, an endless loop is formed, and the applied voltage is read and converted into analog millivolts. The corresponding LED is then turned ON. This process repeats continuously with a 10-ms delay between each iteration.

### *Project Program*

#### *mikroC Pro for PIC*

The mikroC Pro for PIC program is called MIKROC-LED11.C, and the program listing is given in Figure 5.63. At the beginning of the program, the LEDs are given symbols to

```
/*************************************************************************
                              4-DIGIT 7-SEGMENT DISPLAY
                              =========================

In this project two common cathode 7-segment LED displays are connected to PORTD
of a PIC18F45K22 microcontroller and the microcontroller is operated from an 8 MHz crystal.
Four PORTA pins are used to enable/disable the LEDs.

The program displays number 1234.


Author:        Dogan Ibrahim
Date:          August 2013
File:          XC8-LED10.C
*************************************************************************/

#include <xc.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

#define DIGIT1 PORTAbits.RA0
#define DIGIT2 PORTAbits.RA1
#define DIGIT3 PORTAbits.RA2
#define DIGIT4 PORTAbits.RA3

//
// This function finds the bit pattern to be sent to the port to display a number
// on the 7-segment LED. The number is passed in the argument list of the function.
//
unsigned char Display(unsigned char no)
{
   unsigned char Pattern;
   unsigned char SEGMENT[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};

   Pattern = SEGMENT[no];                          // Pattern to return
   return (Pattern);
}


void main()
{
    unsigned int Cnt = 1234;
    unsigned int D1,D2,D3,D4,D5,D6;
    ANSELA = 0;                                    // Configure PORTA as digital
    ANSELD = 0;                                    // Configure PORTD as digital
    TRISA = 0;                                     // Configure PORTA as outputs
    TRISD = 0;                                     // Configure PORTD as outputs

    DIGIT1 = 0;                                    // Disable digit 1
    DIGIT2 = 0;                                    // Disable digit 2
    DIGIT3 = 0;                                    // Disable digit 3
```

**Figure 5.59: MPLAB XC8 Program Listing.**

```
    DIGIT4 = 0;                                  // Disable digit 4

    for(;;)                                      // Endless loop
    {
      D1 = Cnt/1000;                             // 1000s  digit
      PORTD = Display(D1);                       // Send to PORTD
      DIGIT4 = 1;                                // Enable digit 4
      __delay_ms(5);                             // Wait a while
      DIGIT4 = 0;                                // Disable digit 4

      D2 = Cnt % 1000;
      D3 = D2/100;                               // 100s digit
      PORTD = Display(D3);                       // Send to PORTD
      DIGIT3 = 1;                                // Enable digit 3
      __delay_ms(5);                             // Wait a while
      DIGIT3 = 0;                                // Disable digit 3

      D4 = D2 % 100;
      D5 = D4/10;                                // 10s digit
      PORTD = Display(D5);                       // Send to PORTD
      DIGIT2 = 1;                                // Enable digit 2
      __delay_ms(5);                             // Wait a while
      DIGIT2 = 0;                                // Disable digit 2

      D6 = D4 % 10;
      PORTD = Display(D6);                       // Send to PORTD
      DIGIT1 = 1;                                // Enable digit 1
      __delay_ms(5);                             // Wait a while
      DIGIT1 = 0;                                // Disable digit 1

    }
}
```

**Figure 5.59**
cont'd

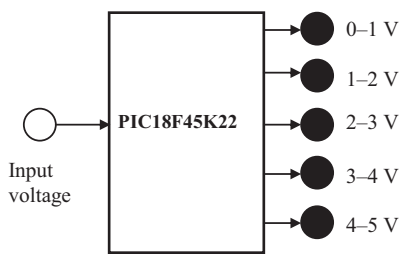


**Figure 5.60: Block Diagram of the Project.**

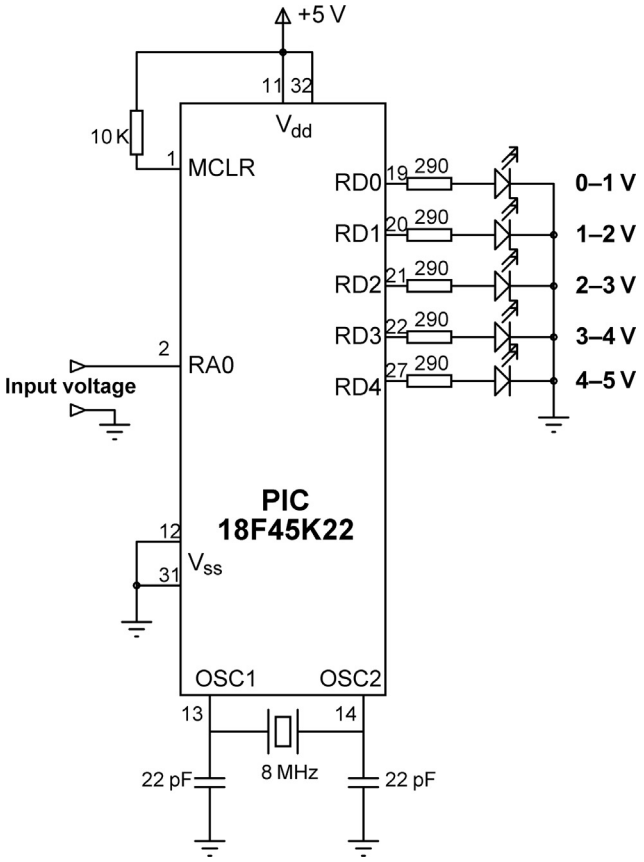


**Figure 5.61: Circuit Diagram of the Project.**

```
BEGIN
    Define LED connections with symbols
    Configure PORTD as digital outputs
    Configure RA0 as analog input
    Configure the A/D converter
    DO FOREVER
        Read analog data (voltage) from channel 0
        Convert the data into millivolts
        Display the data on one of the LEDs
        Wait 10 ms
    ENDO
END
```

**Figure 5.62: PDL of the Project.**

```
/*************************************************************************
                        VOLTMETER WITH LED DISPLAYS
                        ===========================

In this project 5 LEDs are connected to PORTD. Also, input port RA0 (AN0) is used as analog input.
Voltage to be measured is applied to this pin. The microcontroller reads the analog voltage,
converts into digital, and then turns ON one of the LEDs to indicate the voltage range.

Analog input range is 0 to 5 V. A PIC18F45K22 type microcontroller is used in this project,
operated with an 8 MHz cystal.

Analog data is read using the mikroC Pro for PIC built-in function Adc_Read.


Author:         Dogan Ibrahim
Date:           August 2013
File:           MIKROC-LED11.C
*************************************************************************/
#define LED01V PORTD.RD0
#define LED12V PORTD.RD1
#define LED23V PORTD.RD2
#define LED34V PORTD.RD3
#define LED45V PORTD.RD4

void main()
{
   unsigned long Vin, mV;

   ANSELD = 0;                          // Configure PORTD as digital
   ANSELA = 1;                          // COnfigure RA0 as analog
   TRISD = 0;                           // Configure PORTD as outputs
   TRISA = 1;                           // Configure RA0 as input

//
// Configure A/D converter. Channel 0 (RA0, or AN0) is used in this project.
//
   ADCON2 = 0x80;                       // Right justify the result

   for(;;)                              // Endless loop
   {
     PORTD = 0;                         // Clear LEDs to start with
     Vin = Adc_Read(0);                 // Read from channel 0 (AN0)
     mV = (Vin * 5000) >> 10;           // mv = Vin x 5000/1024
//
// Find which LED to turn ON
//
     if(mV >= 0 && mV <= 1000)          // Between 1–2 V
     {
       LED01V = 1;                      // Turn ON LED0–1 V
     }
     else if(mV > 1000 && mV <= 2000)   // Between 2–3 V
```

**Figure 5.63: mikroC Pro for PIC Program Listing.**

```
    {
      LED12V = 1;                          // Turn ON LED1–2 V
    }
    else if(mV > 2000 && mV <= 3000)       // Between 2–3 V
    {
      LED23V = 1;                          // Turn ON LED2–3 V
    }
    else if(mv > 3000 && mV <= 4000)       // Between 3–4 V
    {
      LED34V = 1;                          // Turn ON LED3–4 V
    }
    else if(mV > 4000 && mV <= 5000)       // Between 4–5 V
    {
      LED45V = 1;                          // Turn ON LED4–5 V
    }
    Delay_Ms(10);                          // 10 ms delay
  }
}
```

**Figure 5.63**
cont'd

make it easy to identify them. Then, PORTD is configured as a digital output and RA0 is configured as an analog input. The A/D converter is configured to right justify the result. The program executes in an endless loop established using a for statement. Inside this loop, analog data are read from channel 0 using the **Adc_Read(0)** built-in function. The converted digital data are stored in variable **Vin**, which is declared as an **unsigned long**. The A/D converter is 10 bits wide, and thus, there are 1024 steps (0−1023) corresponding to the reference voltage of 5000 mV. Each step corresponds to 5000 mV/1024 = 4.88 mV. Inside the loop variable, **Vin** is converted into millivolts after multiplying by 5000 and dividing into 1024. The division is done by shifting right by 10 digits. At this point, variable **mV** contains the converted input voltage in millivolts. A number of conditional statements are then used to turn ON the required LED.

*MPLAB XC8*

The MPLAB X8 version of the program is shown in Figure 5.64 (XC8-LED11.C). The A/D converter process is slightly more complex. XC8 supports the following A/D converter functions (see *PIC18 Peripheral Library Help Document*).

**OpenADC**: This function is used to configure the A/D module. The number of arguments required depends on the type of microcontroller used (see *PIC18 Peripheral Library Help Document*). PIC18F45K22 microcontroller requires three arguments. The bits of an argument should be separated with the AND operator (&), and the arguments should be separated using commas.

```
/*********************************************************************************
                                VOLTMETER WITH LED DISPLAYS
                                ===========================

In this project 5 LEDs are connected to PORTD. Also, input port RA0 (AN0) is used as analog
input. Voltage to be measured is applied to this pin. The microcontroller reads the analog
voltage, converts into digital, and then turns ON one of the LEDs to indicate the voltage range.

Analog input range is 0 to 5 V. A PIC18F45K22 type microcontroller is used in this project,
operated with an 8 MHz cystal.

Analog data is read using the MPLAB XC8 built-in functions.


Author:         Dogan Ibrahim
Date:           August 2013
File:           XC8-LED11.C
*********************************************************************************/
#include <xc.h>
#include <plib/adc.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

#define LED01V PORTDbits.RD0
#define LED12V PORTDbits.RD1
#define LED23V PORTDbits.RD2
#define LED34V PORTDbits.RD3
#define LED45V PORTDbits.RD4

void main()
{
   unsigned long Vin, mV;

   ANSELD = 0;                                  // Configure PORTD as digital
   ANSELA = 1;                                  // COnfigure RA0 as analog
   TRISD = 0;                                   // Configure PORTD as outputs
   TRISA = 1;                                    // Configure RA0 as input
//
// Configure A/D converter
//
  OpenADC(ADC_FOSC_2 & ADC_RIGHT_JUST & ADC_12_TAD,
       ADC_CH0 & ADC_INT_OFF,
       ADC_TRIG_CTMU & ADC_REF_VDD_VDD & ADC_REF_VDD_VSS);

   for(;;)                                      // Endless loop
   {
     PORTD = 0;                                 // Clear LEDs to start with
     SelChanConvADC(ADC_CH0);                   // Select channel 0 and start conversion
     while(BusyADC());                          // Wait for completion
     Vin = ReadADC();                           // Rea converted data
     mV = (Vin * 5000) >> 10;                   // mv = Vin x 5000/1024
```

**Figure 5.64: MPLAB XC8 Program Listing.**

```
//
// Find which LED to turn ON
//
      if(mV <= 1000)                          // Between 1–2 V
      {
        LED01V = 1;                           // Turn ON LED0–1 V
      }
      else if(mV > 1000 && mV <= 2000)        // Between 2–3 V
      {
        LED12V = 1;                           // Turn ON LED1–2 V
      }
      else if(mV > 2000 && mV <= 3000)        // Between 2–3 V
      {
       LED23V = 1;                            // Turn ON LED2–3 V
      }
      else if(mV > 3000 && mV <= 4000)        // Between 3–4 V
      {
       LED34V = 1;                            // Turn ON LED3–4 V
      }
      else if(mV > 4000 && mV <= 5000)        // Between 4–5 V
      {
       LED45V = 1;                            // Turn ON LED4–5 V
      }
      __delay_ms(10);                         // 10 ms delay
    }
}
```

**Figure 5.64**
cont'd

The following arguments are valid for the PIC18F45K22 microcontroller:

**Argument 1:**

A/D clock source
- * ADC_FOSC_2
- * ADC_FOSC_4
- * ADC_FOSC_8
- * ADC_FOSC_16
- * ADC_FOSC_32
- * ADC_FOSC_64
- * ADC_FOSC_RC
- * ADC_FOSC_MASK

A/D result justification
- * ADC_RIGHT_JUST
- * ADC_LEFT_JUST
- * ADC_RESULT_MASK

A/D acquisition time select
* ADC_0_TAD
* ADC_2_TAD
* ADC_4_TAD
* ADC_6_TAD
* ADC_12_TAD
* ADC_16_TAD
* ADC_20_TAD
* ADC_TAD_MASK

**Argument 2:**

Channel
* ADC_CH0
* ADC_CH1
* ADC_CH2
* ADC_CH3
* ADC_CH4
* ADC_CH5
* ADC_CH6
* ADC_CH7
* ADC_CH8
* ADC_CH9
* ADC_CH10
* ADC_CH11
* ADC_CH12
* ADC_CH13

ADC_CH14
* ADC_CH15
* ADC_CH16
* ADC_CH17
* ADC_CH18
* ADC_CH19
* ADC_CH20
* ADC_CH21
* ADC_CH22
* ADC_CH23
* ADC_CH24
* ADC_CH25
* ADC_CH26
* ADC_CH27

* ADC_CH_CTMU
* ADC_CH_DAC
* ADC_CH_FRV

A/D Interrupts

* ADC_INT_ON
* ADC_INT_OFF
* ADC_INT_MASK

**Argument 3:**

Special Trigger Select

* ADC_TRIG_CTMU
* ADC_TRIG_CCP5

A/D VREF + Configuration

* ADC_REF_VDD_VDD
* ADC_REF_VDD_VREFPLUS
* ADC_REF_FVR_BUF

A/D VREF− Configuration

* ADC_REF_VDD_VSS
* ADC_REF_VDD_VREFMINUS

| | |
|---|---|
| **SetChanADC**: | This function selects the channel to be used for the A/D converter. |
| **SetChanConvADC**: | This function selects the channel to be used and at the same time starts the conversion. |
| **ConvertADC**: | This function starts the A/D conversion. |
| **BusyADC**: | This function returns the A/D conversion status. |
| **ReadADC**: | This function returns the A/D result. |
| **CloseADC**: | This function turns off the A/D converter module. |

The MPLAB XC8 A/D conversion process is carried out as follows:

- Configure the A/D converter (OpenADC).
- Select channel to be used (SelChanConvADC).
- Wait until the conversion is complete (BusyADC).
- Read converted data (ReadADC).

## *Project 5.12—LCD Voltmeter*

### *Project Description*

In this project, the design of a voltmeter with an LCD display is described. The voltmeter can be used to measure voltages in the range 0−5 V. The voltage to be measured is applied to one of the analog inputs of a PIC18F45K22-type microcontroller. The microcontroller reads the analog voltage, converts into digital, and then displays on an LCD.

In microcontroller systems, the output of a measured variable is usually displayed using LEDs, 7-segment displays, or LCD-type displays. LCDs have the advantages that they can be used to display alphanumeric or graphical data. Some LCDs have ≥40 character lengths with the capability to display several lines. Some other LCD displays can be used to display graphics images. Some modules offer color displays while some others incorporate back lighting so that they can be viewed in dimly lit conditions.

There are basically two types of LCDs as far as the interface technique is concerned: parallel LCDs and serial LCDs. Parallel LCDs (e.g. Hitachi HD44780) are connected to a microcontroller using more than one data line, and the data are transferred in parallel form. It is common to use either four or eight data lines. Using a four-wire connection saves I/O pins, but it is slower since the data are transferred in two stages. Serial LCDs are connected to the microcontroller using only one data line, and data are usually sent to the LCD using the standard RS-232 asynchronous data communication protocol. Serial LCDs are much easier to use, but they cost more than the parallel ones do.

The programming of a parallel LCD is usually a complex task and requires a good understanding of the internal operation of the LCD controllers, including the timing diagrams. Fortunately, most high-level languages provide special library commands for displaying data on alphanumeric as well as on graphical LCDs. All the user has to do is connect the LCD to the microcontroller, define the LCD connection in the software, and then send special commands to display data on the LCD.

### HD44780 LCD Module

HD44780 is one of the most popular alphanumeric LCD modules used in the industry and also by hobbyists. This module is monochrome and comes in different sizes. Modules with 8, 16, 20, 24, 32, and 40 columns are available. Depending on the model chosen, the number of rows varies between 1, 2, or 4. The display provides a 14-pin (or 16-pin) connector to a microcontroller. Table 5.11 gives the pin configuration and pin functions of a 14-pin LCD module. Below is a summary of the pin functions:

$V_{SS}$ is the 0-V supply or ground. The $V_{DD}$ pin should be connected to the positive supply. Although the manufacturers specify a 5-V dc supply, the modules will usually work with as low as 3 V or as high as 6 V.

Pin 3 is named $V_{EE}$, and this is the contrast control pin. This pin is used to adjust the contrast of the display, and it should be connected to a variable voltage supply. A potentiometer is normally connected between the power supply lines with its wiper arm connected to this pin so that the contrast can be adjusted.

**Table 5.11: Pin Configuration of the HD44780 LCD Module**

| Pin Number | Name | Function |
|---|---|---|
| 1 | $V_{SS}$ | Ground |
| 2 | $V_{DD}$ | +ve supply |
| 3 | $V_{EE}$ | Contrast |
| 4 | RS | Register select |
| 5 | R/W | Read/write |
| 6 | E | Enable |
| 7 | D0 | Data bit 0 |
| 8 | D1 | Data bit 1 |
| 9 | D2 | Data bit 2 |
| 10 | D3 | Data bit 3 |
| 11 | D4 | Data bit 4 |
| 12 | D5 | Data bit 5 |
| 13 | D6 | Data bit 6 |
| 14 | D7 | Data bit 7 |

Pin 4 is the Register Select (RS), and when this pin is LOW, data transferred to the display are treated as commands. When RS is HIGH, character data can be transferred to and from the module.

Pin 5 is the Read/Write (R/W) line. This pin is pulled LOW to write commands or character data to the LCD module. When this pin is HIGH, character data or status information can be read from the module.

Pin 6 is the Enable (E) pin that is used to initiate the transfer of commands or data between the module and the microcontroller. When writing to the display, data are transferred only on the HIGH to LOW transition of this line. When reading from the display, data become available after the LOW to HIGH transition of the enable pin, and these data remain valid as long as the enable pin is at logic HIGH.

Pins 7–14 are the eight data bus lines (D0–D7). Data can be transferred between the microcontroller and the LCD module using either a single 8-bit byte, or as two 4-bit nibbles. In the latter case, only the upper four data lines (D4–D7) are used. Four-bit mode has the advantage that four less I/O lines are required to communicate with the LCD. In this book, we shall be using alphanumeric-based LCD only and look at the 4-bit interface only.

### *Connecting the LCD to the Microcontroller*

The following pins are used in 4-bit mode:

D4:D7
E
R/S

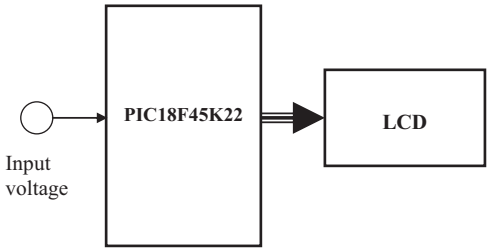


**Figure 5.65: Block Diagram of the Project.**

In this book, we shall be connecting the LCD to our microcontroller as in the following table:

| LCD Pin | Microcontroller Pin |
|---|---|
| D4 | RB0 |
| D5 | RB1 |
| D6 | RB2 |
| D7 | RB3 |
| R/S | RB4 |
| E | RB5 |

### *Project Hardware*

Figure 5.65 shows the block diagram of the project. The microcontroller reads the analog voltage, converts into digital, formats it, and then displays on the LCD.

The circuit diagram of the project is shown in Figure 5.66. The voltage to be measured (between 0 and 5 V) is applied to port AN0 where this port is configured as an analog input in software. The LCD is connected to PORTB of the microcontroller as described earlier. A potentiometer is used to adjust the contrast of the LCD display.

### *Project PDL*

The PDL of the project is shown in Figure 5.67. At the beginning of the program, PORTB is configured as the output, and PORTA is configured as the input. Then the LCD and the A/D converter are configured. The program then enters an endless loop where analog input voltage is converted into digital and displayed on the LCD. This process is repeated every second.

### *Project Program*

#### *mikroC Pro for PIC*

The mikroC Pro for PIC program listing (MIKROC-LCD1.C) is given in Figure 5.68. At the beginning of the program, the connections between the LCD and the microcontroller are defined using a number of *sbit* statements. PORTB is defined as the output and PORTA as the input. Then, the LCD is configured, and the text "VOLTMETER" is displayed on

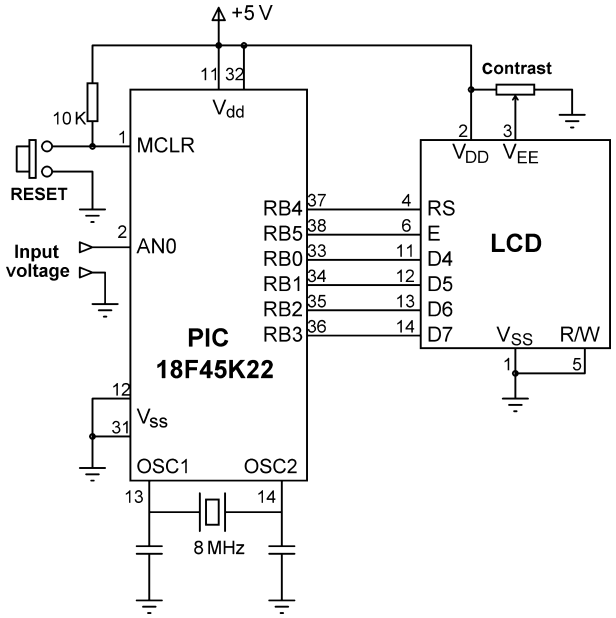


**Figure 5.66: Circuit Diagram of the Project.**

the LCD for 2 s. The A/D is configured by setting register ADCON1 to 0x80 so that the A/D result is right justified, $V_{ref}$ voltage is set to $V_{DD}$ (+5 V), and all PORTA pins are configured as analog inputs. The message "VOLTMETER" is displayed on the first row of the LCD for 2 s.

The main program loop starts with a **for** statement. Inside this loop, the LCD is cleared, analog data are read from channel 0 (pin AN0) using statement **Adc_Read(0)**. The converted digital data are stored in variable **Vin**, which is declared as an **unsigned long**. The A/D converter is 10 bits wide, and thus, there are 1024 steps (0−1023) corresponding to the reference voltage of 5000 mV. Each step corresponds to 5000 mV/1024 = 4.88 mV. Inside the loop, variable **Vin** is converted into millivolts by multiplying by 5000 and

```
BEGIN
    Configure PORTB as digital outputs
    Configure RA0 as input
    Configure the LCD
    Display heading
    Configure the A/D converter
    DO FOREVER
        Read analog data (voltage) from channel 0
        Format the data
        Display the data (voltage)
        Wait 1 s
    ENDO
END
```

**Figure 5.67: PDL of the Project.**

```
/************************************************************************
                        VOLTMETER WITH LCD DISPLAY
                        ==========================

In this project an LCD is connected to PORTB. Also, input port AN0 is used as analog input.
Voltage to be measured is applied to AN0. The microcontroller reads the analog voltage,
converts into digital, and then displays on the LCD.

Analog input range is 0 to 5 V. A PIC18F45K22 type microcontroller is used in this project,
operated with an 8 MHz crystal.

Analog data is read using the Adc_Read built-in function.

The LCD is connected to the microcontroller as follows:

Microcontroller     LCD
============     ===

   RB0              D4
   RB1              D5
   RB2              D6
   RB3              D7
   RB4              R/S
   RB5              E

Author:          Dogan Ibrahim
Date:            August 2013
File:            MIKROC-LCD1.C
************************************************************/
// LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End LCD module connections


void main()
{
   unsigned long Vin, mV;
   unsigned char op[12];
   unsigned char i,j,lcd[5];
```

**Figure 5.68: mikroC Pro for PIC Program Listing.**

```
    char *str;

    ANSELB = 0;                                   // Configure PORTB as digital
    ANSELA = 1;                                   // Configure RA0 as analog
    TRISB = 0;                                    // Configure PORTB as outputs
    TRISA = 1;                                    // Configure RA0 as input

//
// Configure LCD
//
    Lcd_Init();                                   // LCD is connected to PORTB
    Lcd_Cmd(_LCD_CLEAR);
    Lcd_cmd(_LCD_CURSOR_OFF);                     // Hide cursor
    Lcd_Out(1,1,"VOLTMETER");
    Delay_ms(2000);
//
// Configure A/D converter. AN0 is used in this project.
//
    ADCON1 = 0x80;                                // Use AN0 and Vref = +5 V
//
// Program loop
//
    for(;;)                                       // Endless loop
    {
      Lcd_Cmd(_LCD_CLEAR);                        // Clear LCD
      Vin = Adc_Read(0);                          // Read from channel 0 (AN0)
      Lcd_Out(1,1,"mV = ");                       // Display "mV = "
      mV = (Vin * 5000) >> 10;                    // mv = Vin x 5000/1024
      LongToStr(mV,op);                           // Convert to string in "op"
      str = Ltrim(op);                            // Remove leading spaces
//
// Display result on LCD
//
      Lcd_Out(1,6,str);                           // Output to LCD
      Delay_ms(1000);                             // Wait 1 s
    }
}
```

**Figure 5.68**
cont'd

dividing into 1024. The division is done by shifting right by 10 digits. At this point, variable **mV** contains the converted data in millivolts.

Function **LongToStr** is called to convert **mV** into a string in character array **op**. **LongToStr** converts a long variable into a string having a fixed width of 12 characters. If the resulting string is <12 characters, then the left of the data is filled with leading blanks. These leading blanks are removed using built-in function **Ltrim**, and the data are stored in character variable called **str**. Function **Lcd_Out** is called to display the data on the LCD starting from column 6 of row 1. For example, if the measured voltage is 1267 mV, it is displayed on the LCD as

mV = 1267

**Table 5.12: MPLAB XC8 LCD Functions**

| Function | Description |
|---|---|
| BusyXLCD | Check if the LCD controller is busy |
| OpenXLCD | Configure I/O port lines for the LCD and initialize |
| putcXLCD | Write a byte of data to the LCD |
| putsXLCD | Write a string from the data memory to the LCD |
| putrsXLCD | Write a string from the program memory to the LCD |
| ReadAddrXLCD | Read the address byte from the LCD controller |
| ReadDataXLCD | Read a byte from the LCD controller |
| SetCGRamAddr | Set the character generator address |
| SetDDRamAddr | Set the display data address |
| WriteCmdXLCD | Write a command to the LCD (the LCD must not be busy before sending a command) |

The above process is repeated after a 1-s delay.

*MPLAB XC8*

The MPLAB XC8 version of the program is slightly more complex because of the LCD functions. Table 5.12 gives a list of the MPLAB XC8 LCD functions available. Note that the header file "xlcd.h" must be included at the beginning of a program when any of these functions are used.

The LCD library requires that the following delay functions must be included in a program using the LCD library:

| | |
|---|---|
| **DelayFor18TCY** | Delay for 18 cycles |
| **DelayPORXLCD** | Delay for 15 ms |
| **DelayXLCD** | Delay for 5 ms |

Assuming a microcontroller clock frequency of 4 MHz, the instruction cycle time is 1 µs. With a clock frequency of 8 MHz, the instruction cycle time is 0.5 µs. Figure 5.69 shows how the above delay functions could approximately be obtained for both 4- and 8-MHz clock frequencies. The 18-cycle delay is obtained using NOP operations, where each NOP operation takes one cycle to execute. The end of a function with no "return" statement takes two cycles. When a "return" statement is used, a BRA statement branches to the end of the function where a RETURN 0 is executed to return from the function, thus adding two more cycles. For example, the following function takes four cycles to execute:

```
void test(void)
{
   nop();  ; 1 cycle
   nop();  ; 1 cycle
}          ; RETURN 0, takes 2 cycles
```

**4 MHz Clock**

```
#include<delays.h>

void DelayFor18TCY(void)
{
        Nop(); Nop(); Nop(); Nop();                    // 18 cycle delay
        Nop(); Nop(); Nop(); Nop();
        Nop(); Nop(); Nop(); Nop();
        Nop(); Nop();
        return;
}


void DelayPORXLCD(void)                                // 15 ms delay
{
        Delay1KTCYx(15);
}


void DelayXLCD(void)
{
        Delay1KTCYx(5);                                // 5 ms delay
}
```

**8 MHz Clock**

```
#include <delays.h>

void Delayfor18TCY(void)
{
        Nop(); Nop(); Nop(); Nop();                    // 18 cycle delay
        Nop(); Nop(); Nop(); Nop();
        Nop(); Nop(); Nop(); Nop();
        Nop(); Nop();
        return;
}


void DelayPORXLCD(void)                                // 15 ms delay
{
        Delay1KTCYx(30);
}


void DelayXLCD(void)
{
        Delay1KTCYx(10);                               // 5 ms delay
}
```

**Figure 5.69: LCD Delay Functions for 4- and 8-MHz Clock.**

and the following function takes six cycles to execute:

```
void test(void)
{
   nop();  ; 1 cycle
   nop();  ; 1 cycle
   return; ; BRA X, 2 cycles
}          ; X: RETURN 0, 2 cycles
```

Note that we could also use the *__delay_ms(15)* and *__delay_ms(5)* for the 15- and 5-ms delays, respectively, instead of the Delay1KTCYx() function.

A brief description of the commonly used C18 LCD functions is given below.

### *BusyXLCD*

This function checks to determine whether or not the LCD controller is busy and data or commands should not be sent to the LCD if the controller is busy. The function returns a 1 if the controller is busy, and 0 otherwise. The program can be forced to wait until the LCD controller is ready by using the following statement:

```
while(BusyXLCD());
```

Note that this function requires the RW pin of the LCD to be connected to the microcontroller (not to ground).

### *OpenXLCD*

This function is used to configure the interface between the microcontroller I/O ports and the LCD pins. The function requires an argument to specify the interface mode (4- or 8-bit), and the LCD character mode and number of lines used. A value should be selected and logically AND ed from the following two groups:

```
FOUR_BIT
EIGHT_BIT
LINE_5X7
LINE_5X10
LINES_5X7
```

LINES_5X7 is used for multiple row displays. For example, if we are using a four-wire connection with an LCD having a single row with 5x7 characters, then the function should be initialized as follows:

```
OpenXLCD(FOUR_BIT & LINE_5X7);
```

For a two-row display the initialization is

```
OpenXLCD(FOUR_BIT & LINES_5X7);
```

The actual physical connection between the LCD and microcontroller I/O ports is defined in file "xlcd.h" and the default settings use PORTB pins in the 4-bit mode where the low 4 bits of the port (RB0—RB3) are connected to the upper data lines (D4—D7) of the LCD (see the manual *MPLAB C18 C Compiler Libraries* for more information on the default connection).

The default connection between an LCD and the microcontroller is shown below. Note that E and RS are assumed to be connected to RB4 and RB5, respectively. This is the other way round if using the EasyPIC V7 development board. In addition, the RW pin is connected to the ground on the EasyPIC V7 development board.

| LCD Pin | Microcontroller Pin |
|---|---|
| E | RB4 |
| RS | RB5 |
| RW | RB6 |
| D4 | RB0 |
| D5 | RB1 |
| D6 | RB2 |
| D7 | RB3 |

The connection between an LCD and a microcontroller can be modified by editing the xlcd.h file. It will then be necessary to recompile the xlcd routines and add them to the microcontroller library of the target microcontroller.

### *putcXLCD*

This function is used to write a byte to the LCD. The byte is passed as an argument to the function. In the following example, character "A" is displayed on the LCD:

```
unsigned char x = 'A';
putcXLCD(x);
```

### *putsXLCD*

This function writes a string of characters from the data memory to the LCD. The writing stops when a NULL character is detected. An example use of this function is given below:

```
char txt[] = "My text";
putsXLCD(txt);
```

### *putrsXLCD*

This function writes a string of characters from the program memory to the LCD. The writing stops when a NULL character is detected. An example use of this function is given below:

```
putrsXLCD("My Computer");
```

*SerDDRamAddr*

This function sets the display data address. For example, using this function, we can write to a specified row and column of the LCD. The busy status of the LCD should be checked before calling this function.

Each character occupies one DDRAM address. The first row addresses are from 00 to 0x27, the second row addresses start from 0x40 to 0x67. For example, to move the cursor to the beginning of the second row, the DDRAM address will be 0x40, and the required command is

```
SetDDRamAddr(0x40);
```

*WriteCmdXLCD*

This function sends a command to the LCD. The following commands can be specified in the command argument:

| | |
|---|---|
| DOFF | Turn display off |
| CURSOR_OFF | Enable display, hide cursor |
| BLINK_ON | Enable cursor blinking |
| BLINK_OFF | Disable cursor blinking |
| SHIFT_CUR_LEFT | Shift cursor left |
| SHIFT_CUR_RIGHT | Shift cursor right |
| SHIFT_DISP_LEFT | Shift display to the left |
| SHIFT_DISP_RIGHT | Shift display to the right |

In addition, the LCD control functions given in Table 5.13 can be specified in the argument to control the LCD.

It is important that the LCD controller should not be busy (check with function BusyXLCD) when commands are sent to it. Some example commands are given below:

```
WriteCmdXLCD(BLINK_ON);  // Blink ON
WriteCmdXLCD(1);      // Clear LCD
```

**Table 5.13: LCD Functions**

| Command | Operation |
|---|---|
| 0 x 1 | Clear display |
| 0 x 2 | Home cursor |
| 0 x 0C | Cursor off |
| 0 x 0E | Underline cursor on |
| 0 x 0F | Blinking cursor on |
| 0 x 10 | Move the cursor left by one position |
| 0 x 14 | Move the cursor right by one position |
| 0 x 80 | Move the cursor to the beginning of the first row |
| 0 x C0 | Move the cursor to the beginning of the second row |
| 0 x 94 | Move the cursor to the beginning of the third row |
| 0 x D4 | Move the cursor to the beginning of the fourth row |

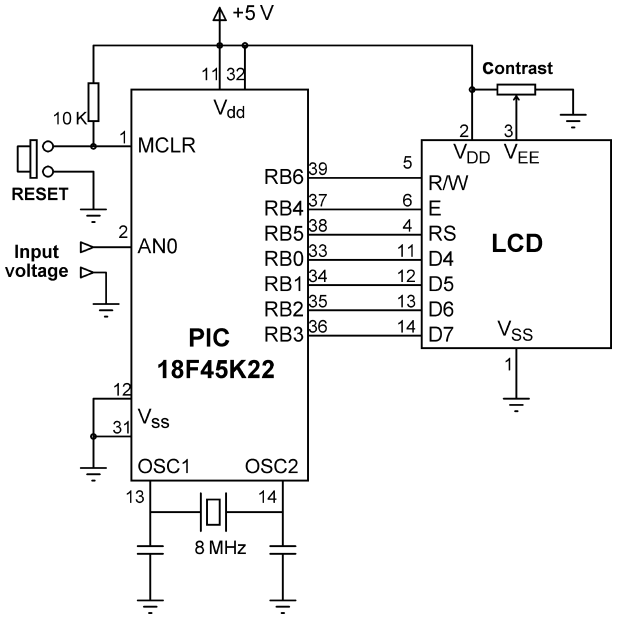


**Figure 5.70: Circuit Diagram of the Project.**

Figure 5.70 shows the circuit diagram of this project when used with the MPLAB XC8 compiler. Notice that this is similar to Figure 5.66 but the E and RS pins of the LCD are interchanged and also the RW pin is connected to pin RB6 of the microcontroller.

The MPLAB X8 version of the program for this project is shown in Figure 5.71 (XC8-LCD1.C). The following functions are used in the program:

| | |
|---|---|
| Delay_Seconds | create delay by the specified number of seconds |
| DelayFor18TCY | 18 cycles delay (required by the xlcd library) |
| DelayPORXLCD | 15 ms delay (required by the xlcd library) |
| DelayXLCD | 5 ms delay (required by the xlcd library) |
| LCD_Clear | Clear the LCD |
| LCD_Move | Move cursor to the specified row and column |

The program configures the LCD and the A/D converter. Then, the heading "VOLTMETER" is displayed, and the program enters an endless loop. Inside this loop, the external voltage at channel 0 is read and converted into millivolts. The voltage is then converted into ASCII string using MPLAB XC8 function *itoa* and is displayed starting from column 6 of the LCD. This process is repeated after a 1-s delay.

```
/*********************************************************************************
                                VOLTMETER WITH LCD DISPLAY
                                ==========================

In this project an LCD is connected to PORTB. Also, input port AN0 is used as analog input.
Voltage to be measured is applied to AN0. The microcontroller reads the analog voltage,
converts into digital, and then displays on the LCD.

Analog input range is 0 to 5 V. A PIC18F45K22 type microcontroller is used in this project,
operated with an 8 MHz crystal.

Analog data is read using the Adc_Read built-in function.

The LCD is connected to the microcontroller as follows:

Microcontroller          LCD
===============          ===

   RB0                   D4
   RB1                   D5
   RB2                   D6
   RB3                   D7
   RB4                   E
   RB5                   R/S
   RB6                   RW

Author:          Dogan Ibrahim
Date:            August 2013
File:            XC8-LCD1.C
*************************************************************/

#include <xc.h>
#include <stdlib.h>
#include <plib/adc.h>
#include <plib/xlcd.h>
#include <plib/delays.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000

//
// This function creates seconds delay. The argument specifies the delay time in seconds.
//
void Delay_Seconds(unsigned char s)
{
   unsigned char i,j;

   for(j = 0; j < s; j++)
   {
      for(i = 0; i <  100; i++)__delay_ms(10);
   }
}
```

**Figure 5.71: MPLAB XC8 Program Listing.**

```
//
// This function creates 18 cycles delay for the xlcd library
//
void DelayFor18TCY( void )
{
Nop(); Nop(); Nop(); Nop();
Nop(); Nop(); Nop(); Nop();
Nop(); Nop(); Nop(); Nop();
Nop(); Nop();
return;
}


//
// This function creates 15 ms delay for the xlcd library
//
void DelayPORXLCD( void )
{
   __delay_ms(15);
   return;
}


//
// This function creates 5 ms delay for the xlcd library
//
void DelayXLCD( void )
{
   __delay_ms(5);
   return;
}

//
// This function clears the screen
//
void LCD_Clear()
{
   while(BusyXLCD());
   WriteCmdXLCD(0x01);
}

//
// This function moves the cursor to position row,column
//
void LCD_Move(unsigned char row, unsigned char column)
{
 char ddaddr = 40*(row – 1) + column;
 while( BusyXLCD() );
 SetDDRamAddr( ddaddr );
}
```

**Figure 5.71**
cont'd

```
void main()
{
   unsigned long Vin, mV;
   char op[10];

   ANSELB = 0;                                   // Configure PORTB as digital
   ANSELA = 1;                                   // COnfigure RA0 as analog
   TRISB = 0;                                    // Configure PORTB as outputs
   TRISA = 1;                                    // Configure RA0 as input
//
// Configure the LCD to use 4-bits, in multiple display mode
//
   Delay_Seconds(1);
   OpenXLCD(FOUR_BIT & LINES_5X7);
//
// Configure the A/D converter
//
  OpenADC(ADC_FOSC_2 & ADC_RIGHT_JUST & ADC_20_TAD,
            ADC_CH0 & ADC_INT_OFF,
            ADC_TRIG_CTMU & ADC_REF_VDD_VDD & ADC_REF_VDD_VSS);

   while(BusyXLCD());                            // Wait if the LCD is busy
   WriteCmdXLCD(DON);                            // Turn Display ON
   while(BusyXLCD());                            // Wait if the LCD is busy
   WriteCmdXLCD(0x06);                           // Move cursor right
   putrsXLCD("VOLTMETER");                       // Display heading
   Delay_Seconds(2);                             // 2 s delay
   LCD_Clear();                                  // Clear display

   for(;;)                                       // Endless loop
   {
     LCD_Clear();                                // Clear LCD
     SelChanConvADC(ADC_CH0);                    // Select channel 0 and start conversion
     while(BusyADC());                           // Wait for completion
     Vin = ReadADC();                            // Read converted data
     mV = (Vin * 5000) >> 10;                    // mv = Vin x 5000/1024
     LCD_Move(1,1);                              // Move to row = 1, column = 1
     putrsXLCD("mV = ");                         // Display "mV = "
     mV = (Vin * 5000) >> 10;                    // mv = Vin x 5000/1024
     itoa(op, mV, 10);                           // Convert mV into ASCII string
//
// Display result on LCD
//
     LCD_Move(1,6);                              // Move to row = 1,column = 6
     putsXLCD(op);                               // Display the measured voltage
     Delay_Seconds(1);                           // Wait 1 s
   }
}
```

**Figure 5.71**
cont'd

## Project 5.13—Generating Sound

### Project Description

This project shows how sound with different frequencies can be generated using a simple buzzer. The project shows how the simple melody "Happy Birthday" can be played using a buzzer.

Figure 5.72 shows the block diagram of the project.

### Project Hardware

A buzzer is a small piezoelectric device that gives a sound output when excited. Normally, buzzers are excited using square wave signals, also called Pulse Width Modulated (PWM) signals. The frequency of the signal determines the pitch of the generated sound, and duty cycle of the signal can be used to increase or decrease the volume. Most buzzers operate in the frequency range 2−4 kHz. The PWM signal required to generate sound can be using the PWM module of the PIC microcontrollers.

In this project, a buzzer is connected to pin RC2 of a PIC18F45K22-type microcontroller through a transistor switch as shown in Figure 5.73.

If you are using the EasyPIC V7 development board, the following jumper should be selected:

J21: Select RC2

### Project PDL

When playing a melody, each note is played for a certain duration and with a certain frequency. In addition, a certain gap is necessary between two successive notes. The frequencies of the musical notes starting from middle C (i.e. C4) are given below. The harmonic of a note is obtained by doubling the frequency. For example, the frequency of C5 is $2 \times 262 = 524$ Hz.

| Notes | C4 | C4# | D4 | D4# | E4 | F4 | F4# | G4 | G4# | A4 | A4# | B4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hz | 261.63 | 277.18 | 293.66 | 311.13 | 329.63 | 349.23 | 370 | 392 | 415.3 | 440 | 466.16 | 493.88 |

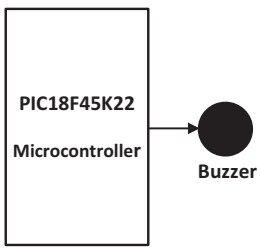


**Figure 5.72: Block Diagram of the Project.**

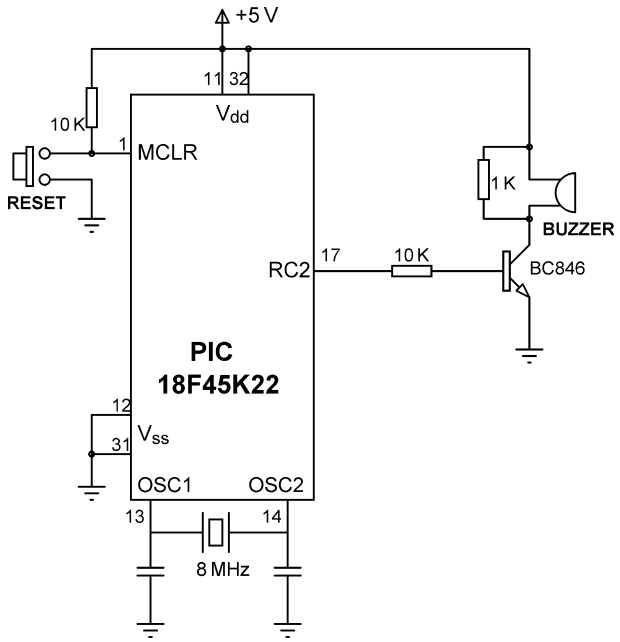


**Figure 5.73: Circuit Diagram of the Project.**

To play the tune of a song, we need to know its musical notes. Each note is played for a certain duration, and there is a certain time gap between two successive notes.

The next thing we want is to know how to generate a sound with a required frequency and duration. Fortunately, mikroC Pro for the PIC compiler provides a sound library with the following two functions:

| | |
|---|---|
| Sound_Init | This function is used to specify to which port and which pin the sound device is connected to |
| Sound_Play | This function generates a square wave signal with the specified frequency (hertz) and duration (milliseconds) from the port pin specified by the initialization function. The frequencies given in the above table are approximated to their nearest integers since this function accepts only integer frequencies. |

In this project, we will be generating the classic "Happy Birthday" melody, and thus, we need to know the notes and their durations. These are given in the table below where the durations are in milliseconds and should be multiplied by 400 to give correct values.

| **Note** | **C4** | **C4** | **D4** | **C4** | **F4** | **E4** | **C4** | **C4** | **D4** | **C4** | **G4** | **F4** | **C4** | **C4** | **C5** | **A4** | **F4** | **E4** | **D4** | **A4#** | **A4#** | **A4** | **F4** | **G4** | **F4** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Duration | 1 | 1 | 2 | 2 | 2 | 3 | 1 | 1 | 2 | 2 | 2 | 3 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 2 | 2 | 2 | 4 |

The PDL of the project is shown in Figure 5.74. Basically, two tables are used to store the notes and their corresponding durations. Then, the Sound_Play function is called in a loop to play all the notes. The melody repeats after a 3-s delay.

```
BEGIN
        Create Notes and Durations tables
        Initialize the Sound library
        DO FOREVER
                DO for all notes
                        Play the note with specified duration
                        Delay 100 ms
                ENDDO
        ENDDO
        Wait 3 s
END
```

**Figure 5.74: PDL of the Project.**

*mikroC Pro for PIC*

The mikroC Pro for the PIC program listing (MIKROC-SOUND1.C) is given in Figure 5.75. Tables *Notes* and *Durations* store the frequencies (hertz) and durations (1/400 ms) of each note, respectively. After initializing the sound library, an endless loop is established where inside this loop each note is played with the specified duration. A 100-ms gap is used between each note. The durations and the gap can be changed to increase or decrease the speed of the melody.

*MPLAB XC8*

The MPLAB XC8 compiler does not have a built-in sound library. We can however write our own function to generate sound. There are basically two ways in which we can generate accurate square waves for sounding the buzzer: using the processor built-in PWM module, or writing a timer interrupt service routine. The problem with the PWM module is that it is not possible to generate audio frequency signals using the PWM module. In this section, we shall see how to develop a timer interrupt service routine to generate square waves in the audio band.

The musical note frequencies we wish to generate are within the frequency range of several hundred to several thousand Hertz. We shall be using Timer0 in the 16-bit interrupt mode such that every time an interrupt is generated we shall toggle the output port pin connected to the buzzer. Thus, effectively, we will be generating square waves.

When TIMER0 is operating in the 16-bit mode, the value to be loaded into registers TMR0H and TMR0L to generate an interrupt after time T is given by

$$\text{TMR0H:TMR0L} = 65{,}536 - T/(4 \times T_{\text{osc}} \times \text{Prescaler value})$$

Our clock frequency is 8 MHz, which has the period of $T_{\text{osc}} = 0.125$ μs. If we give the Prescaler the value 2, then, the above equation becomes

$$\text{TMR0H:TMR0L} = 65{,}536 - T$$

```
/************************************************************************
                                PLAYING MELODY
                                ==============

In this project a buzzer is connected to port pin RC2 of a PIC18F45K22 microcontroller,
operating with an 8 MHz crystal.

The program plays the classical "Happy Birthday" melody.


Author:       Dogan Ibrahim
Date:         August 2013
File:         MIKROC-SOUND1.C
************************************************************/
#define Max_Notes 25

void main()
{
   unsigned char i;

   unsigned int Notes[Max_Notes] =
   {
    262, 262, 294, 262, 349, 330, 262, 262, 294, 262, 392,
    349, 262, 262, 524, 440, 349, 330, 294, 466, 466, 440,
    349, 392, 349
   };

   unsigned char Durations[Max_Notes] =
   {
    1, 1, 2, 2, 2, 3, 1, 1, 2, 2, 2, 3, 1, 1, 2, 2, 2, 2, 2,
    1, 1, 2, 2, 2, 3
   };

   ANSELC = 0;                                  // Configure PORTC as digital

   Sound_Init(&PORTC, 2);                       // Initialize the Sound library

   for(;;)                                      // Endless loop
   {
    for(i = 0; i < Max_Notes; i++)              // Do for all notes
    {
      Sound_Play(Notes[i], 400*Durations[i]);   // Play the notes
      Delay_ms(100);                            // Note gap
    }
    Delay_Ms(3000);                             // Repeat after 3 s
   }
}
```

**Figure 5.75: mikroC Pro for PIC Program Listing.**

The time $T$ to generate an interrupt depends upon the frequency $f$ of the waveform we wish to generate. When an interrupt occurs, if the output signal is at logic 1, then it is cleared to 0, if it is at logic 0, then it is set to 1. Assuming that we wish to generate square waves with equal ON and OFF times, the time $T$ in microseconds is then given by

$$T = 500{,}000/f$$

We can therefore calculate the value to be loaded into the timer registers as follows:

$$\text{TMR0H:TMR0L} = 65{,}536 - 500{,}000/f$$

For example, assuming that we wish to generate a square wave signal with frequency, $f = 100$ Hz, the value to be loaded into the Timer0 registers will be

$$\text{TMR0H:TMR0L} = 65{,}536 - 500{,}000/100 = 60{,}536$$

Here 60,536 is equivalent to 0xEC78 in hexadecimals. Thus, TMR0H = 0xEC and TMR0L = 0x78.

The MPLAB XC8 program to generate the square waves and play the melody is shown in Figure 5.76 (called XC8-SOUND1.C). The following functions are used in the program:

Interrupt isr: This is the interrupt service routine. Here, the timer registers are reloaded, the buzzer output is toggled, and the timer interrupt flag is cleared so that new timer interrupts can be accepted by the microcontroller.

| | |
|---|---|
| Delay_Ms | This function generates millisecond delays specified by the argument. |
| Sound_Play | This function is similar to the mikroC Pro for PIC Sound_Play function. It generates a square wave signal with the specified frequency and duration. The values to be loaded into the timer registers are calculated as described above. After loading the timer registers, the timer counter is enabled and the program waits until the specified duration occurs. After this point, the timer counter is disabled to stop further interrupt from being generated, which effectively stops the buzzer. |
| Delay_Seconds | This function generates seconds delays specified by the argument. |

Delay_Seconds: This function generates seconds delays specified by the argument.

It is assumed that the buzzer is connected to the RC2 pin of the microcontroller. At the beginning of the program, PORTC is configured as the digital output. Then, TIMER0 is initialized to operate in the 16-bit mode, with internal clock, and with a prescaler value of 2. Global interrupts and TIMER0 interrupts are enabled by setting the appropriate bits in register INTCON. The rest of the program is as in the mikroC Pro for the PIC where the required melody is played by generating the required notes.

## Project 5.14—Generating Custom LCD Fonts

### Project Description

There are some applications where we may want to create custom fonts such as special characters, symbols, or logos on the LCD. This project will show how to create the symbol of an arrow pointing right on the LCD, and then display "Right arrow <symbol of side arrow>" on the first row of the LCD.

```
/*************************************************************************
                                PLAYING MELODY
                                ==============

In this project a buzzer is connected to port pin RC2 of a PIC18F45K22 microcontroller,
operating with an 8 MHz crystal.

The program plays the classical "Happy Birthday" melody.


Author:        Dogan Ibrahim
Date:          August 2013
File:          XC8-SOUND1.C
*************************************************************/

#include <xc.h>
#pragma config MCLRE = EXTMCLR, WDTEN = OFF, FOSC = HSHP
#define _XTAL_FREQ 8000000
#define BUZZER LATCbits.LATC2
#define Max_Notes 25

unsigned char TIMER_H, TIMER_L;

unsigned int Notes[Max_Notes] =
{
 262, 262, 294, 262, 349, 330, 262, 262, 294, 262, 392,
 349, 262, 262, 524, 440, 349, 330, 294, 466, 466, 440,
 349, 392, 349
};

unsigned char Durations[Max_Notes] =
{
 1, 1, 2, 2, 2, 3, 1, 1, 2, 2, 2, 3, 1, 1, 2, 2, 2, 2,
 1, 1, 2, 2, 2, 3
};

//
// Timer interrupt service routine
//
void interrupt isr(void)
{
   TMR0H = TIMER_H;
   TMR0L = TIMER_L;
   BUZZER = ~BUZZER;
   INTCONbits.TMR0IF= 0;

}

//
// This function generates millisecond delays
//
```

**Figure 5.76: MPLAB XC8 Program Listing.**

```
void Delay_Ms(unsigned int s)
{
   unsigned int j;
   for(j = 0; j < s; j++)__delay_ms(1);
}

//
// This function plays a note with the specified frequency and duration
//
void Sound_Play(unsigned int freq, unsigned int duration)
{
 float period;
 period = 500000.0/freq;
 period = 65536 – period;
 TIMER_H = (char)(period/256);
 TIMER_L = (char)(period – 256*TIMER_H);
 TMR0H = TIMER_H;
 TMR0L = TIMER_L;
 T0CONbits.TMR0ON = 1;
 Delay_Ms(duration);
 T0CONbits.TMR0ON = 0;
}

//
// This function creates seconds delay. The argument specifies the delay time in seconds.
//
void Delay_Seconds(unsigned char s)
{
   unsigned char i,j;

   for(j = 0; j < s; j++)
   {
      for(i = 0; i <  100; i++)__delay_ms(10);
   }
}




void main()
{
    unsigned char i;

    ANSELC = 0;                                    // Configure PORTC as digital
    TRISC = 0;                                     // Configure PORTC as output
    BUZZER = 0;                                    // Buzzer = 0 to start with
//
// Configure TIMER0 for 16 bit, prescaler = 2
//
    T0CONbits.TMR0ON = 0;                          // Timer OFF
    T0CONbits.T08BIT = 0;                          // Timer in 16 bit mode
```

**Figure 5.76**
cont'd

```
    T0CONbits.T0CS = 0;                          // Use internal clock
    T0CONbits.T0SE = 0;                          // Low-to-high transition
    T0CONbits.PSA = 0;                           // Use the prescaler
    T0CONbits.T0PS = 0;                          // Prescaler = 2

    INTCON = 0xA0;                               // Enable global and TMR0 interrupts

    for(;;)                                      // Endless loop
    {
     for(i = 0; i < Max_Notes; i++)              // Do for all notes
     {
       Sound_Play(Notes[i], 400 * Durations[i]); // Play the notes
       Delay_Ms(100);                            // Gap between notes
     }
     Delay_Seconds(3);                           // Repeat after 3 s
    }
}
```

**Figure 5.76**
cont'd

mikroC Pro for the PIC compiler provides a tool that makes the creation of custom fonts very easy. The steps for creating a font of any shape are given below:

- Start mikroC Pro for PIC compiler.
- Select Tools → LCD Custom Character. You will see the LCD font editor form shown in Figure 5.77.
- Select 5 × 7 (the default).
- Click "Clear all" to clear the font editor.
- Now, draw the shape of your font by clicking on the squares in the editor window. In this project, we will be creating the symbol of a "right arrow" as shown in Figure 5.78.
- When you are happy with the font, click "mikroC Pro for PIC" tab so that the code generated will be for the mikroC Pro for PIC compiler.
- Click "Generate Code" button. You will get the code as shown in Figure 5.79.
- Click "Copy Code To Clipboard" to save the code.
- We shall see later in the project how to display this font using the generated code.

### *Circuit Diagram*

The circuit diagram of the project is shown in Figure 5.80.

### *Project PDL*

The PDL of this project is very simple and is given in Figure 5.81.

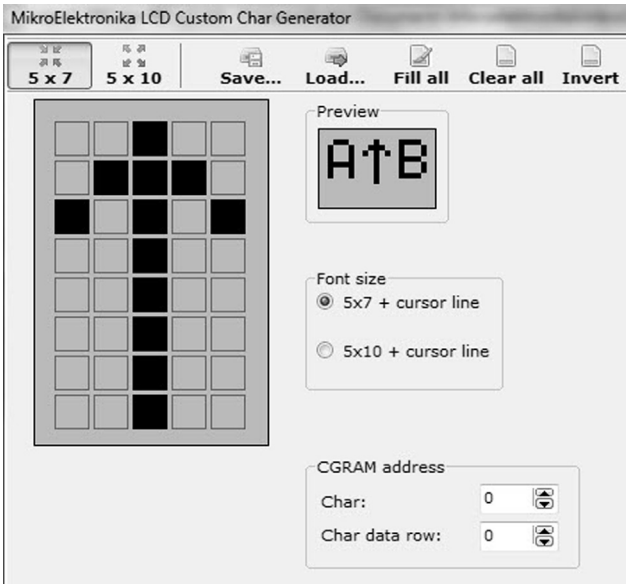


**Figure 5.77: LCD Font Editor.**

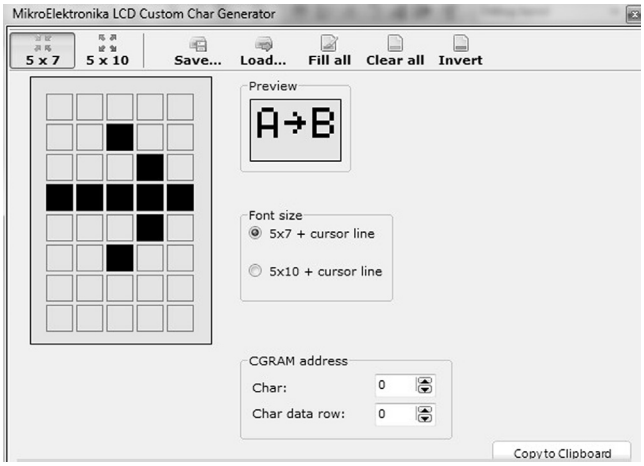


**Figure 5.78: Creating a "Side Arrow" Font.**

## *Project Program*

### *mikroC Pro for PIC*

The mikroC Pro for PIC program is named MIKROC-LCD2.C, and the program listing of the project is shown in Figure 5.82. At the beginning of the project, the connections between the microcontroller and the LCD are defined using *sbit* statements. PORTB is

```
mikroC PRO   mikroPascal PRO   mikroBasic PRO

const char character[] = {0,4,2,31,2,4,0,0};

void CustomChar(char pos_row, char pos_char) {
  char i;
    Lcd_Cmd(64);
    for (i = 0; i<=7; i++) Lcd_Chr_CP(character[i]);
    Lcd_Cmd(_LCD_RETURN_HOME);
    Lcd_Chr(pos_row, pos_char, 0);
}
```

**Figure 5.79: Generating Code for the Font.**

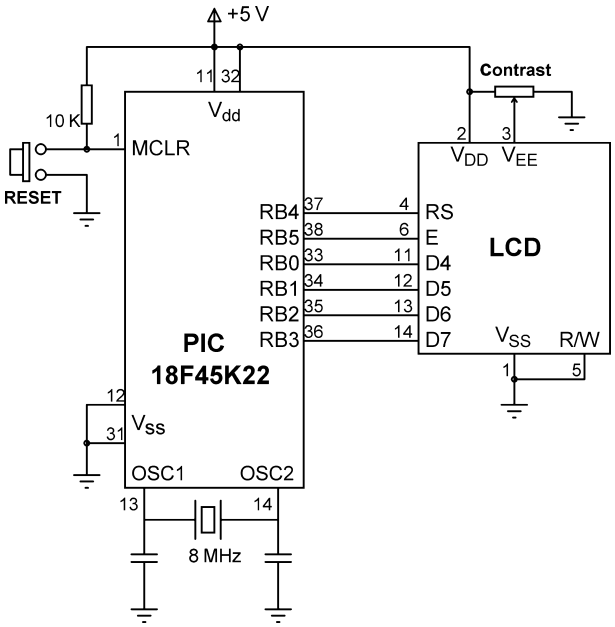


**Figure 5.80: Circuit Diagram of the Project.**

**BEGIN**
Define microcontroller - LCD connections
Define bit map of the required font
Configure PORTB as digital and output
Initialize LCD
Display text on LCD
**CALL** CustomChar to display the created font
**END**

**BEGIN/CustomChar**
Display required font as character 0
**END/CustomChar**

**Figure 5.81: PDL of the Project.**

```
/*************************************************************************
                              CREATING CUSTOM FONT ON LCD
                              ===========================

This project displays a custom font on the LCD. A "Right arrow" is displayed with text as shown
below:

                    Right arrow  <up arrow symbol>

The font has been created using the mikro C font editor.

In this project a HD44780 controller based LCD is connected to a PIC18F45K22 type microcontroller,
operated from an 8MHz crystal.

The LCD is connected to PORTB of the microcontroller as follows:

 LCD pin   Microcontroller pin
   D4          RB0
   D5          RB1
   D6          RB2
   D7          RB3
   R/S         RB4
   E           RB5

R/W pin of the LCD is not used and is connected to GND. The brightness of the LCD is
controlled by connecting the arm of a 10 K potentiometer to pin Vo of the LCD. Other
pins of the potentiometer are connected to power and ground.


Author:  Dogan Ibrahim
Date:    August 2013
File:    MIKROC-LCD2.C
*************************************************************************/
// Start of LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End of LCD module connections

//
// The following code is generated automatically by the mikroC compiler font editor
```

**Figure 5.82: Program Listing of the Project.**

```
//
 const char character[] = {0,4,2,31,2,4,0,0};

void CustomChar(char pos_row, char pos_char) {
  char i;
    Lcd_Cmd(64);
    for (i = 0; i <= 7; i++) Lcd_Chr_CP(character[i]);
    Lcd_Cmd(_LCD_RETURN_HOME);
    Lcd_Chr(pos_row, pos_char, 0);
}

//
// Start of Main program
//
void main()
{

    ANSELB = 0;                                  // Configure PORTB as digital
    TRISB = 0;                                   // Configure PORTB pins as output

    Lcd_Init();                                  // Initialize LCD
    Lcd_Cmd(_LCD_CLEAR);                         // Clear display
    Lcd_Cmd(_LCD_CURSOR_OFF);                    // Cursor off

    Lcd_Out(1, 1, "Right arrow");                // Display text "Right arrow"
    CustomChar(1, 13);                           // Display the "right arrow" symbol

    while(1);                                    // End of program, wait here forever
}
```

**Figure 5.82**
cont'd


**Figure 5.83: The LCD Display.**

configured as a digital output port. The LCD is initialized, cleared, and the cursor is turned OFF. Then, the LCD_Out function is called to display text "Right arrow", starting row 1 and column 1 of the LCD. Function CustomChar is generated by the compiler, and this function displays the created font at the specified row and column positions.

Figure 5.83 shows a picture of the LCD display.

## Project 5.15—Digital Thermometer

### Project Description

In this project, the design of a digital thermometer is described. An analog temperature sensor is used to sense the temperature, and the temperature is displayed on an LCD. The block diagram of the digital thermometer is shown in Figure 5.84.

An LM35DZ type analog temperature sensor is used in this project. This is a three-pin small sensor that provides an analog output voltage directly proportional to the measured temperature. The output voltage is given by

$$V_o = 10 \text{ mV/°C}$$

Thus, for example, at 20 °C, the output voltage is 200 mV, at 25 °C it is 250 mV, and so on.

### Circuit Diagram

The circuit diagram of the project is shown in Figure 5.85. The temperature sensor is connected to analog input AN0 (RA0) of the microcontroller. The LCD is connected to PORTB as in the previous LCD projects.

### Project PDL

The PDL of this project is very simple and is given in Figure 5.86.

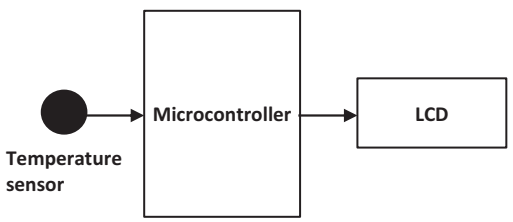


**Figure 5.84: Block Diagram of the Digital Thermometer.**

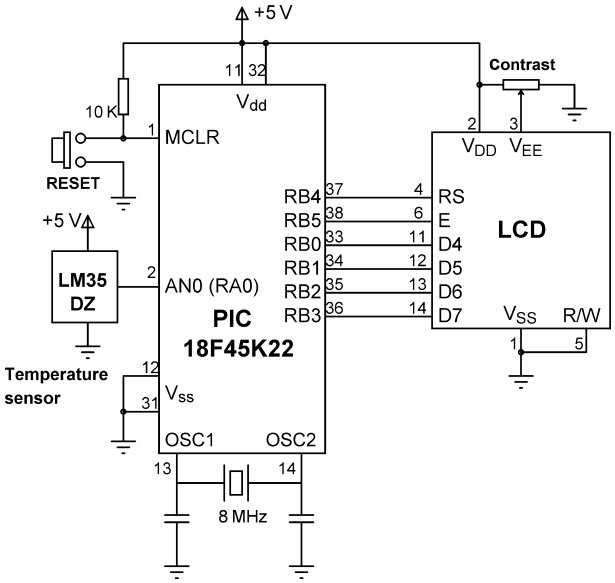


**Figure 5.85: Circuit Diagram of the Project.**

## *Project Program*

### *mikroC Pro for PIC*

The mikroC Pro for the PIC program is named MIKROC-THERMO.C, and the program listing of the project is shown in Figure 5.87. At the beginning of the project, the connections between the microcontroller and the LCD are defined using *sbit* statements.

```
BEGIN
        Define LCD connections
        Configure PORTA and PORTB as digital
        Configure RA0 (AN0) as input
        Initialize LCD
        DO FOREVER
                Read temperature from channel 0
                Convert reading into millivolts
                Divide by 10 to find the temperature in Degrees C
                Convert temperature into string
                Clear display
                Display Heading "Temperature"
                Display the temperature
                Wait 1 s
        ENDDO
END
```

**Figure 5.86: PDL of the Project.**

```
/*************************************************************************
                                Digital Thermometer
                                ===================

This project is a digital thermometer with LCD display. An LM35DZ type analog temperature
sensor is used to sense the temperature. The sensor is connected to analog input AN0 (RA0)
of a PIC18F45K22 microcontroller operating with an 8 MHz crystal. The program reads the
temperature every second and displays on the LCD.

Programmer:    Dogan Ibrahim
Date:          September 2013
File:          MIKROC-THERMO.C

*************************************************************************/
// LCD module connections
sbit LCD_RS at RB4_bit;
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;

sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;
// End LCD module connections

void main()
{
   unsigned char Txt[14];
   unsigned int temp;
   float mV;

   ANSELA = 0;                          // Configure PORTA as digital
   ANSELB = 0;                          // Configure PORTB as digital
   TRISA.RA0 = 1;                       // RA0 is input
   Lcd_Init();                          // Initialize LCD

   Lcd_Cmd(_LCD_CURSOR_OFF);             //  Disable cursor

   for(;;)
   {
    temp = ADc_Read(0);                 // Read from channel 0
    mV = temp * 5000.0/1024.0;          // Convert to mV
    mV = mV/10.0;                       // mV is now the Degrees C
    floatToStr(mV, Txt);                // Convert to string
    Lcd_cmd(_LCD_CLEAR);                // Clear LCD
    Lcd_out(1,1,"Temperature");         // Display heading
    Lcd_Out(2,1,Txt);
    Delay_ms(1000);                    // Display temperature
   }                                    // Wait 1 s and repeat
}
```

**Figure 5.87: mikroC Pro for PIC Program Listing.**

PORTA is configured as a digital input. The LCD is initialized, cleared, and the cursor is turned OFF. The remainder of the program is executed in an endless loop. Here, the temperature is read from analog channel 0 (AN0 or RA0), converted into millivolts by multiplying with 5000, and dividing by the A/D converter resolution (10 bits), and divided by 10 to find the temperature. The temperature is then converted into a string and is displayed on the LCD. In this program, floating point arithmetic is used to find and display the temperature for higher accuracy.