PC-Based Microcontroller Programmer and Training Kit

Manual and Workbook

E-mail: lemorld@gmail.com

jon_x2_legaspi@yahoo.com

Rbpaderna_iii@yahoo.com

Aapdones_027@yahoo.com

This document is bundled with the project, PC-Based Microcontroller Programmer and Training Kit.

This can also be viewed inside the project software PICSoft and will open in a PDF viewer.

Notes, Trademarks, and External References

The project, *PC-Based Microcontroller Programmer and Training Kit* discussed in this document employs the use of Microchip's Hi-Tech PICC Lite compiler, which is available for free. You can download the latest version of the Hi-Tech PICC Lite from its website (http://www.htsoft.com).

The project also uses the freeware WinPic800. More information and some downloads can be accessed from their site http://www.winpic800.com. You can also send an e-mail to winpic800@hotmail.com .

Some of the concepts, codes, and procedures in the project are developed with the help of some other books, publications, articles and web pages. The developers are in deep gratitude to these sources and highly recommend reading these in the course of PIC MCU interfacing. The list of these references can be found in *Section 9- References / Suggested Reading* of this document.

CONTENTS OF THE KIT:

Circuits:

-USB PIC Programmer

-ZIF + Headers block

-LED module

-7-segment module

-Dot matrix module

-LCD module

-Motor module

-Temperature Sensor

-USB A-B Cable

PIC MCUs

-12F683	8-pin PIC MCU
-16F676	14-pin PIC MCU
-16F88	18-pin PIC MCU
-16F648A	18-pin PIC MCU
-16F819	18-pin PIC MCU
-16F737	28-pin PIC MCU
-16F747	40-pin PIC MCU

EEPROM

-24C16	EEPROM

MINIMUM SYSTEM REQUIREMENTS

-Windows 95/98/ME/NT/2000/XP

-Pentium 233-megahertz (MHz) processor or faster (300 MHz or faster is recommended)

-At least 64 megabytes (MB) of RAM (128 MB is recommended)

-At least 300 megabytes (MB) of available space on the hard disk

-CD-ROM or DVD-ROM drive

-Keyboard and a Microsoft Mouse or some other compatible pointing device

-Video adapter and monitor with Super VGA (800 x 600) or higher resolution

INTRODUCTION

Microcontroller units (MCU) are one of the most significant integrated circuits in the technological evolution. They are found in nearly all electronic and digital devices. It is technically a single-chip computer. *Micro* suggests that the device is small, and *controller* suggests that it is used in control applications. Another term for microcontroller is embedded controller, since most of the microcontrollers are built into (or embedded in) the devices they control. A microcontroller has all the support chips incorporated inside the same chip, such as program memory and data memory, I/O interfaces, and external clock circuit.

Among the competing microcontroller products, the PIC Microcontroller by Microchip Inc. has been very popular because of its wide availability, extensive references and user base, low cost, large variety, and its immense capability.

The PC-Based Microcontroller Programmer and Training Kit is a device that utilizes the PIC Microcontrollers to guide students and hobbyists on interfacing with microcontrollers. The software and hardware package can also be used in developing applications for the PIC MCU and the training kit modules included.

This workbook consists of the procedures and steps in preparing, installing, and using the PC-Based Microcontroller Programmer and Training Kit. It also includes exercises that will guide the users in their experimentation with the device and with PIC Microcontrollers.

The students can freely perform the experiments in this workbook with the guidance of an instructor. The software included in the package, 'PICSoft', includes visual guides and lots of other additional information that will simplify learning for the students.

The experiments include a discussion that explains the modules briefly and gives preliminary knowledge about the PIC MCU and the devices to be interfaced with it. The students can then solidify their skills in the exercises that follow the experiments. The students are also encouraged to note their observations and conclusion at the end of the activity.

TABLE OF CONTENTS

1 INSTALLATION GUIDE

1.1 Preparing the Device

- Open the device case. The kit is composed of several circuits shown.

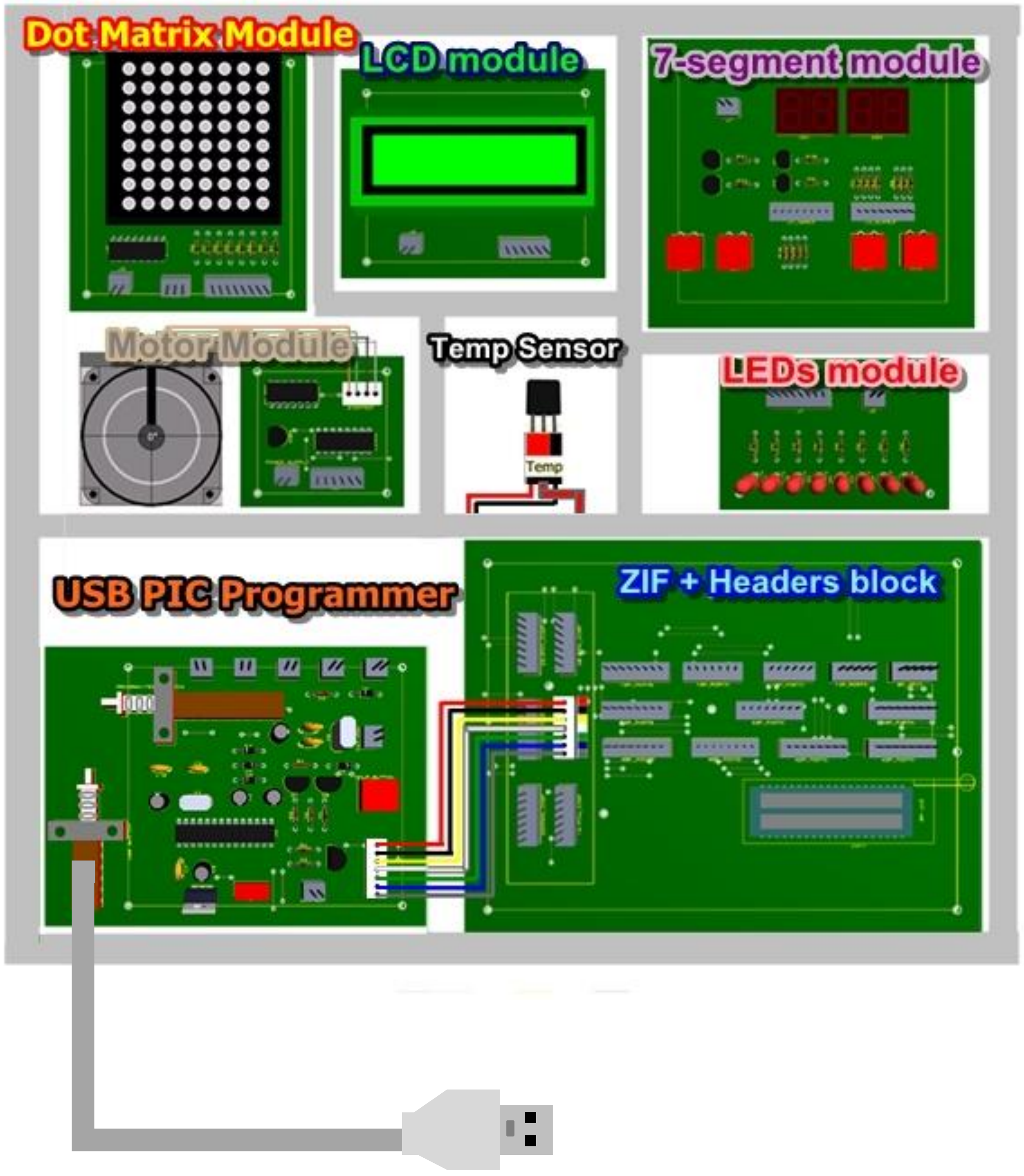


Figure 1 **PC-Based Microcontroller Programmer and Training Kit**

1.2 Device Driver Installation

- The first time you are using the device, or if the device is connected to another host PC, the first thing to do is to install the device driver.
- Connect the USB Cable from the USB PIC Programmer to a USB Port at the host PC.
- If the device is not yet installed, the LED indicator will light RED as sown in the figure below. Otherwise, if the indicator lights GREEN, it means that it is already installed and you can skip this section and go to Section 2.3 "Software Installation"

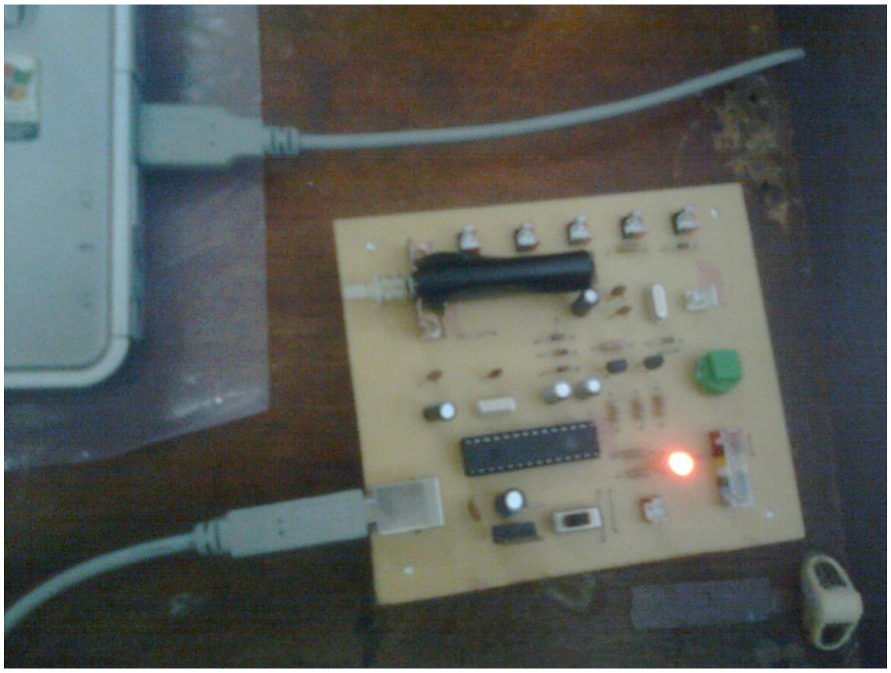
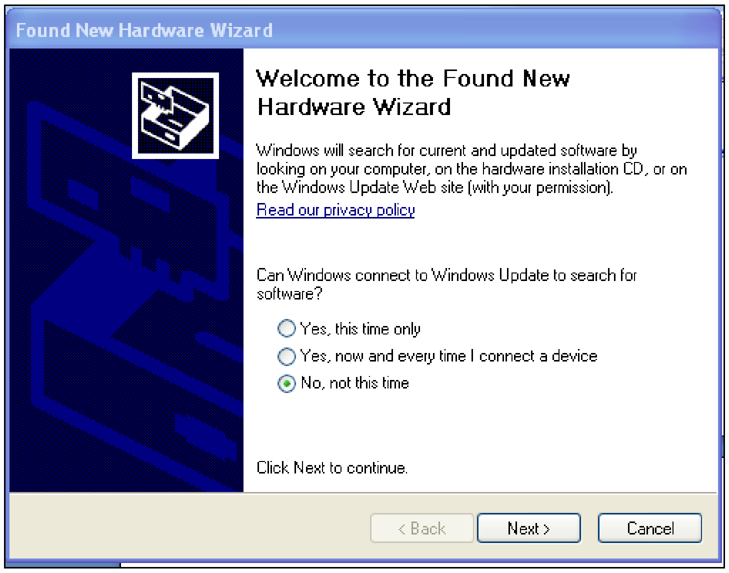


Figure 2 **Red LED indicating device driver is not yet installed**

- It is most likely that you will also see this message indicating that the host PC recognized the hardware but needs the drivers to install it. Choose the 3rd option – “No, not this time” , then click Next



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

Read our privacy policy

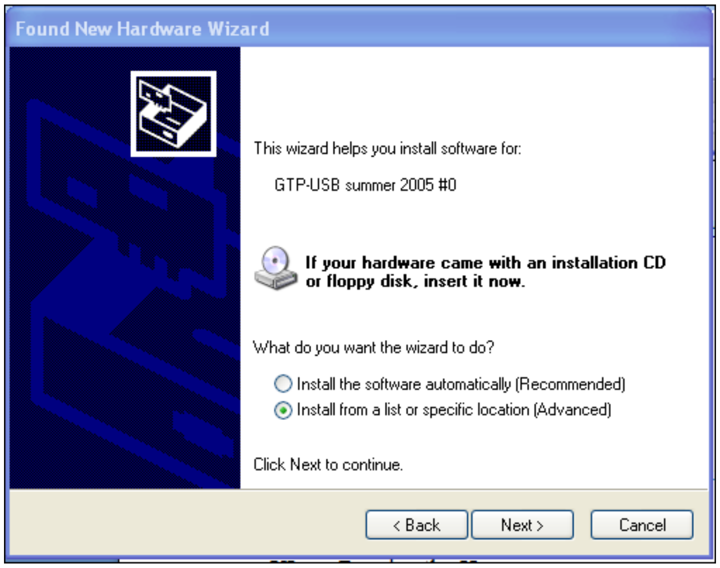
Can Windows connect to Windows Update to search for software?

Yes, this time only
Yes, now and every time I connect a device
[✓ No, not this time]

Click Next to continue.

Figure 3 **Found new hardware window**

- In the next window. Choose the 2nd option – “install from a list or specific location (advanced)’



Found New Hardware Wizard

This wizard helps you install software for:

GTP-USB summer 2005 #0

If your hardware came with an installation CD or floppy disk, insert it now.

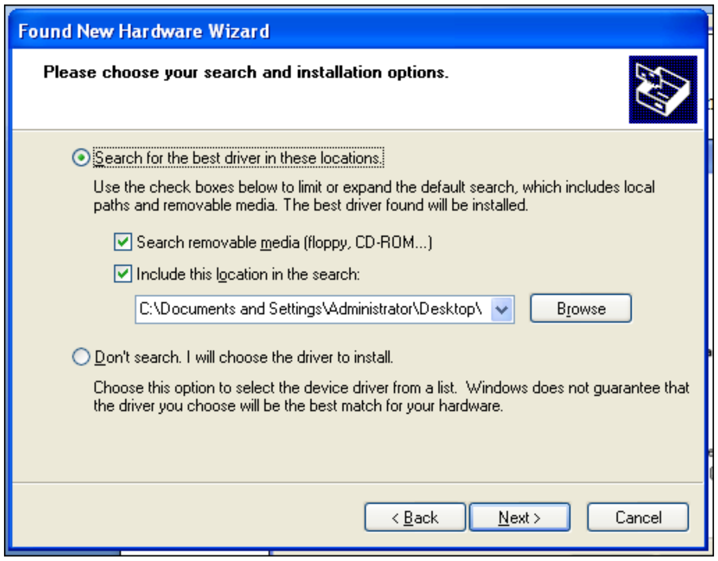
What do you want the wizard to do?

Install the software automatically (Recommended)
[✓ Install from a list or specific location (Advanced)]

Click Next to continue.

Figure 4 **Driver installation option window**

- Choose the first and second option – "search for the best driver…" and "Include this location" option, then click Browse to search for the drivers.


Found New Hardware Wizard

Please choose your search and installation options.

(●) Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

[✓] Search removable media (floppy, CD-ROM...)

[✓] Include this location in the search:

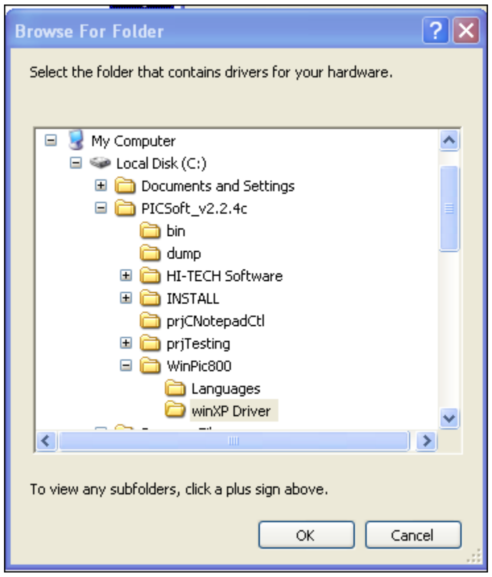
C:\Documents and Settings\Administrator\Desktop\

( ) Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

Figure 5 **Search for drivers window**

- Locate the folder PICSoft Setup/WinPic800/winXP_driver. Click OK, then click Next


Browse For Folder

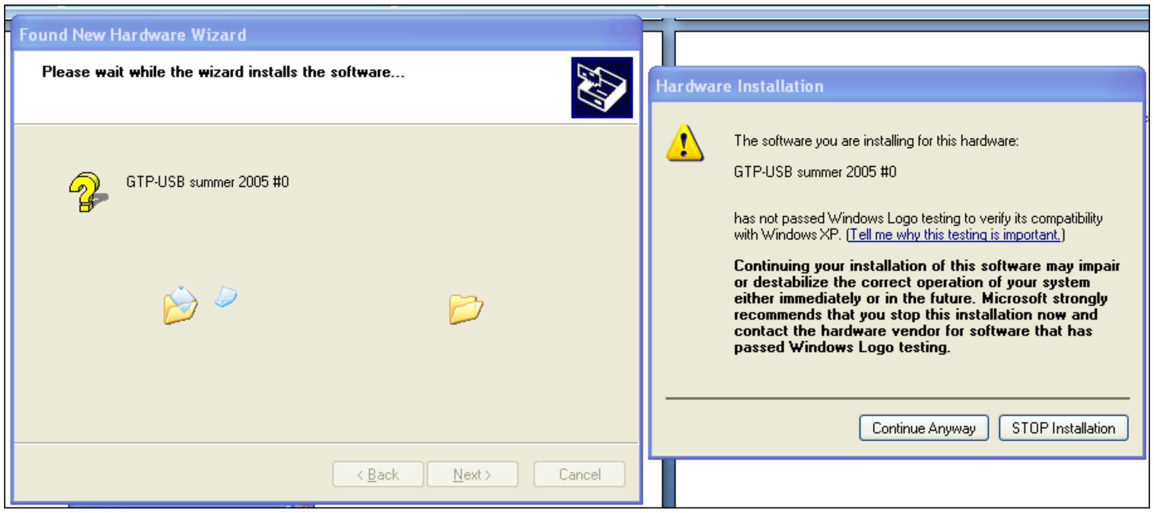
Select the folder that contains drivers for your hardware.

```
My Computer
  Local Disk (C:)
    Documents and Settings
    PICSoft_v2.2.4c
      bin
      dump
      HI-TECH Software
      INSTALL
      prjCNotepadCtl
      prjTesting
      WinPic800
        Languages
        winXP Driver
```

To view any subfolders, click a plus sign above.

Figure 4 **Locate driver directory window**

- This windows would appear, click “Continue Anyway”



Please wait while the wizard installs the software...

GTP-USB summer 2005 #0

The software you are installing for this hardware:

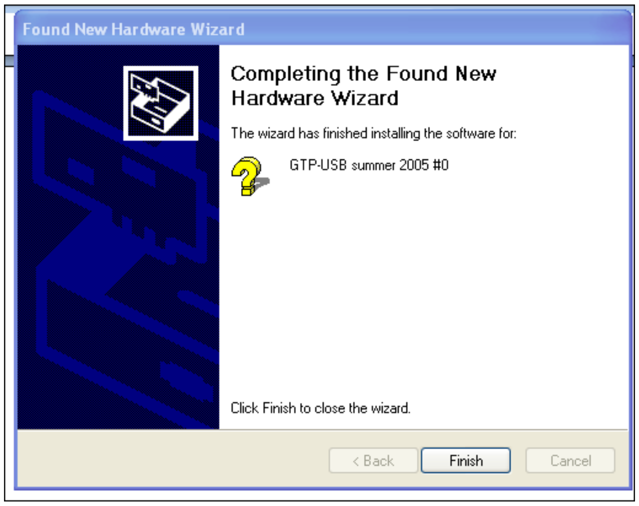
GTP-USB summer 2005 #0

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.

Figure 5 **installation window**

- It will then copy some files to the windows/system32 folder. Afterwards, this message of successful installation would appear. Click Finish.



Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

GTP-USB summer 2005 #0

Click Finish to close the wizard.

Figure 6 **installation complete window**

- After this, you should see the LED in the device turning to GREEN. It means the device is already installed.

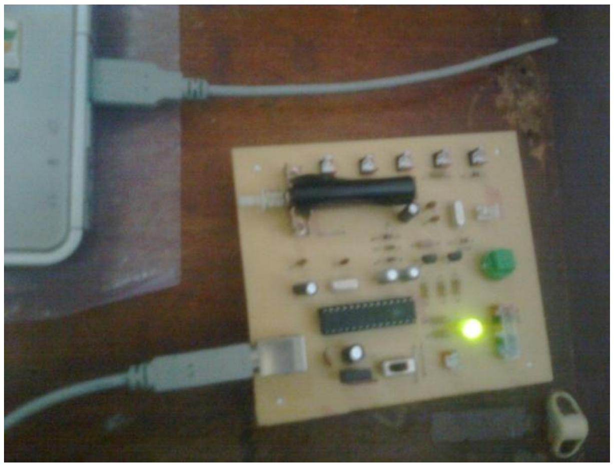


Figure 5 **Green LED indicating device is installed**

- You can also confirm the driver installation in the device manager of Windows XP. Go to My Computer, right click, then Properties. Under Hardware tab, click Device Manager. You should see this entry inside the device manager window named "GTP-USB summer 2005 #0"

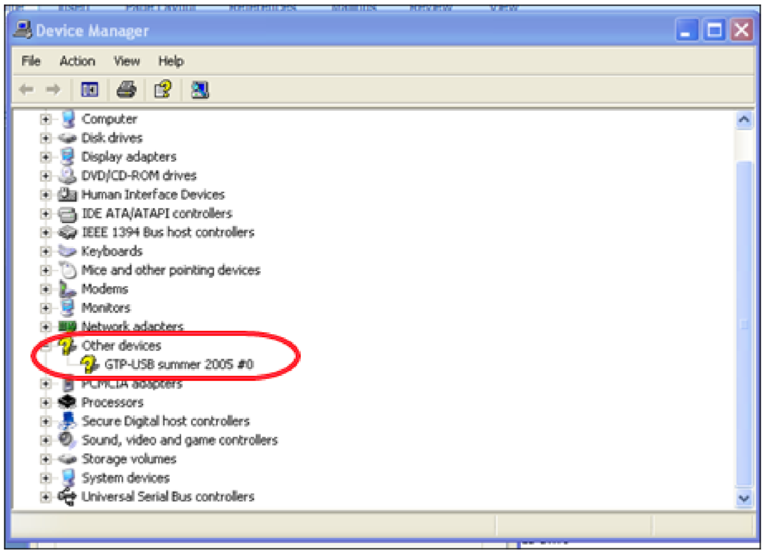


Figure 6 **Device manager window**

1.3 Software Installation

- This section applies to installation of PICSoft if it is not yet installed. Otherwise, you can go to Section 3- Familiarization or Section 4 – Lab. Experiments.
- Locate and run the “PICSoft Setup” installer inside the PICSoft folder provided.

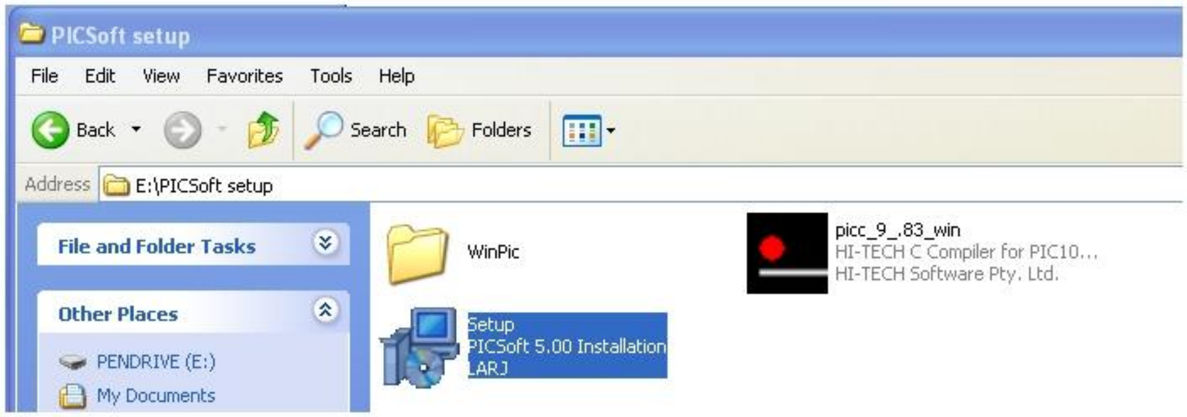
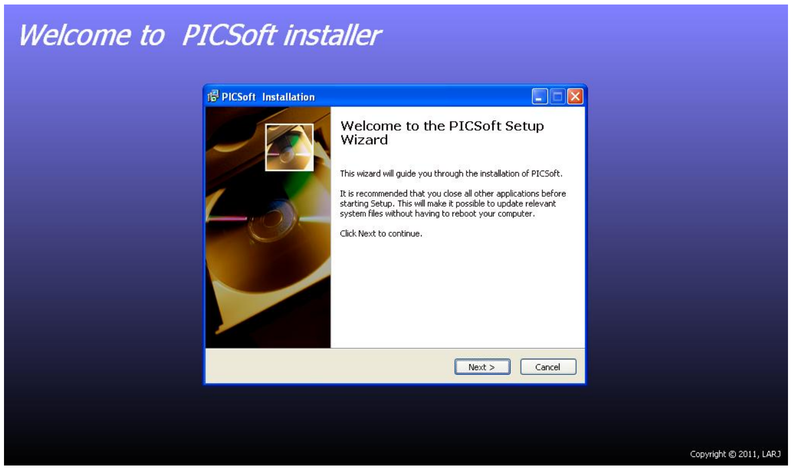


Figure 7 **PICSoft setup executable**

- At the welcome prompt, click Next.



Welcome to the PICSoft Setup Wizard

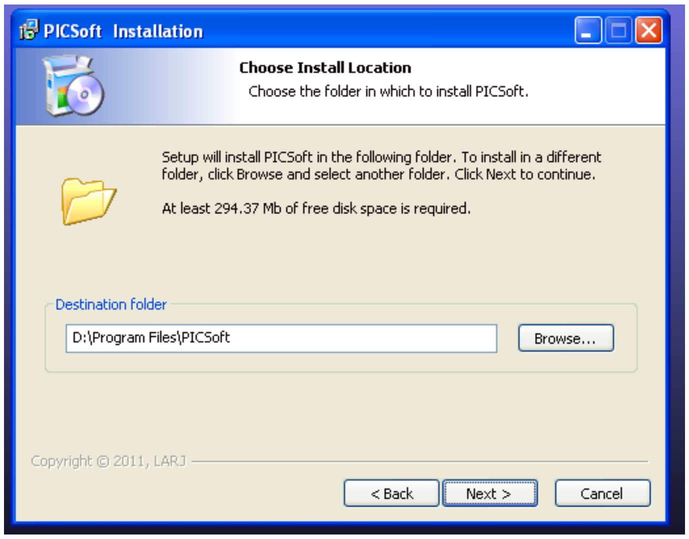
This wizard will guide you through the installation of PICSoft.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure 8 **PICSoft installer Welcome Prompt**

- At the next window, you can change the directory where PICSoft would be installed. It is recommended that you leave the default install folder, which is at the Program Files folder. Click Next.


PICSoft Installation

Choose Install Location

Choose the folder in which to install PICSoft.

Setup will install PICSoft in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

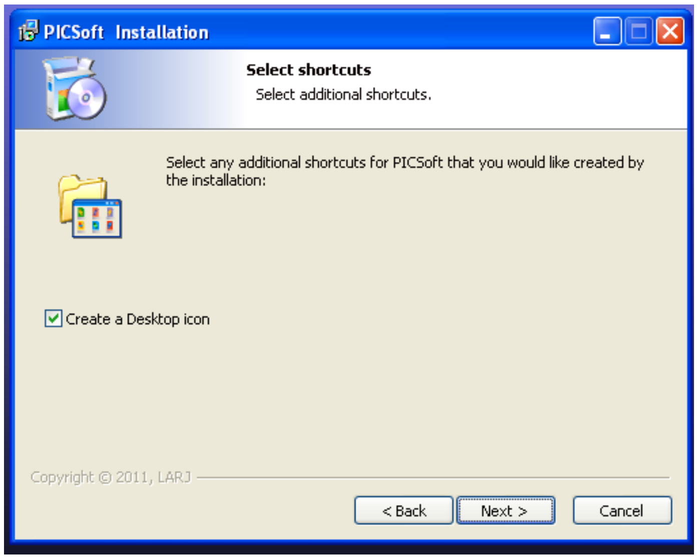
At least 294.37 Mb of free disk space is required.

Destination folder: D:\Program Files\PICSoft

Copyright © 2011, LARJ

Figure 9 **PICSoft install location**

- At the next window, you can choose to create shortcuts or not. Afterwards, click Next.


PICSoft Installation

Select shortcuts

Select additional shortcuts.

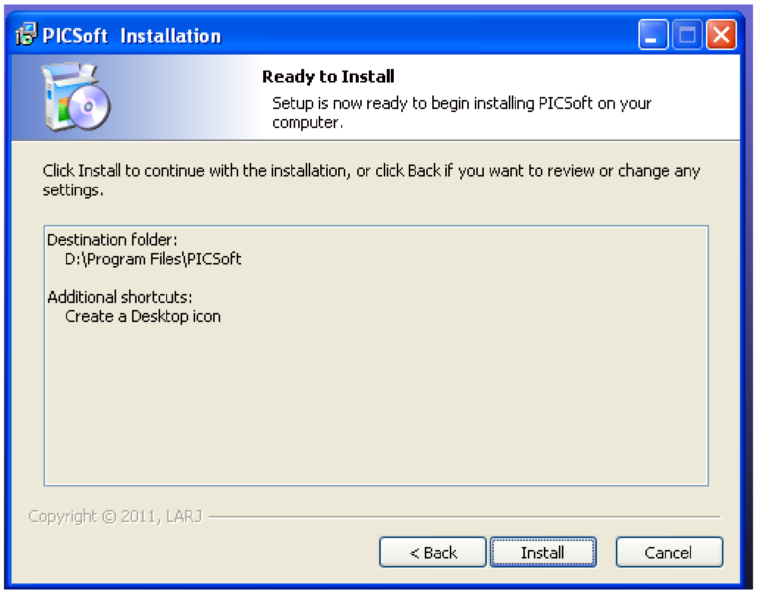
Select any additional shortcuts for PICSoft that you would like created by the installation:

[✓] Create a Desktop icon

Copyright © 2011, LARJ

Figure 10 **Create Shortcuts option**

- Installation is now ready to begin. Click Install to begin installation.



Ready to Install

Setup is now ready to begin installing PICSoft on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

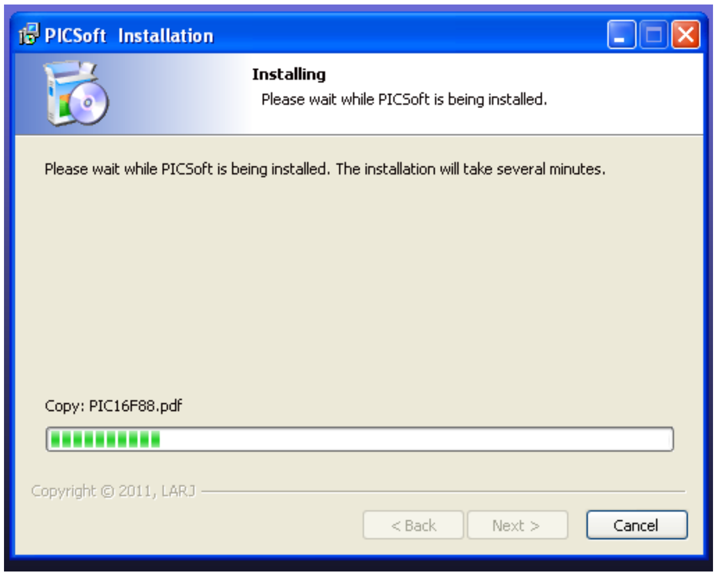
Destination folder:
D:\Program Files\PICSoft

Additional shortcuts:
Create a Desktop icon

Copyright © 2011, LARJ

Figure 11 **Installation start**

- At this point, wait until installation is finished.



Installing

Please wait while PICSoft is being installed.

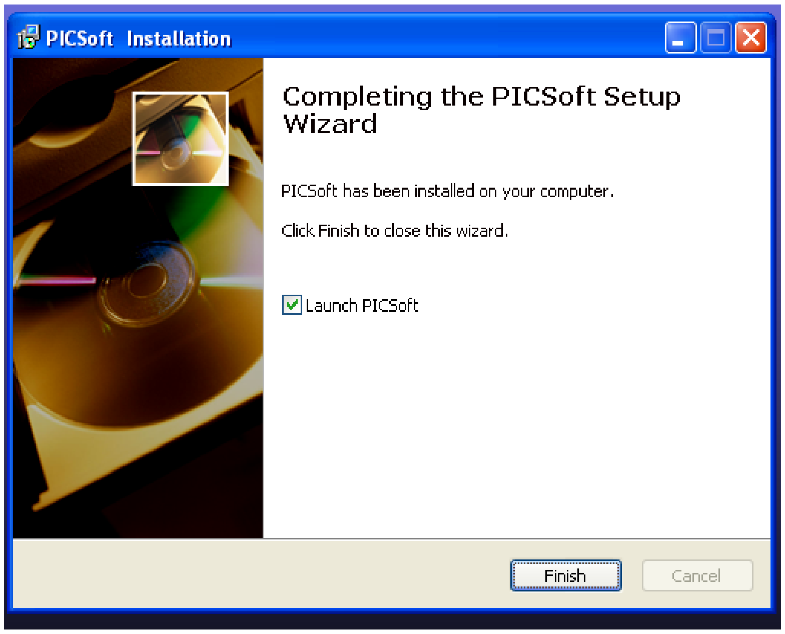
Please wait while PICSoft is being installed. The installation will take several minutes.

Copy: PIC16F88.pdf

Copyright © 2011, LARJ

Figure 12 **Installation ongoing**

- When the installation is finished, you can also choose to launch the program. Click Finish.



PICSoft Installation

Completing the PICSoft Setup Wizard

PICSoft has been installed on your computer.

Click Finish to close this wizard.

[✓] Launch PICSoft

Figure 13 **Installation complete**

Compiler Installation

- Locate and run the "picc_9_83_win" installer inside the PICSoft folder provided.

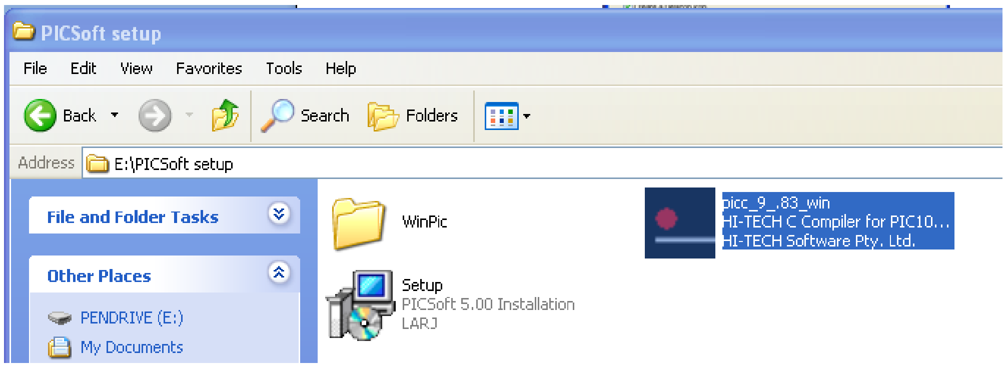
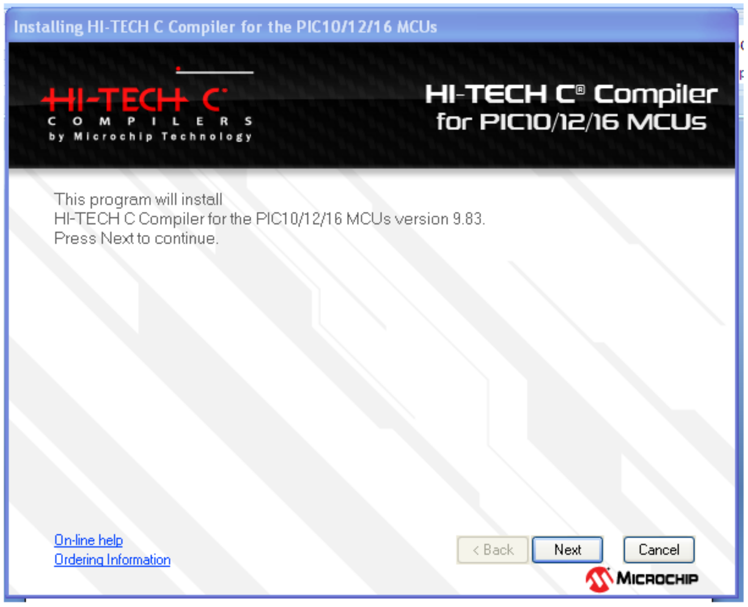


Figure 14 **PICC setup executable**

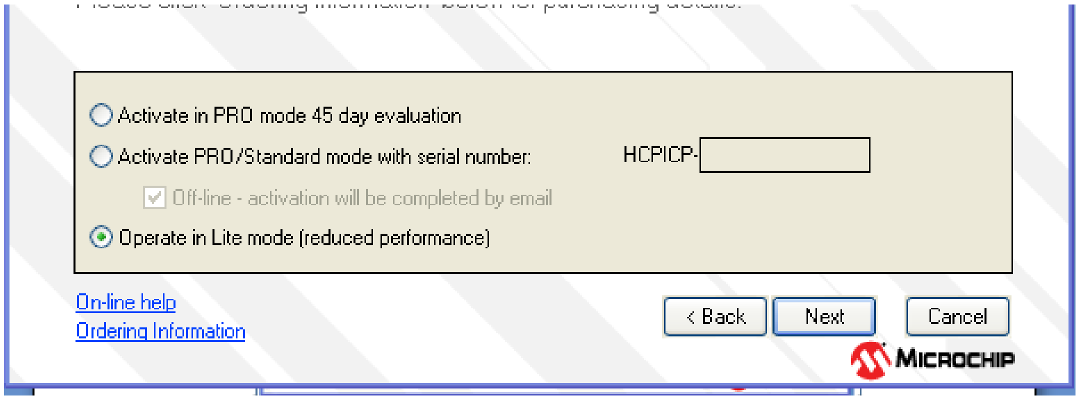
- At the welcome prompt, click ‘Next’


Installing HI-TECH C Compiler for the PIC10/12/16 MCUs
HI-TECH C COMPILERS by Microchip Technology
HI-TECH C® Compiler for PIC10/12/16 MCUs
On-line help
Ordering Information
< Back
Next
Cancel
MICROCHIP

This program will install
HI-TECH C Compiler for the PIC10/12/16 MCUs version 9.83.
Press Next to continue.

Figure 15 **PICC installer Welcome Prompt**

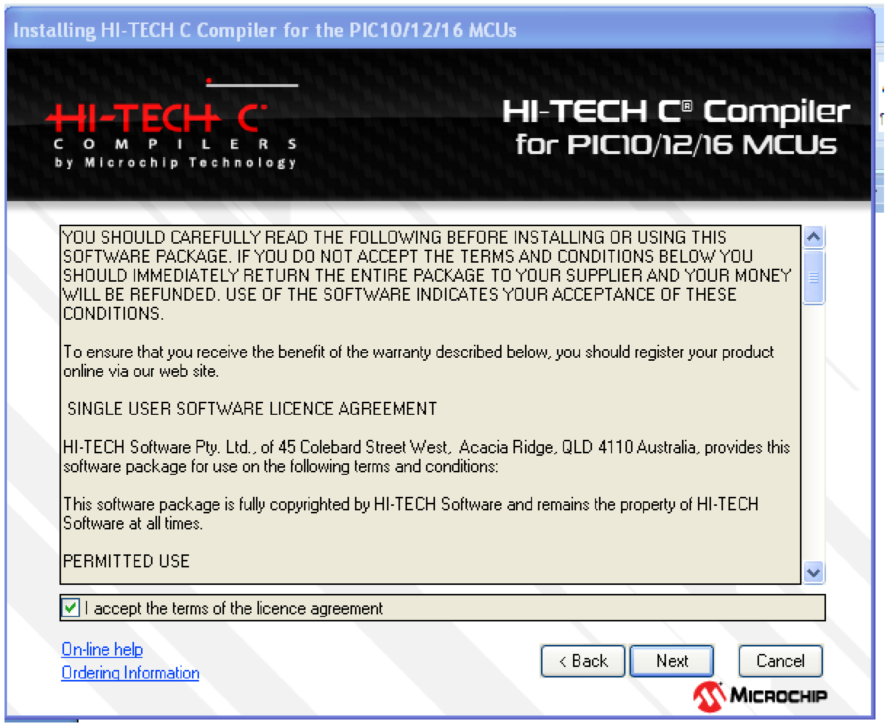
- At the options window, choose ‘Operate in Lite mode’. Operating in Lite mode is free of charge but does not include some advanced features. Nevertheless, its performance is more than enough for most experiments, including the ones in this workbook. Click ‘Next’.



○ Activate in PRO mode 45 day evaluation
○ Activate PRO/Standard mode with serial number: HCPICP-
☑ Off-line - activation will be completed by email
[✓ Operate in Lite mode (reduced performance)]

Figure 16 **Compiler mode option window**

- At the License Agreement window, check the box 'I agree…" after reading the terms and condition. Click Next.



YOU SHOULD CAREFULLY READ THE FOLLOWING BEFORE INSTALLING OR USING THIS SOFTWARE PACKAGE. IF YOU DO NOT ACCEPT THE TERMS AND CONDITIONS BELOW YOU SHOULD IMMEDIATELY RETURN THE ENTIRE PACKAGE TO YOUR SUPPLIER AND YOUR MONEY WILL BE REFUNDED. USE OF THE SOFTWARE INDICATES YOUR ACCEPTANCE OF THESE CONDITIONS.

To ensure that you receive the benefit of the warranty described below, you should register your product online via our web site.

SINGLE USER SOFTWARE LICENCE AGREEMENT

HI-TECH Software Pty. Ltd., of 45 Colebard Street West, Acacia Ridge, QLD 4110 Australia, provides this software package for use on the following terms and conditions:

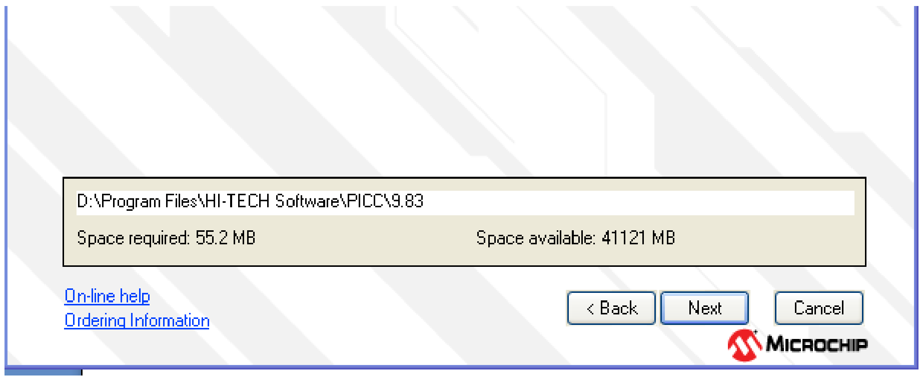
This software package is fully copyrighted by HI-TECH Software and remains the property of HI-TECH Software at all times.

PERMITTED USE

✓ I accept the terms of the licence agreement

Figure 17 **License agreement window**

- Do not change the path in the installation directory box. Click Next.



D:\Program Files\HI-TECH Software\PICC\9.83

Space required: 55.2 MB Space available: 41121 MB

Figure 18 **Installation directory window**

- Leave the default language for the program as ‘English”. Click Next.

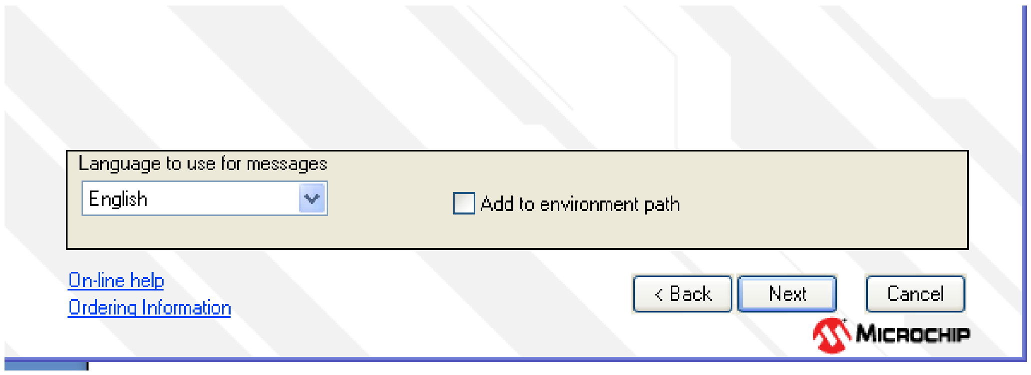
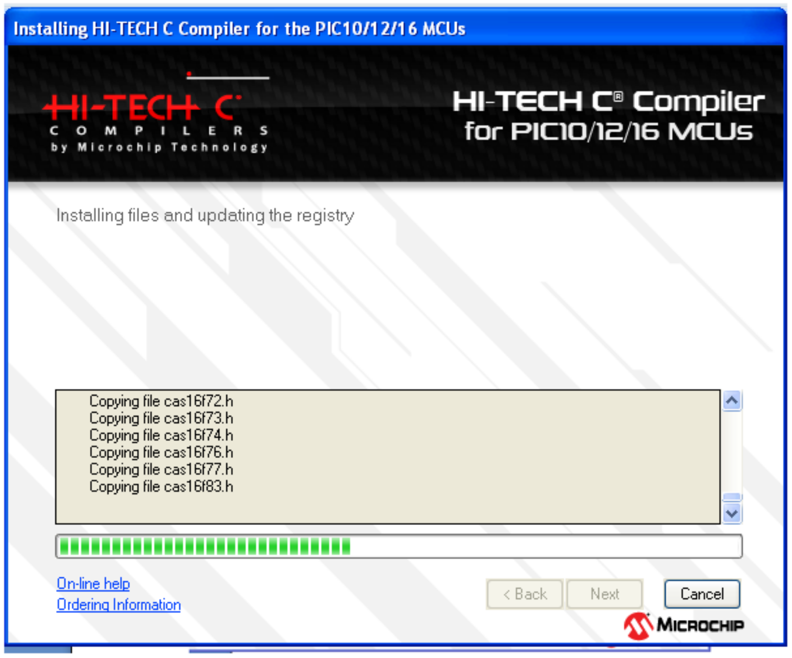

Language to use for messages
English
Add to environment path

Figure 19 **Choose language window**

- Wait for the installation to complete.


Installing HI-TECH C Compiler for the PIC10/12/16 MCUs

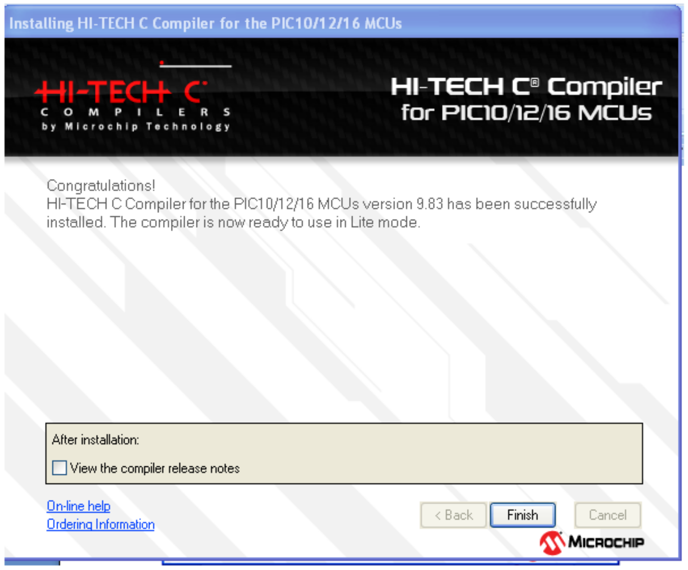
HI-TECH C® Compiler for PIC10/12/16 MCUs

Installing files and updating the registry

```
Copying file cas16f72.h
Copying file cas16f73.h
Copying file cas16f74.h
Copying file cas16f76.h
Copying file cas16f77.h
Copying file cas16f83.h
```

Figure 20 **Installation progress window**

- Click ‘Finish’ at the successful install prompt.


Installing HI-TECH C Compiler for the PIC10/12/16 MCUs

Congratulations!
HI-TECH C Compiler for the PIC10/12/16 MCUs version 9.83 has been successfully installed. The compiler is now ready to use in Lite mode.

After installation:
☐ View the compiler release notes

HI-TECH C COMPILERS by Microchip Technology
HI-TECH C® Compiler for PIC10/12/16 MCUs
On-line help
Ordering Information
< Back
Finish
Cancel
MICROCHIP

Figure 21 **Installation complete window**

2 FAMILIARIZATION

This section aims to introduce the hardware and software before the students can perform the laboratory experiments. It includes a comprehensive guide on its parts, functions, and operation.

2.1 Familiarization with Hardware

The PC-Based MCU Programmer and Training Kit is divided into several parts, the USB PIC Programmer, the ZIF + Headers block, and the training kits. The training kit is further divided into the LED module, 7-segment module, LCD module, Dot Matrix module, Motor module, and Temperature Sensor module.

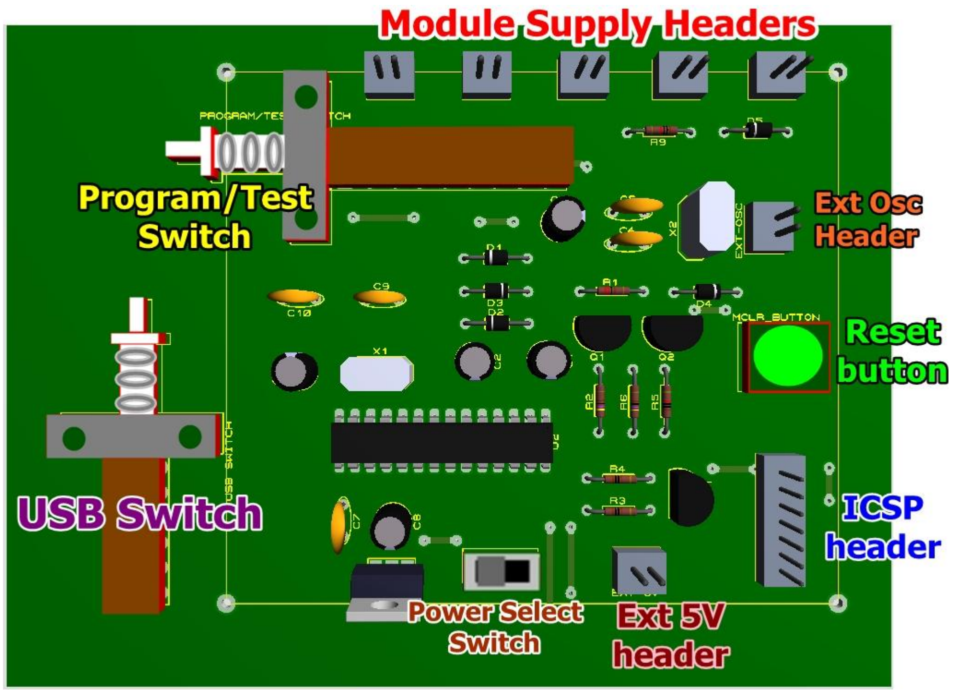


Figure 1 **USB PIC Programmer**

USB PIC Programmer

The USB PIC Programmer handles programming and testing of the program and serves as the main component of the device. Its main components are discussed shortly.

- **Switches**

  USB switch - a 4P2T (Four Pole – Double Throw) push-switch that toggles the device ON/OFF while directly connected to the host PC USB port. The four poles handle the VDD, GND, DATA, and CLOCK lines of the USB interface.

  Program/ Test switch - a 6P2T (Six Pole – Double Throw) push-switch that determines the operating mode of the device. When in its initial position (not pushed or the spring is released), the device is in Program Mode. In this mode, the PIC MCU pins communicate with the programmer and are able to accept programming signals, such as clock and data. When pushed (the spring is pressed), the device is in "Test Mode‟, wherein the PIC MCU pins is configured as I/O to interface with the training kit modules.

  Power Select Switch – a SPDT (Single Pole – Double Throw) slide switch that lets the user choose from the internal 5V supply (from the USB port) and from an external 5V supply connected to the Ext 5V Header (if available)\

  Reset button – a SPST (Single Pole – Single Throw) Momentary NO (Normally Open) pushbutton that resets the executing program in the PIC MCU when triggered. This button can be enabled and disabled through the program.

- **Headers**

  ICSP header – ICSP (In Circuit Serial Programming) is a technique used to program a microcontroller with simpler circuitry. The ICSP header, composed of 7 pins, is essential to connect the USB PIC Programmer to the PIC MCU housed in the ZIF + Headers Block. A 7-pin header connector is used.

  Module Supply header – this 2-pin header, composed of +5V and GND, is used to connect power rails to the training kit module used.

  Ext 5V header - this 2-pin header is used to connect the USB PIC Programmer to an external 5V power supply. Connecting an external 5V power supply would be useful when the current requirement of the modules exceeds the current capability sourced by the host PC, indicated by "USB Power Surge" notifications.

  Ext Oscillator header – this 2-pin header is used to provide clocking for certain type of PIC MCUs that doesn't include an internal oscillator. Oscillators are important in MCU operations, especially in timing-dependent applications.

- **USB Port** – a type B female USB port would be connected to the host PC using a USB cable

- **USB PIC Programmer chip** – the 18F2550 PIC MCU serves as the brain of the USB PIC Programmer. It holds the firmware that performs processing of the USB signals and programming of the target PIC MCU. The user would not be dealing with this chip, and is expected not to touch it, nor remove it from its socket.

ZIF+Headers Block

The ZIF + Headers Block provides the slot for the PIC MCU and all the connections necessary for interfacing the latter with the Training Kit modules. The circuitry employed allows the PIC MCU not to be removed from the socket to be programmed and tested.

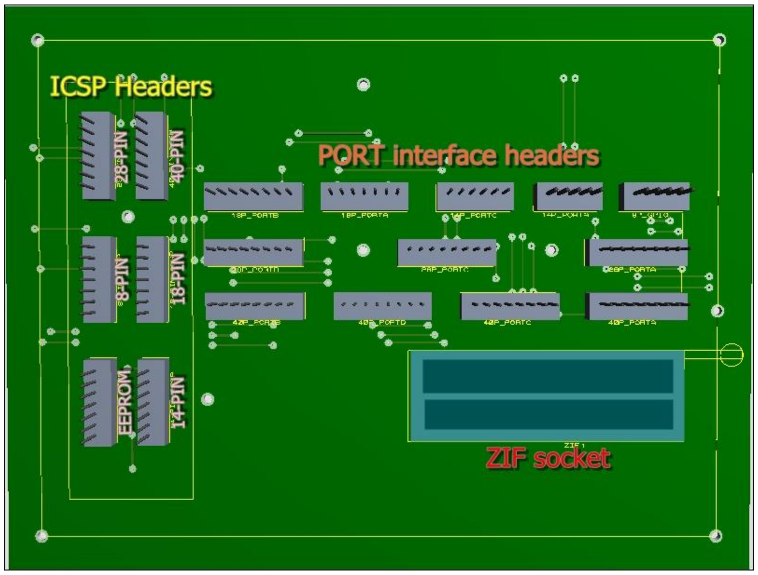


Figure 2 **ZIF+Headers Block**

- **ZIF Socket** – the 40-pin ZIF (Zero Insertion Force) Socket can accommodate 8/14/18/28/40 – pin PIC MCUs and would allow easy removal of the chips via the lever.
- **ICSP headers** – these are the connection points of the ICSP header in the USB PIC Programmer. The five varieties refer to the PIC MCU types based on pin number. One of the headers is reserved for the 24Cxx EEPROM chips.
- **Port Interface headers** – these headers provides the actual connection between the PIC MCU in the ZIF Socket and the electronic devices in the training kit modules. These headers are categorized according to pin number and the port name of the PIC MCUs.

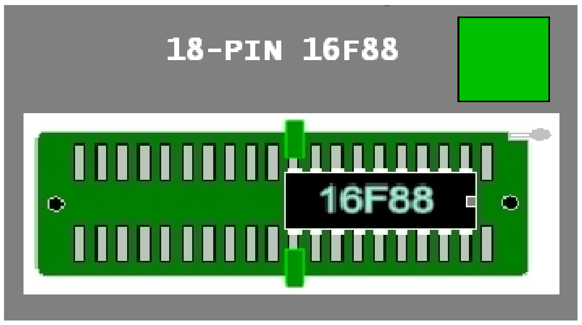
Training Kit Modules

The modules in the training kit are necessary for the actual testing of the program. The modules are LEDs, 7-segment, Dot Matrix, LCD, Motor, and Temperature Sensor. All of the modules contain I/O headers which would be connected to the Port Interface Headers in the ZIF + Headers Block. All of the modules also have Module Supply headers which should be connected to the supply headers inside the USB PIC Programmer circuit. These headers are color coded to guide in proper connection.

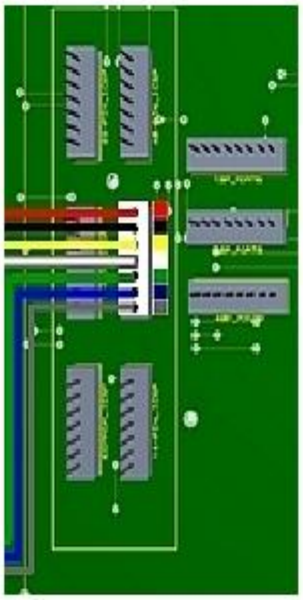
Each of the training kit modules will be discussed in detail in the experiments in Chapter 3.

Before opening the program…

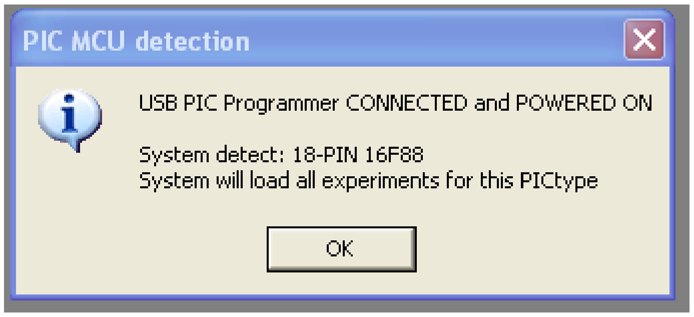
- The user can insert a PIC MCU in the ZIF Socket. For example, the 18-pin PIC 16F88 can be inserted to its socket, color-coded green. The ZIF socket has color guides that will simplify insertion.



- The ICSP cable must be connected to its corresponding ICSP header



- When the program is opened, the system will detect the PIC inserted

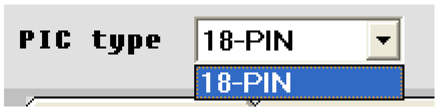


PIC MCU detection

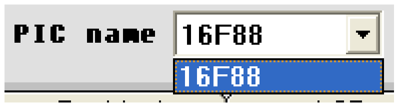
USB PIC Programmer CONNECTED and POWERED ON

System detect: 18-PIN 16F88
System will load all experiments for this PICtype

- The system will only load the experiments for the PIC type and PIC name inserted. The other PIC types would not be available

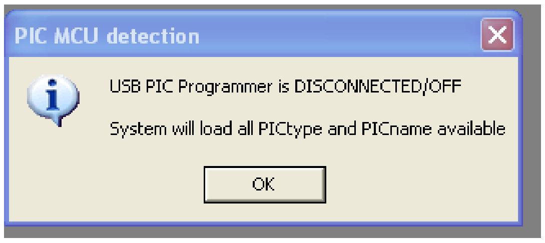


PIC type 18-PIN



PIC name 16F88

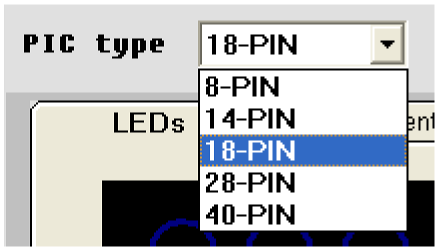
- Otherwise, if no PIC is inserted when the program is opened, the system will load all experiments for all the PIC types.



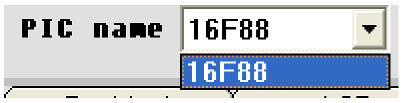
PIC MCU detection

USB PIC Programmer is DISCONNECTED/OFF

System will load all PICtype and PICname available



PIC type: 18-PIN

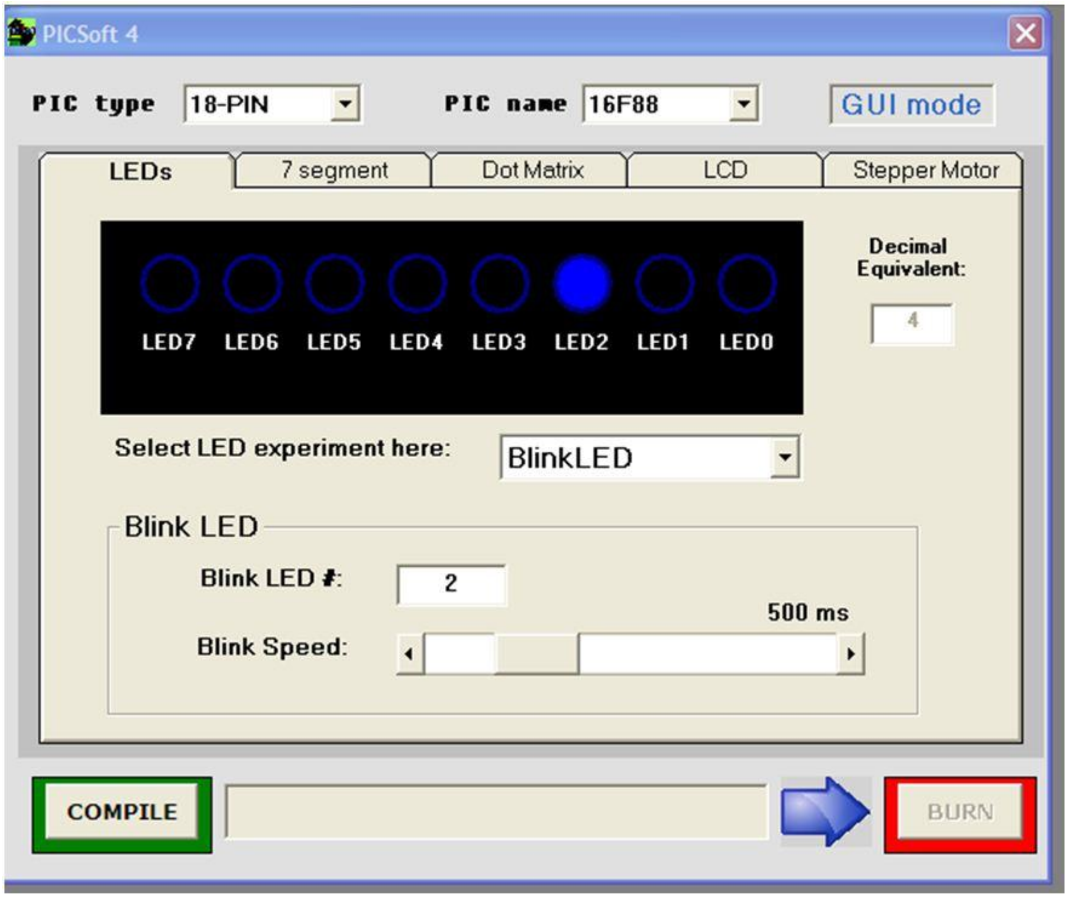


PIC name: 16F88

2.2 Familiarization with Software

a. Beginner / GUI mode

The Beginner/ GUI mode contain windows, buttons, drop-down menus, and other clickable objects that can be used to build the code without actual programming. The GUI windows contains options to choose the PIC MCU to be used, select the training kit module to be controlled, edit the output code and invoke compilation and burning. The Virtual Emulator also runs in conjunction with this mode.



PIC type: 18-PIN
PIC name: 16F88
Decimal Equivalent: 4
Select LED experiment here: BlinkLED
Blink LED #: 2
Blink Speed: 500 ms

Figure 3 **PICSoft main window**

b. Virtual Emulator

The virtual emulator can simulate the code and the way it would execute in the hardware components in the training kit. This is only available in Beginner mode.

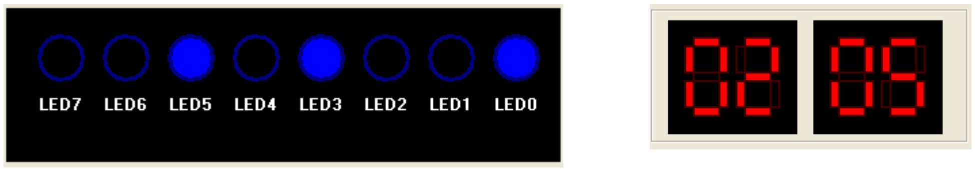
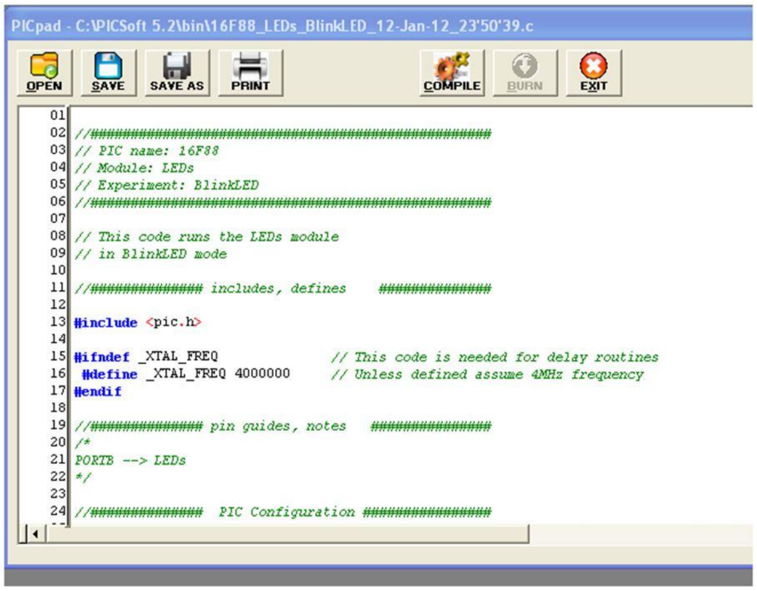


Figure 4 **Virtual emulator**

c. Advanced Editing Mode / PICpad

In ‘Advanced Editing Mode’, the user would be presented with a code editor named PICpad. This mode provides options for opening and saving C code files (*.C) , as well as for compiling and burning. It is also possible to load the code running at the Beginner /GUI mode directly to PICpad by clicking View Code (Figure 6) from Beginner/GUI mode. This option helps the user analyze and develop code.

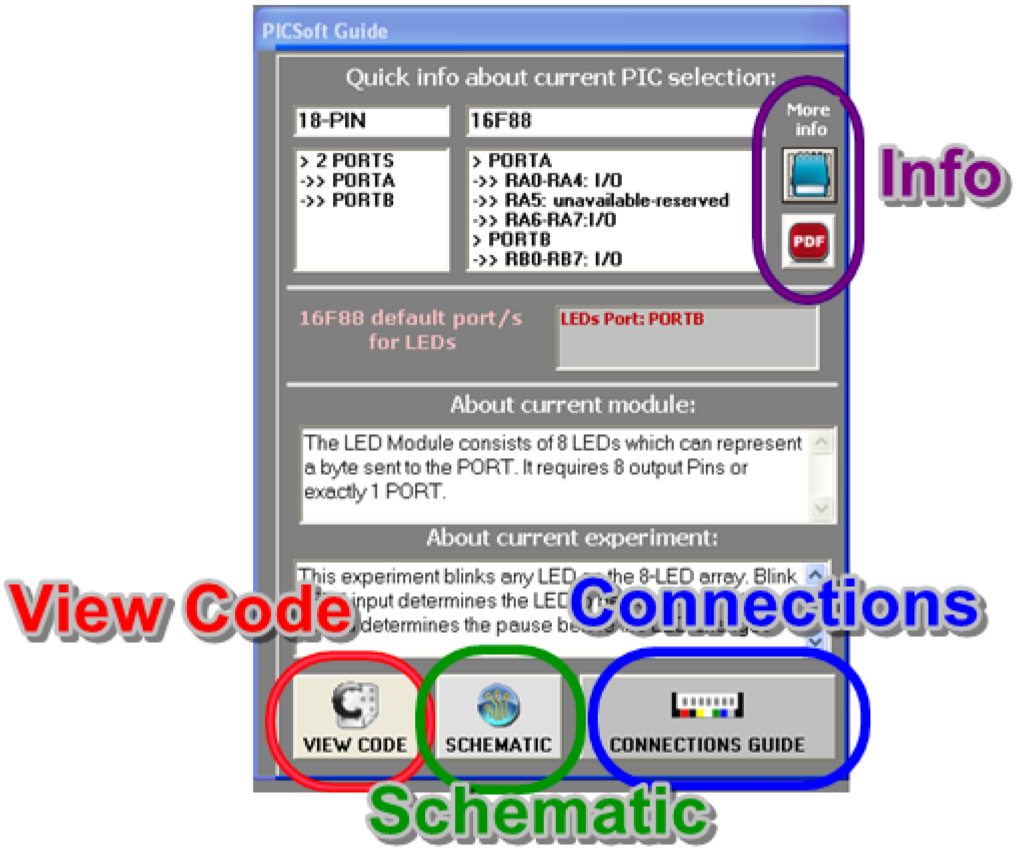

PICpad - C:\PICSoft 5.2\bin\16F88_LEDs_BlinkLED_12-Jan-12_23'50'39.c
OPEN
SAVE
SAVE AS
PRINT
COMPILE
BURN
EXIT

```

//##########################################################
// PIC name: 16F88
// Module: LEDs
// Experiment: BlinkLED
//##########################################################

// This code runs the LEDs module
// in BlinkLED mode

//################ includes, defines     ################

#include <pic.h>

#ifndef _XTAL_FREQ              // This code is needed for delay routines
 #define _XTAL_FREQ 4000000     // Unless defined assume 4MHz frequency
#endif

//################ pin guides, notes    #################
/*
PORTB --> LEDs
*/

//################  PIC Configuration #################
```

Figure 5 **PICpad code editor**

d. Software guides and information window

The project software includes a software guide and a hardware guide. The software guide provides general help in using the program, and information about the PIC MCU to be used.


PICSoft Guide

Quick info about current PIC selection:

18-PIN

16F88

> 2 PORTS
->> PORTA
->> PORTB

> PORTA
->> RA0-RA4: I/O
->> RA5: unavailable-reserved
->> RA6-RA7:I/O
> PORTB
->> RB0-RB7: I/O

16F88 default port/s for LEDs

LEDs Port: PORTB

About current module:

The LED Module consists of 8 LEDs which can represent a byte sent to the PORT. It requires 8 output Pins or exactly 1 PORT.

About current experiment:

This experiment blinks any LED on the 8-LED array. Blink ... input determines the LED ... determines the pause be...

Figure 6 **PICSoft guide window**

The program also allows easy access to a hardware guide consisting of step-by-step procedures on placing the PIC MCU in the ZIF Socket, connecting the module to the programmer, etc. This feature aims to lessen errors in working with the hardware. The user can also view the schematic representation of the interface.

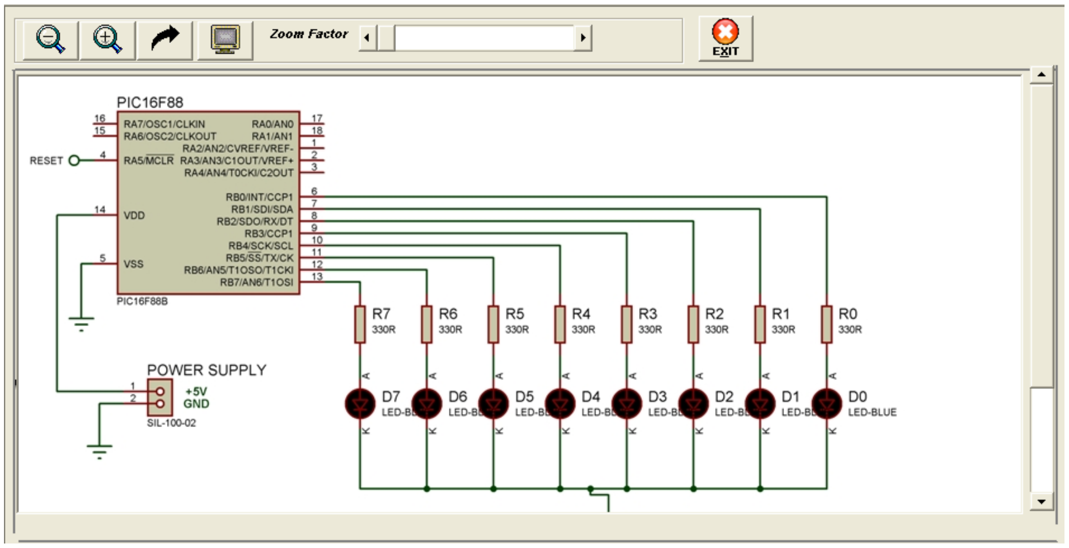


Figure 7 **Schematic view window**

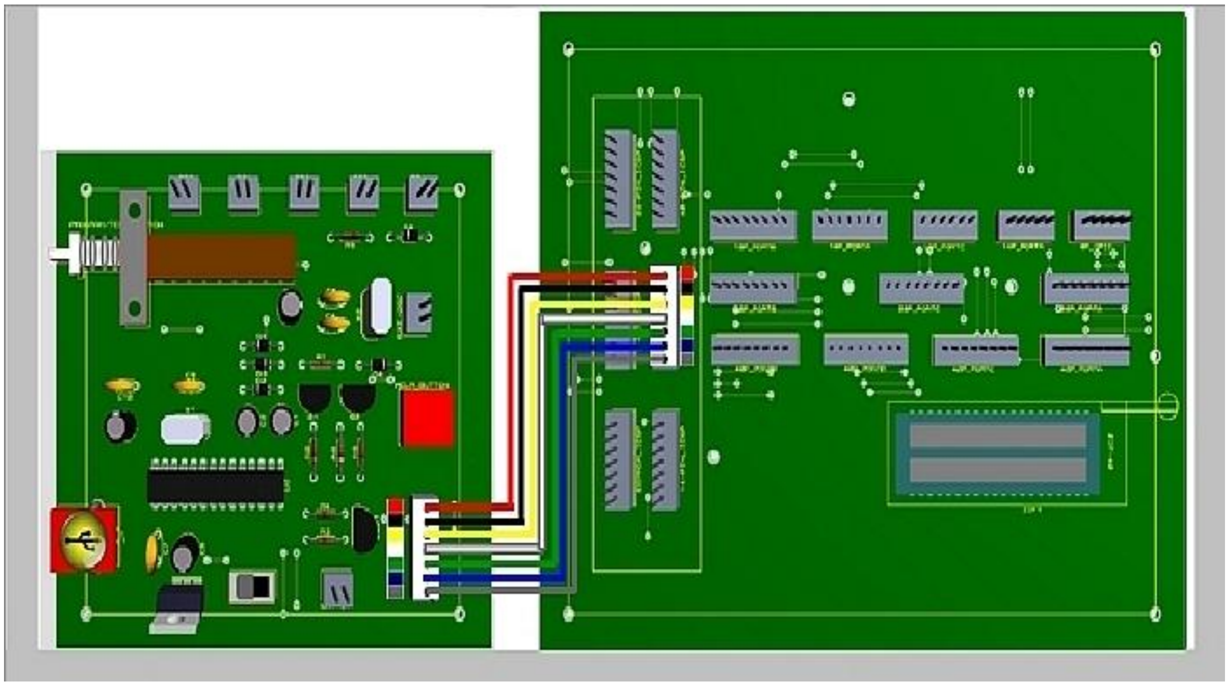


Figure 8 **Connections guide window**

2.3 C Language Structure

For modern systems, C is the programming language of choice because it is available for a wide range of systems and processors (including the PIC® microcontroller). C is often referred to as the *universal assembly language* because it is designed in such a way that it can access a system's lowest levels efficiently. High-level languages such as C can be easier for most beginners compared to learning Assembly Language (ASM). C language holds several advantages over other high- level languages such as speed, portability, and flexibility.

The C compiler included in the software package is the **Hi-Tech PICC Lite Compiler**. It is a Freeware ANSI C Compiler, designed primarily for the use of students and hobbyists for educational and small projects. The advantage of ANSI compilers are standardization, meaning the syntax of PICC Lite is similar to any other C compilers that follow the standard, such as MikroC, CCS C, Turbo C, LCC, GCC, etc. Thus, students who have a programming background in C would understand PICC easier.

Let's start with a short PIC C code that blinks one LED. This code blinks the LED connected to RB2 [pin2 of PORTB] third from the right in upright position.

```
//##################################################
// PIC name: 16F88
// Module: LEDs
// Experiment: BlinkLED
//##################################################

// This code runs the LEDs module
// in BlinkLED mode

//############## includes, defines    ##############

#include <pic.h>

#ifndef _XTAL_FREQ         // This code is needed for delay routines
 #define _XTAL_FREQ 4000000  // Unless defined assume 4MHz frequency
#endif

//############## pin guides, notes   ###############
```

```
19    /*
20    PORTB --> LEDs
21    */
22
23    //##############  PIC Configuration ################
24
25    __CONFIG(FOSC_INTOSCIO & WDTE_OFF & PWRTE_ON & MCLRE_ON & \
26      BOREN_OFF & LVP_OFF & CP_OFF & CPD_OFF & DEBUG_OFF & \
27      WRT_OFF );
28    __CONFIG(FCMEN_OFF & IESO_OFF );
29
30
31
32    //##################  main   #######################
33
34    main()
35    {
36
37    //         #########  register settings ############
38
39    ANSEL = 0;                //  Turn off ADC
40    OSCCON = 0b01100000;       //  Set intRC to 4 MHz
41    TRISB = 0;                //  Make PORTB Output
42
43    //         #########  initializations ##############
44
45    PORTB = 0;                //  Turn off all PORTB pins at start
46
47
48    //         ############# program loop ##############
49
50    while(1 == 1)               //  Loop Forever
51        {
52
53         RB2 = 1;                // LED on
54         __delay_ms(500);       //pause
55         RB2 = 0;                // LED off
56         __delay_ms(500);       //pause
57
58         }  // elihw
59    }  //  End
60
61    //#################### end  #######################
```

- Green statements are comments. Comments can either start with `//` (lines 7 and 8) or enclosed with `/*` and `*/` (lines 18-21)
- Comments do nothing regarding the execution of the program. These are used to explain certain parts of the program and increase code readability.
- Comments can also appear beside working statements, such as in lines 39-41.

- Code blocks like this `//#########################################################` In line 1 to 5; and `//############## includes, defines    ##############` in line 10 are commented lines used as code blocks. Code blocks divide a program into certain parts, making it clear and easy to understand. It simplifies programming in a way that the part of the code to be modified or studied is easily separated from other parts.
- Based from these code blocks, we will now analyze the blocks that make up the program, one by one.

Description Block (Lines 1-8)

```
1     //##################################################
2     // PIC name: 16F88
3     // Module: LEDs
4     // Experiment: BlinkLED
5     //##################################################
6
7     // This code runs the LEDs module
8     // in BlinkLED mode
```

This block is an entire commented block and actually does nothing. However, this block displays general information about the program, such as the PIC MCU and module to be used, as well as the specific experiment and description of what the program does.

Directives Block / Includes and Defines (Lines 10-16)

```
10    //############## includes, defines    ##############
11
12    #include <pic.h>
13
14    #ifndef _XTAL_FREQ         // This code is needed for delay routines
15     #define _XTAL_FREQ 4000000  // Unless defined assume 4MHz frequency
16    #endif
```

Directives are special commands to the compiler. The **#include** directive instructs the PICC compiler to add the contents of the **pic.h** header file into the program (line 12). The **#ifndef - #define - #endif** directives in lines 14-16 are used to define a constant. In this code, it is used to define the crystal oscillator frequency in millihertz (MHz). This is necessary for delay functions in the program.

Pin Guides Block (Lines 10-16)

```
18    //############## pin guides, notes   ###############
19    /*
20    PORTB --> LEDs
21    */
```

Same as the description block, this is a fully commented block and does nothing. It contains information regarding which PORT and PINS of the PIC MCU the module devices are connected. Line 20 informs that the LEDs are connected to PORTB of the PIC MCU for this experiment.

PIC Configuration Block (Lines 23-28)

```
23    //##############  PIC Configuration ################
24
25    __CONFIG(FOSC_INTOSCIO & WDTE_OFF & PWRTE_ON & MCLRE_ON & \
26      BOREN_OFF & LVP_OFF & CP_OFF & CPD_OFF & DEBUG_OFF & \
27      WRT_OFF );
28    __CONFIG(FCMEN_OFF & IESO_OFF );
```

The PIC MCU should be configured correctly to match the properties and requirements of the experiment to be done. The configuration bits coded in this block controls the features and settings of the PIC MCU.

FOSC_INTOSCIO selects the internal oscillator to be used; WDTE_OFF disables Watchdog Timer; PWRTE_ON enables Power-on Timer; MCLRE_ON enables Master Clear (Green Reset button); BOREN_OFF disables Brownout Detector; LVP_OFF disables Low Voltage Programming; CP_OFF disables Code Protection; CPD_OFF disables Code Data Protection; DEBUG_OFF disables debugger mode; WRT_OFF disables Flash Write Protection; FCMEN_OFF disables Fail-Safe Clock Monitor; and IESO_OFF disables Internal Oscillator Switchover.

Under simple I/O interfacing such as in most of the experiments in this workbook, these settings are recommended to be left unchanged. The topics that cover these settings are cumbersome and complex especially for beginners. Details on these features can be obtained from the datasheet of the PIC MCU in use. Look for this button inside PICSoft to access such information.

Main Block (Lines 32-61)

```
32    //###################  main   ######################
33
34    main()
35    {
36
37    //          #########  register settings ############
38
39    ANSEL = 0;                 //  Turn off ADC
40    OSCCON = 0b01100000;        //  Set intRC to 4 MHz
41    TRISB = 0;                 //  Make PORTB Output
42
43    //          #########  initializations ##############
44
45    PORTB = 0;                 //  Turn off all PORTB pins at start
46
47
48    //          ############# program loop ##############
49
50    while(1 == 1)                //  Loop Forever
51        {
52
53         RB2  = 1;                //  LED on
54         __delay_ms(500);        //pause
55         RB2  = 0;                //  LED off
56         __delay_ms(500);        //pause
57
58         }  // elihw
59    }  //  End
60
61    //#################### end  #######################
```

The codes inside the main function are the ones executed by the PIC MCU. Under normal circumstances, program execution starts at the first statement in main() after the opening bracket [{] (line 35) and terminates at the last statement in main(), before the closing bracket [}] (line 59).The main block itself contains several blocks of code, which are discussed shortly.

- **Register Settings block (Lines 37-41)**

```
37    //         #########  register settings ############
38
39    ANSEL = 0;                //  Turn off ADC
40    OSCCON = 0b01100000;       //  Set intRC to 4 MHz
41    TRISB = 0;                //  Make PORTB Output
```

ANSEL, OSCCON and TRISB are some of the internal registers of the PIC MCU.

ANSEL register controls the selectable analog inputs of the pins. Line 39 turns off ADC so that all of the pins can be used as digital I/O, which is the requirement to drive LEDs.

TRISB register controls the direction of PORTS and individual PINS. Line 41 sets the entire PORTB to become output by loading it with 0.

- **Initialization (Lines 43-45)**

```
43    //         #########  initializations ##############
44
45    PORTB = 0;                //  Turn off all PORTB pins at start
```

Initializations contain codes that prepare the variables and registers at the start of the program. PORTB is the PIC MCU register that controls the output of each PIN or the entire PORT. Line 45 turns off all PORTB pins by loading it with 0.

- **Program Loop (Lines 48-58)**

```
48    //          ############# program loop ##############
49
50    while(1 == 1)                 //  Loop Forever
51        {
52
53         RB2 = 1;                  // LED on
54         __delay_ms(500);         //pause
55         RB2 = 0;                  // LED off
56         __delay_ms(500);         //pause
57
58         }  // elihw
```

The Program Loop contains the codes that would be executed by the PIC MCU repeatedly in its normal operation. Line 50 uses a statement that would always be true (1==1) thus making the code inside it loop forever. Removing line 50 and its closing bracket at line 58 will only run the code once.

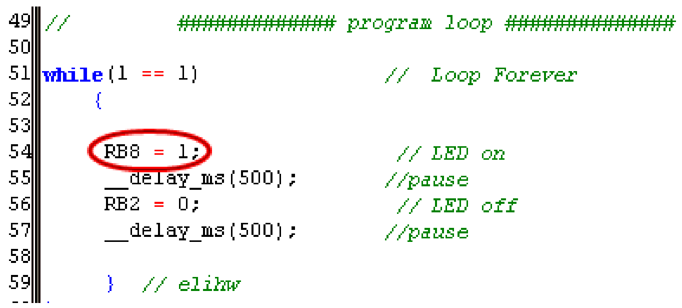
Actually, Lines 53-56 are the only operation that the PIC MCU does. Line 53 turns on the LED by giving RB2 (PORTB # 2) a logic 1. Line 55 turns of the LED by giving a logic 0 to RB2. Lines 54 and 56 provides a 500 ms delay to create the blinking effect. Removing either one of these delay lines would ruin the blink effect. Notice how the comments beside the statements make it very clear and easier to understand.

After the execution of this 4 lines, the program would loop back again to line 54, and repeats in a cycle until power is cut off or the loaded code in the PIC MCU is reset through the Green Reset button in the USB PIC Programmer.

Other blocks of codes can appear in the experiments. Some blocks are functional in one experiment but are not useful to another. Make sure to read all the comments provided inside the code blocks to understand the program.

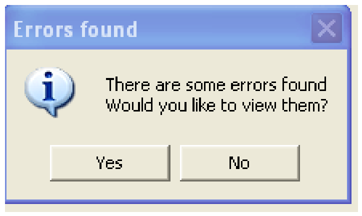
Compilation

- When in Advanced Mode / PICpad, the code must be saved before compiling. After successful compilation of the code to become a *.hex file, the code can then be burned into the device.
- In times when the compilation is not successful, it is necessary to examine and debug the code. The Hi-Tech PICC compiler automatically detects program errors, such as in syntax, structure, semantic, and some logic errors.
- PICPad, in conjunction with the Hi-Tech PICC Compiler, can display and describe error messages, as well as the lines of codes that caused the errors, or related to the cause.
- For example, suppose we intentionally corrupted part of the BlinkLED code to assign a logic 1 to a non-existent pin RB8 (PORTB pins only range from RB0 to RB7)



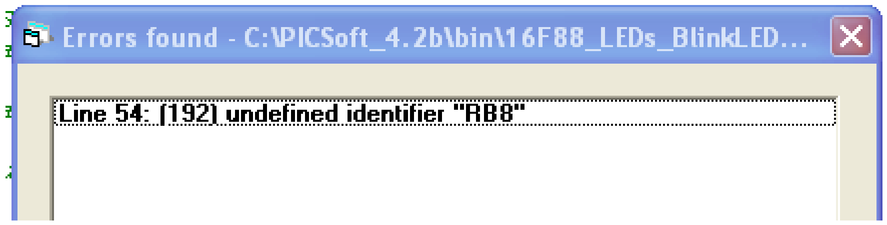
```
//         ############## program loop ###############

while(1 == 1)                // Loop Forever
    {

      RB8 = 1;                // LED on
      __delay_ms(500);       //pause
      RB2 = 0;                // LED off
      __delay_ms(500);       //pause

      }  // elihw
```

- After the code is saved and then compiled, these error prompt would appear



There are some errors found
Would you like to view them?

- If we choose to view the errors, a list of the detected errors would be displayed, along with the line number and description



```
Line 54: [192] undefined identifier "RB8"
```

- This feature can help in debugging programs, especially as the program increases its size.
- Note that most logical errors are not detected by the compiler, because they may perfectly follow the compiler's rules, but not the intended purpose of the program (or part of it).

3 LABORATORY EXPERIMENTS

The following section consists of the laboratory experiments for each of the training kit modules.

Interfacing with the LED Module

OBJECTIVES:

- To be able to use the LED Module of the PC-Based Microcontroller Programmer and Training Kit
- To be able to use the Beginner Mode of PICSoft to create a pattern for the LEDs
- To be able to edit and compile a C code using the code editor in Advanced Mode of PICSoft
- To be able to interface a 18-pin PIC MCU with LEDs

MATERIALS:

- PC-Based Microcontroller Programmer and Training Kit
- PC with Windows XP running PICSoft
- PIC16F88 18-Pin PIC MCU

INTRODUCTION:

LED module

The LED Module consists of an 8 LED array corresponding to binary output. There are two headers included, one I/O header and one Module Supply header. The LEDs represent the ON/OFF status of each pin in the PIC MCU port.

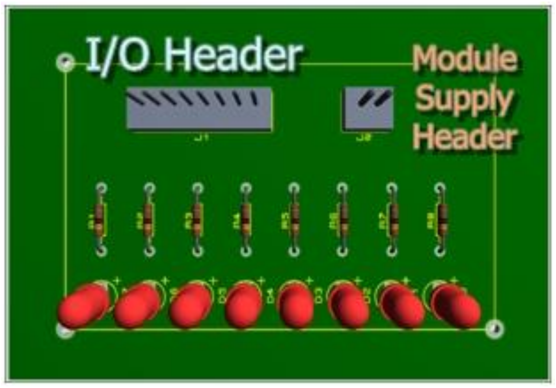


Figure 1 **LEDs Module**

PIC MCU PORTS

In this experiment, we will use the 18-Pin PIC MCU 16F88. It has two PORTS: PORTA and PORTB. Ports represent physical connection of the PIC MCU's processor to the outside world in the form of the pins. For this experiment, PORTB will be used for its whole 8-bits can be used for I/O.

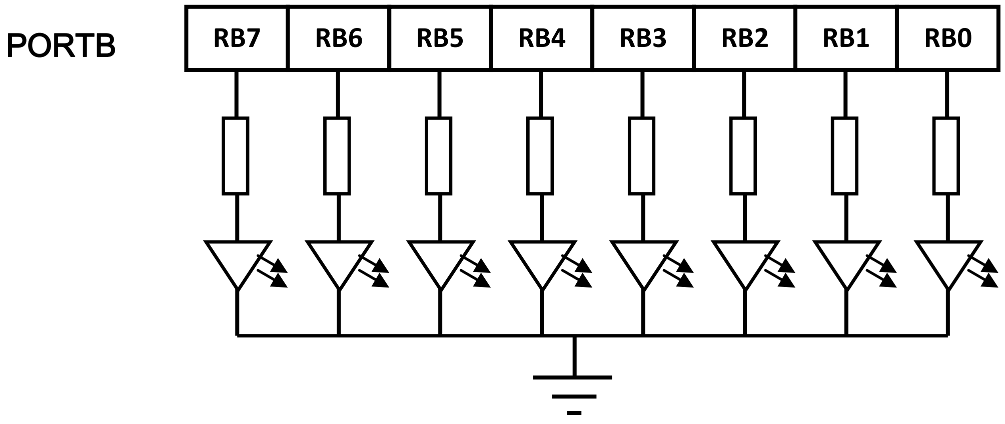


Figure 2 **PORTB connections to LEDs**

In coding, 1 simply represents ON and 0 for OFF. The ports can be accessed by the PORT name (such as PORTA, PORTB, etc.) or through their individual bits RB0, RB1, RB3…

Thus, if an 8-LED array is connected to PORTB, loading it with 11111111 will light all the LEDs, 11110000 will light the left half, 00000000 will turn off all the LEDs, and so on.

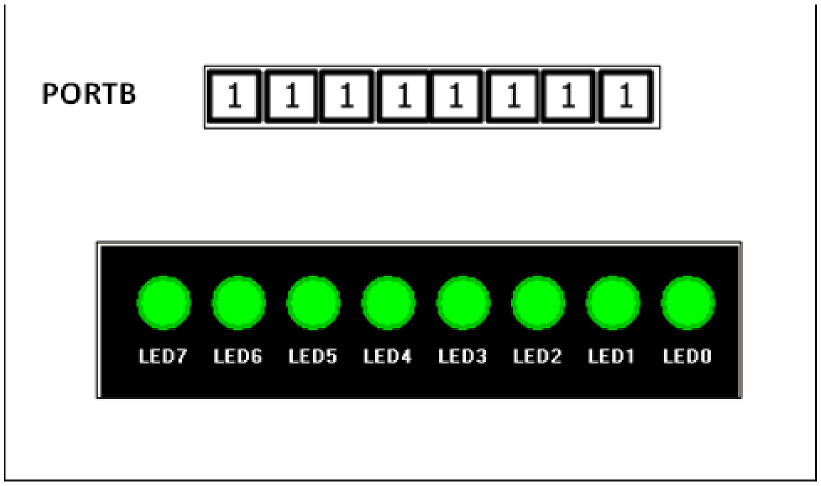


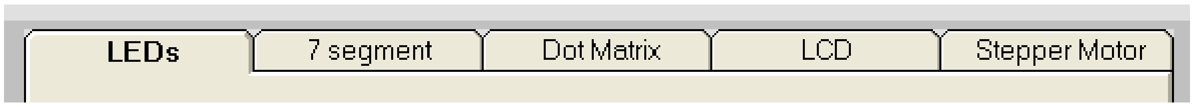
Figure 3 **Lighting all LEDs**

PROCEDURES:

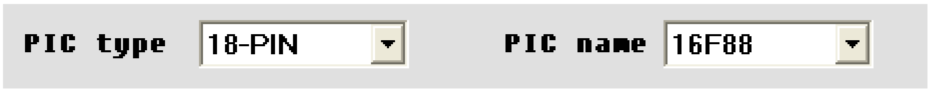
Before starting with the procedures, make sure you have already prepared the kit and installed the Device Driver and the Software. (See Chapter 2 for help).

A. Using the GUI and Virtual Emulator

1. Open PICSoft on the computer.
2. Select the LEDs in the devices tab.

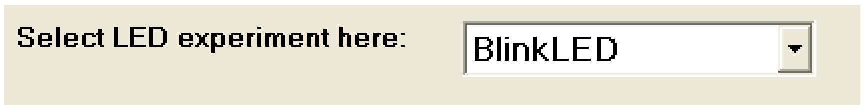


3. Select 18-PIN in the PIC type listbox and 16F88 in the PIC name listbox.

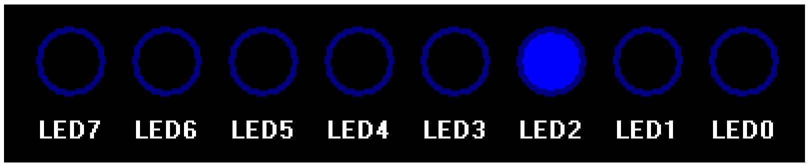


PIC type 18-PIN PIC name 16F88

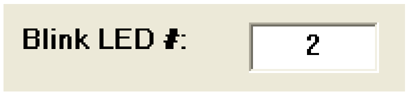
4. There are 6 available experiments in the LED experiment listbox. For this time, click the "BlinkLED".


Select LED experiment here: BlinkLED

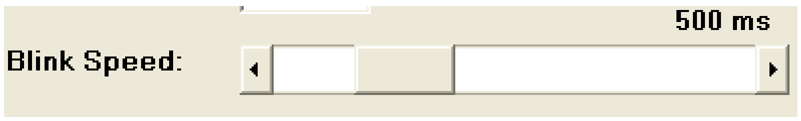
5. Notice the graphical animation of a blinking LED. This is the Virtual Emulator for the LEDs.



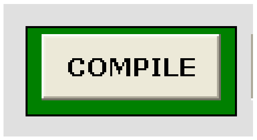
6. Change which LED# to blink by typing a digit (0 to 7) in the Blink LED# textbox


Blink LED #: 2

7. Change the Blink Speed in milliseconds by moving the slider.

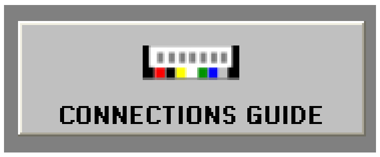

Blink Speed: 500 ms

8. Click 'Compile'.



9. Click 'OK' when the Successful Compilation prompt appears.

10. At this point the device connections must be done before burning. Click the "Connections Guide" at the bottom- right side.

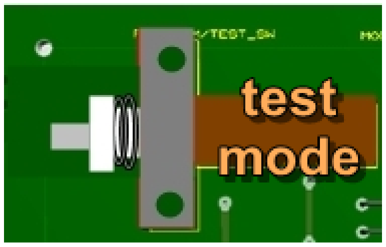


11. Follow the instructions and navigate through them by using the Prev and Next buttons. This guide will show you how to properly connect the headers for programming and testing of the compiled program.
12. Click ‘Exit Guide’ at the end of the guide (Connect USB page).

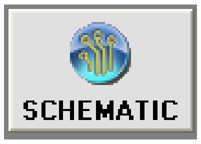
13. At this moment, click Burn.

14. A confirmation window would appear. You can disregard this and click “Proceed to Burning” (Green Button) if you have read the connections guide earlier.

15. Click ‘Yes” at the Burning Confirmation message box.
16. Wait until burning is finished. If all the proper connections are made, the Burning Complete prompt would appear. Close the window.
17. Push the Program/Test switch at the USB PIC Programmer to test the program.



18. Observe the actual LED Module and determine which LED is blinking and how fast. Don’t close PICSoft.
19. Click “Schematic”. You should now see the schematic representation of the connections done in the kit. This could also help you in programming.



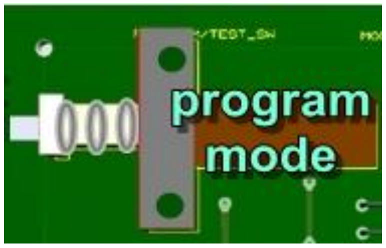
Questions:

- Determine which LED is blinking at the schematic diagram. Which pin and in which PORT is the blinking LED connected?

  ____

B. Using the Code Editor

1. Push the Program/Test switch back to go back into program mode. The LED should stop blinking for power supply will be cut off at this mode.



2. Go back to the main window by clicking Exit at the Schematic Window (if it is still open).
3. Click “View Code”.

4. The PICpad editor will appear with the C code of the BlinkLED experiment. It is advised to read section 2.3 “C Language Structure” beforehand to get acquainted with coding.
5. The green parts starting with // are comments. Comments are useful to programs to provide notes, reminders and keep code that would not be used for a time.

6. Locate this block of code near the bottom within the editor. This block is called the “Program Loop”. It contains the commands that control the LEDs.

```
51    while(1 == 1)                 //  Loop Forever
52        {
53
54         RB2 = 1;                  // LED on
55         __delay_ms(500);         //pause
56         RB2 = 0;                  // LED off
57         __delay_ms(500);         //pause
58
59        }  // elihw
```

7. At line 51, the while(1 == 1) ensures that the code keeps repeating. Removing this will only run the pattern once.
8. RB2 = 1;   at line 54 turns the LED connected to RB2 on. It can also be coded as

   PORTB = 0b00000100;   0b is the radix used for binary digits, whereas decimal numbers does not need any radix, such as

   PORTB = 4;   which also outputs the same pattern
9. On the other hand

   RB2 = 0;   at line 56 turns off the LED
10. __delay_ms(500);   at lines 55 and 57 creates the blinking effect and works by pausing execution by 500 millisecond
11. The closing bracket at line 59 closes the while loop.

Questions:

- What should be the code to light up the leftmost and rightmost LED?

  ____

- What will happen if the 2nd delay function is removed (line 7 in our example)?

  ____

12. Feel free to edit the code as desired. You can increase the number of blinking LEDs by accessing more pins (RB0,RB1,RB2,…RB7). You can also adjust the delay by replacing the number inside the __delay_ms( ) function.

    For example, this code blinks 2 LEDs at a faster rate.

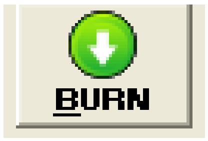
```
while(1 == 1)               //  Loop Forever
    {

     RB2 = 1;                // LED on
     RB3 = 1;                // LED on
     __delay_ms(250);       //pause
     RB2 = 0;                // LED off
     RB3 = 0;                // LED off
     __delay_ms(250);       //pause

     }  // elihw
```

13. This time, we will not change the PIC type and PIC name so we do not have to change the connections.

14. To test the codes, click the Save button.

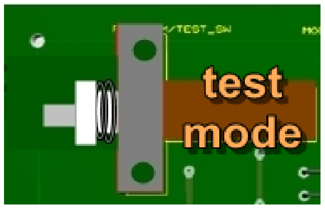
15. Click Compile button. If there are no errors, you would be prompted to burn the program. In case there are errors, compiling would not finish and the errors will be displayed.

16. Click Burn then click ‘Proceed with Burning’.



17. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Success prompt would appear. Close the window.

18. Push the Program/Test switch at the USB PIC Programmer to test the program. Observe the resulting pattern in the LED Module.



19. To modify the program, push it back to Program mode and follow the steps above in editing, saving, compiling, burning and testing.

EXERCISES:

1. Create a code that will alternately light the even-numbered LEDs (RB0, RB2, RB4, and RB6) and the odd-numbered LEDs (RB1, RB3, RB5, and RB7). Put the delay of your choice for the timing. You can write part of the code here before/after testing in the device.

2. Create a code that will create the effect in which the light is shifting to the left while speeding up and shift to the right while slowing down. You can write part of the code here before/after testing in the device.

3. Try the other five LED experiments in the GUI. Don't forget to customize them by using the options provided. You can also study and edit the program shown in the Visual Emulator by clicking the "View Code". Note their differences in the space provided below.

OBSERVATION:

CONCLUSION:

Interfacing with the 7-Segment Module

OBJECTIVES:

- To be able to use the 7-segment Module of the PC-Based Microcontroller Programmer and Training Kit
- To be able to use an input device in the form of a pushbutton
- To be able to create a simple Scoreboard
- To be able to interface a 18-pin PIC MCU with 7-segment Display

MATERIALS:

- PC-Based Microcontroller Programmer and Training Kit
- PC with Windows XP running PICSoft
- PIC16F88 18-Pin PIC MCU

INTRODUCTION:

7 segment module

The 7-segment module is composed of a pair of dual 7-segment displays and 4 SPST Momentary NO push buttons. This circuit implements multiplexing of the 4 digits, which requires less microcontroller pins to be used. This module also comprises 3 headers: the Module Supply header and two I/O headers.

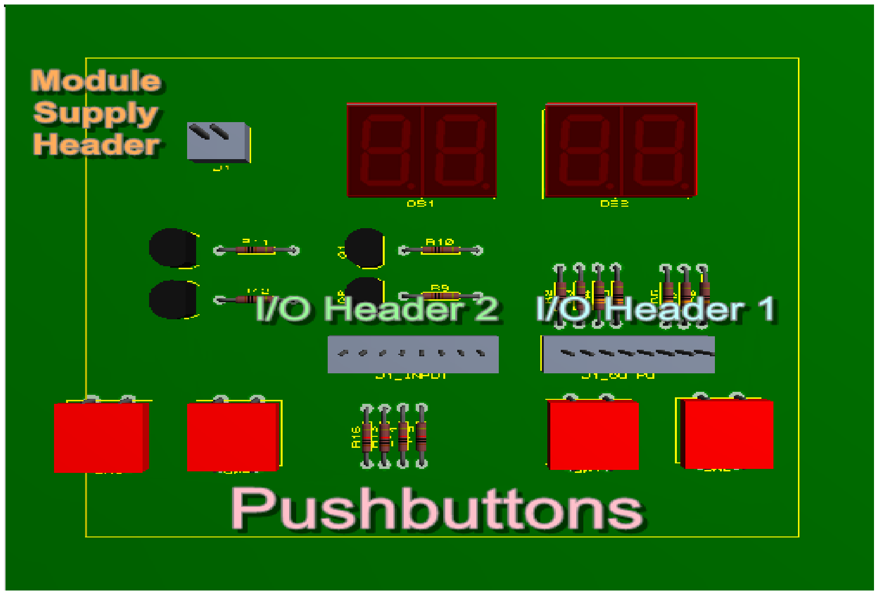


Figure 1 **7- segment Module**

The 7-segment display to be used is a Common-Anode type. In a common-anode display, the anodes of all the segment LEDs are tied together (see Figure 3) and then this common point is connected to V supply voltage. A required segment is turned on by applying a logic 0 to the cathode of this segment.

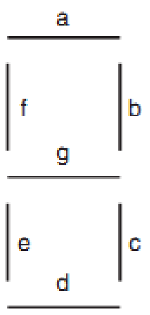


Figure 2 **Segment names of a 7-segment display**

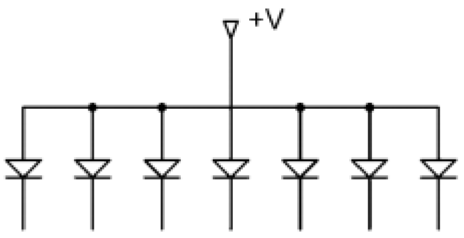


Figure 3 **Common Anode display**

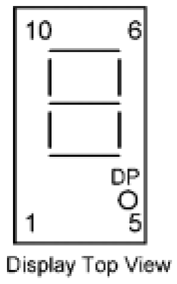


Figure 4 **Pin config of a 7-segment display**

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 1 **Pin config of a 7-segment display**

The relationship between the displayed numbers and the data to be sent to PORTB is shown in Table 2. The display is connected to the microcontroller using segments a to g. In this Table, x is a don't care entry, taken as 0 and is used to make the bit number 8. For example, to display number 2, we have to send binary 01011011 to PORTB. Similarly, to display number 8, we have to send binary number 01111111 to PORTB.

Number	x g f e d c b a
0	0 0 1 1 1 1 1 1
1	0 0 0 0 0 1 1 0
2	0 1 0 1 1 0 1 1
3	0 1 0 0 1 1 1 1
4	0 1 1 0 0 1 1 0
5	0 1 1 0 1 1 0 1
6	0 1 1 1 1 1 0 1
7	0 0 0 0 0 1 1 1
8	0 1 1 1 1 1 1 1
9	0 1 1 0 1 1 1 1

Table 2 **Displayed number and data sent to PORTB**

To control four 7-segment display digits simultaneously, one must employ "multiplexing" of the digits. The segments of the displays are connected in parallel and their common anodes are driven separately, each one for a brief period of time. For example, to display number 25, we have to send 2 to the first digit and enable its common point. After a few milliseconds, number 5 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.

- **PIC MCU Ports**

For this experiment, both PORTA and PORTB will be used.CA stands for Common Anode. PB stands for Push Button.

PORTB

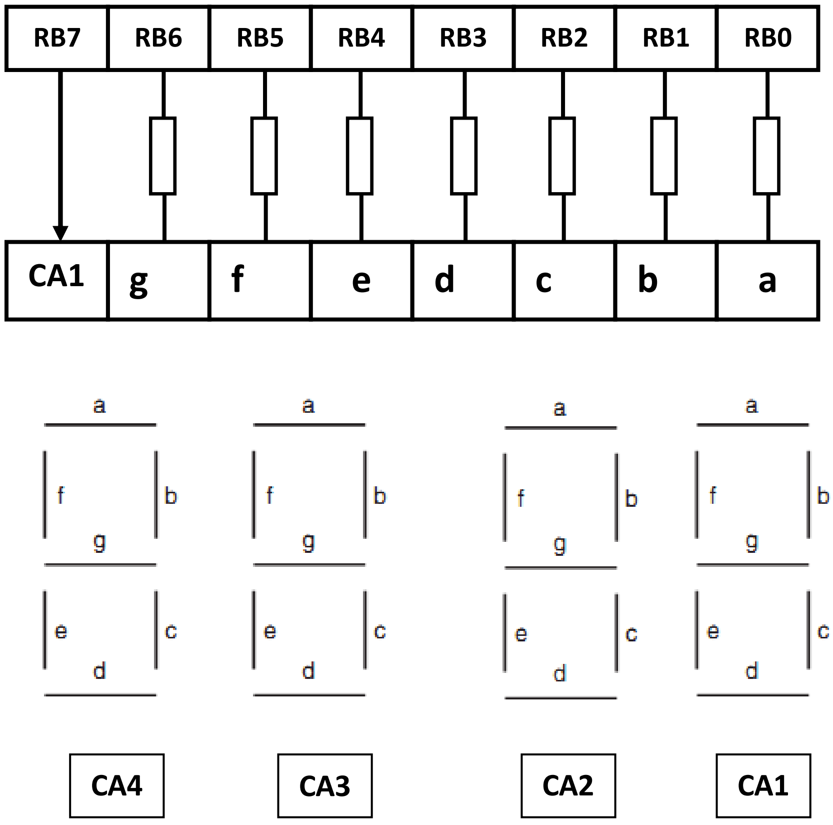


Figure 5 **Pin connections of PORTB**

PORTA

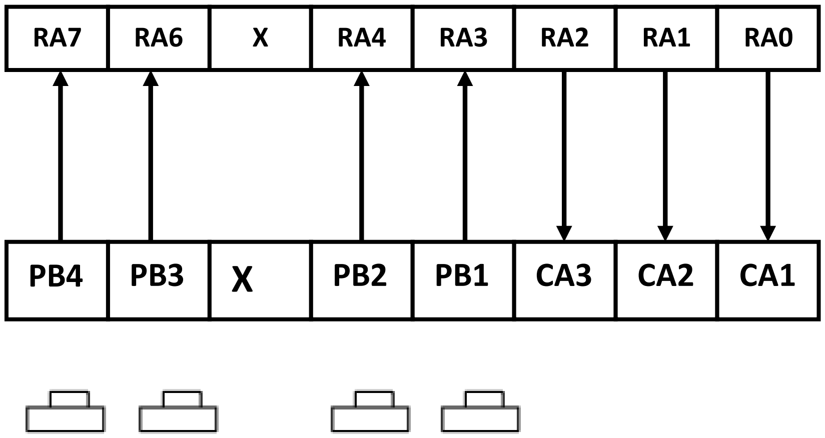


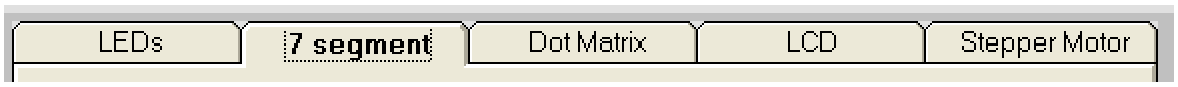
Figure 6 **Pin connections of PORTA**

<u>PROCEDURES:</u>

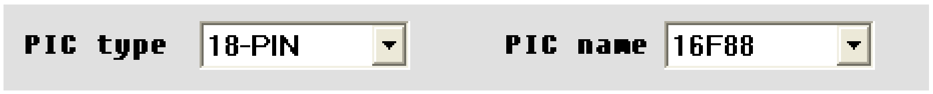
Before starting with the procedures, make sure you have already prepared the kit and installed the Device Driver and the Software. (See Chapter 2 for help).

A. Using the GUI and Virtual Emulator

1. Open PICSoft on the computer.
2. Select the 7-segment in the devices tab.



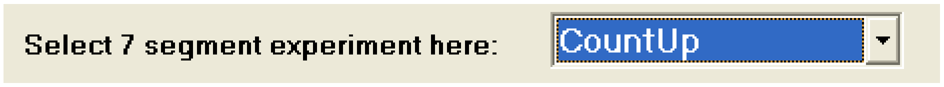
3. Select 18-PIN in the PIC type listbox and 16F88 in the PIC name listbox.



PIC type 18-PIN

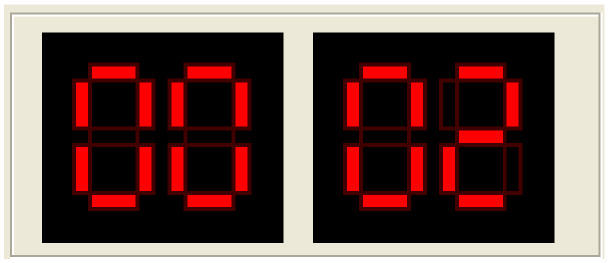
PIC name 16F88

4. There are 5 available experiments in the 7-segment experiment listbox. For this time, click the “Count Up”.

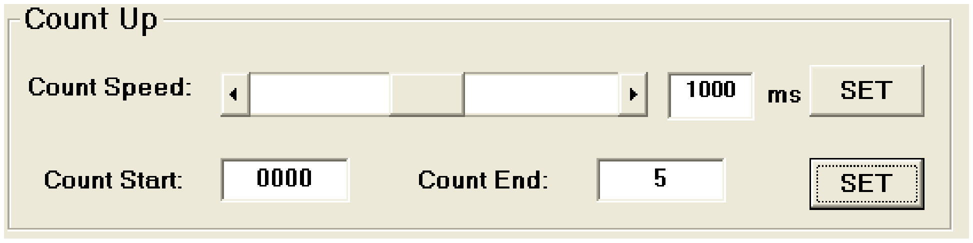


Select 7 segment experiment here: CountUp

5. Notice the graphical animation of the numerical counting. This is the Virtual Emulator for the 7-segment.



6. Change the starting and ending count, as well as the count speed.



Count Up

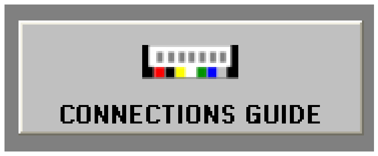
Count Speed: 1000 ms SET

Count Start: 0000 Count End: 5 SET

7. Click ‘Compile’.

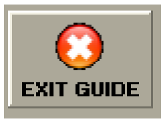
8. Click ‘OK’ when the Successful Compilation prompt appears.

9. At this point the device connections must be done before burning. Click the “Connections Guide” at the bottom- right side.

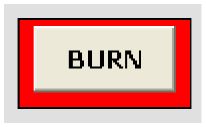


10. Follow the instructions and navigate through them by using the Prev and Next buttons. This guide will show you how to properly connect the headers for programming and testing of the compiled program.

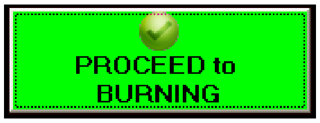
11. Click Exit at the end of the guide (Connect USB page).



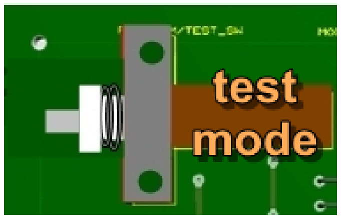
12. At this moment, click Burn.



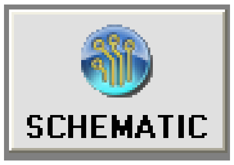
13. A confirmation window would appear. You can disregard this and click "Proceed to Burning" if you have read the connections guide earlier.



14. Click 'Yes" at the Burning Confirmation message box.
15. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Complete prompt would appear. Close the window.
16. Push the Program/Test switch at the USB PIC Programmer to test the program.



17. Observe the actual 7-segment Module. Don't close PICSoft.
18. Click "Schematic". You should now see the schematic representation of the connections done in the kit. This could also help you in programming.



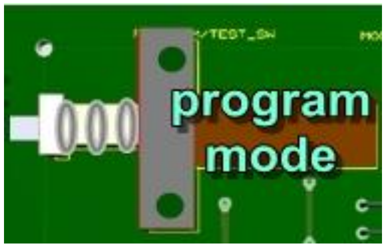
Question:

- In which PORT are the cathodes connected? What about their common anodes?

  ____

B. Using the Code Editor

1. Push the Program/Test switch back to go back into program mode.



2. Go back to the main window by clicking Exit at the Schematic Window (if it is still open).
3. Click "View Code".

4. The PICpad editor will appear with the C code of the CountUp experiment. It is advised to read section 2.3 "C Language Structure" beforehand to get acquainted with coding.
5. As you will notice, the code is long and contains some parts that can be difficult to understand. Make sure to read all of the comments to help you understand the purpose of each line.

6. Notice the `//########### Global Variables and Arrays ##########` block

   It contains variables and arrays used in the program. The `const int LEDChar[]`

declaration contains the binary patterns used in forming the numbers and characters.

```
const int LEDChar[]= { 0b00111111,     // 0
                       0b00000110,     // 1
                       0b01011011,     // 2
                       0b01001111,     // 3
```

...

Question:

- What would be the binary number to be used when displaying…
    - ‘c’ ______________________________
    - ’H’ ______________________________
    - ’L’ ______________________________
    - ’P’ ______________________________
    - ’S’ ______________________________
    - ’u’ ______________________________

7. Examine the `//################  Interrupt #####################` block. The interrupt block contains code that updates the 7-segment displays. It is called every 10ms.

```
  void interrupt tmr0int(void)
  {                                 // Respond to Timer 0  Interrupt
//this interrupt updates the 7segment displays every 10 ms
 TMR0 = 216;     // reset TMR0 starting value to start counting again

  PORTB = d1 ;        // Load Digit 1 pattern to cathodes

  RB7 = 1;
  RA0 = 0;             // common anode 1 on
  RA1 = 1;
  RA2 = 1;

 __delay_ms(2);              // short delay for persistence of vision

  PORTB = d2 ;        // Load Digit 2 pattern to cathodes

  RB7 = 0;              // common anode 2 on
93      RA0 = 1;
94      RA1 = 1;
95      RA2 = 1;
96
97      __delay_ms(2);           // short delay for persistence of vision
98
99      PORTB = d3 ;       // Load Digit 3 pattern to cathodes
100
101     RB7 = 1;
102     RA0 = 1;
103     RA1 = 1;
104     RA2 = 0;           // common anode 3 on
105
106     __delay_ms(2);     // short delay for persistence of vision
107
108     PORTB = d4 ;        // Load Digit 4 pattern to cathodes
109
110     RB7 = 1;
111     RA0 = 1;
112     RA1 = 0;           // common anode 4 on
113     RA2 = 1;
114
115     __delay_ms(2);       // short delay for persistence of vision
116
117     TMR0IF = 0;            // clear interrupt flag
118     }  //  tmr0int
```

Lines 81, 90, 99, and 108 loads the binary pattern from LEDChar[ ] array to the 7-segment cathodes through PORTB. Lines 84, 92, 104, and 112 switch each of the anodes 'on' in sequence. The 2ms delay in lines 88, 97, 106, and 115 provides the very short delay between switching of the digits, creating the illusion that the 4 digits are lighting up simultaneously, but are actually lighting up one after another, in a very short period of time. This effect is called "persistence of vision".

8. Now go to the `//            ############# program loop ##############` block

```
176          while(1 == 1)                //  Loop Forever
177
178          {
179                for (ctr=0000;ctr<=9999;ctr++)
180                {
…                        …
225                 d1 = LEDChar[dig1] ^ 0b11111111 ;    // display Digit 1
226                   d2 = LEDChar[dig2] ^ 0b11111111 ;    // display Digit 2
227                   d3 = LEDChar[dig3] ^ 0b11111111 ;    // display Digit 3
228                   d4 = LEDChar[dig4] ^ 0b11111111 ;    // display Digit 4
229                        __delay_ms(222);        //delay before next count
230                 }
231           } // elihw
```

9. Line 179 is the for loop that makes the digit count from 0000 to 9999. Changing the numbers would change the count start and ending.

Questions:

- What would happen if the for loop is `for (ctr=0000;ctr>9999;ctr++)?`

  ____

- What would happen if we change the loop to

  `for (ctr=0000;ctr<9999;ctr++)`?

  ____

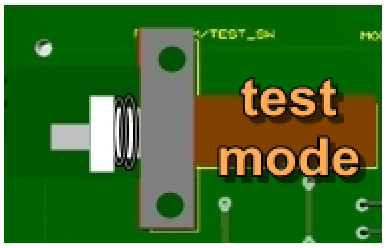
10. Lines 225-228 gets the pattern from the LEDChar[ ] array and XOR it with 0b11111111 to negate each bit. This is necessary because the 7-segment used is common-anode type, where the patterns are loaded in the cathodes.
11. `__delay_ms(500);` at line 229 creates the pause of 222 milliseconds before incrementing
12. Feel free to edit the code as desired. You can adjust the count start and end. You can also adjust the delay by replacing the number inside the `__delay_ms(222);` function.
13. This time, we will not change the PIC type and PIC name so we do not have to change the connections.
14. To test the codes, click the Save button.

15. Click Compile button. If there are no errors, you would be prompted to burn the program. In case there are errors, compiling would not finish and the errors will be displayed.

16. Click Burn then click ‘Proceed with Burning’.

17. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Success prompt would appear. Close the window.

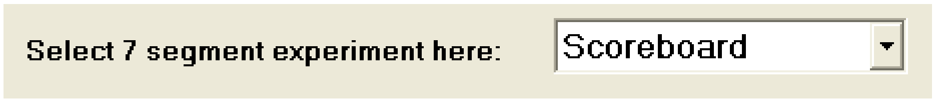
18. Push the Program/Test switch at the USB PIC Programmer to test the program. Observe the sequence in the 7-segment display.



19. To modify the program again, push it back to Program mode and follow the steps above in editing, saving, compiling, burning and testing back.

C. Programming the Scoreboard

1. If it is closed, open PICSoft.
2. Select the 7-segment in the devices tab.
3. Don’t change the PICtype and PICname settings (18-PIN, 16F88)
4. Choose “Scoreboard” in the experiments list.



Select 7 segment experiment here: Scoreboard

5. Click the 4 scoring buttons to test the emulator (L+2,L+1,R+2,R+1). There are 2 teams (Left and Right) and two buttons for each.

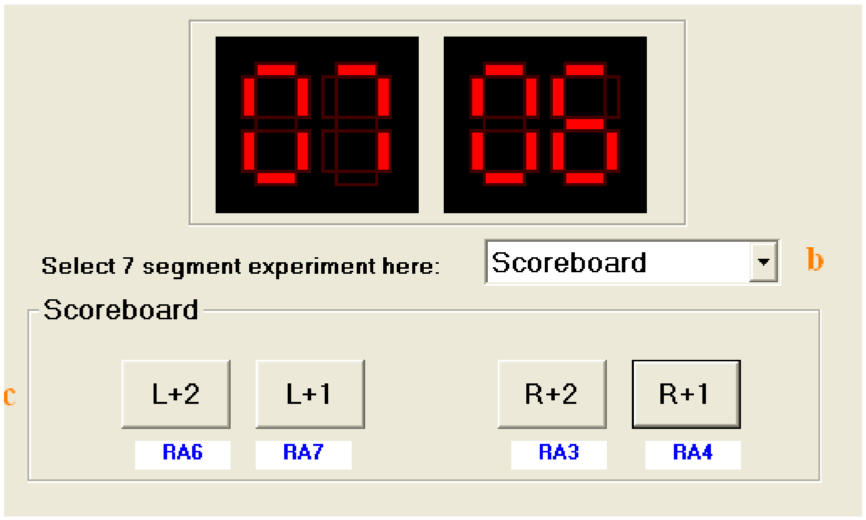


Figure 7 **Scoreboard emulation**

6. Click “View Code”.

7. Go to the `//          #########  register settings ############` block

```
126   ANSEL = 0;                   //  Turn off ADC
127   OSCCON = 0b01100000;          //  Set intRC to 4 MHz
128
129   TRISB = 0b00000000;         //  Set PORTB as output
130                        // Set RB0 - RB6 Output for the 7 segments (A - G)
131                                        // Set RB7 output for the CA of D1
132
133    TRISA0 = 0;                     // RA0 -- CA of D2
134    TRISA1 = 0;                     // RA1 -- CA of D3
135    TRISA2 = 0;                     // RA2 -- CA of D4
136    TRISA3 = 1;                     // push button input 1
137    TRISA4 = 1;                     // push button input 2
138    TRISA6 = 1;                     // push button input 3
139    TRISA7 = 1;                     // push button input 4
```

8. Line 126 Turns off ADC (Analog-to-Digital Converter) of PIC for we are only using digital I/O.

9. Line 127 sets the Internal Oscillator to 4 MHz.
10. Line 129 sets PORTB as output for the segment cathodes and the first common anode by accessing the TRISB register. Setting the TRIS of a pin/port as 0 makes it an output, while 1 makes it input.
11. Lines133-139 sets RA0:2 as output for the three common anodes and RA3,4,6,7 as input for the pushbuttons.
12. Now, look at the `//          ############# program loop ##############` block starting at Line 177. Here is one block of 'if' code that polls the buttons.

```
if (RA4 == 0)       // if Pushbutton 1 is clicked - > Right Team + 1
    {
        if (ctr >= 99)                       // if score exceeds 99
         {ctr = 0;                           // reset to 0
         }
         __delay_ms(100);                // delay for debouncing button;
         ctr++;                              // increment count by 1
     }
```

13. As explained in the comments provided, this block checks the first pushbutton (R+1) if it is clicked, then increments the score if it is. It is compared to 0 `[ if (RA4 == 0) ]` because the button is initially pulled-high by a 10k resistor, waiting to be shorted to GND if clicked. Thus, RA4 is always equals 1, but goes down to 0 if it is clicked. The schematic is shown in figure 8.

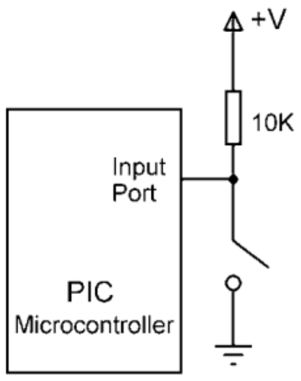


Figure 8 **Pulled-up input pin**

20. Feel free to edit the code as desired. You can change what each button does. For example, this simple change of code would subtract 3 from the score instead of + 1.

```
if (RA4 == 0)      // if Pushbutton 1 is clicked - > Right Team - 3
   {
       if (ctr >= 99)                     // if score exceeds 99
        {ctr = 0;                         // reset to 0
        }
        __delay_ms(100);          // delay for debouncing button;
        ctr = ctr - 3;                   // decrement count by 3
    }
```

21. We will not change the PIC type and PIC name so we do not have to change the connections.

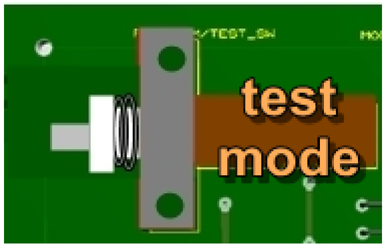
22. To test the codes, click the Save button.

23. Click Compile button. If there are no errors, you would be prompted to burn the program. In case there are errors, compiling would not finish and the errors will be displayed.

24. Click Burn then click 'Proceed with Burning'.

25. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Success prompt would appear. Close the window.

26. Push the Program/Test switch at the USB PIC Programmer to test the program. Observe the actual application in the 7-segment Module.



27. Use the 4 pushbutton switches to increment the score for both teams. Use the Green Reset button in the USB PIC Programmer to reset the scores.

28. To modify the program, push it back to Program mode and follow the steps above in editing, saving, compiling, burning and testing back.

EXERCISES:

1. Modify the code of the scoreboard so that each of the pushbuttons will score by 3s and 4s for both teams. You can write part of the code here before/after testing in the device.

2. Try the Count Down and other experiments in the GUI under the 7segment tab. Do not forget to customize it by using the options provided. You can also study and edit the program shown in the Visual Emulator by clicking the “View Code”. What are the differences between the code of Count Up and Count Down experiments?

OBSERVATION:

CONCLUSION:

Interfacing with the Dot Matrix Module

OBJECTIVES:

- To be able to use the Dot Matrix Module of the PC-Based Microcontroller Programmer and Training Kit
- To be able to display a message in various patterns using a 8x8 Dot Matrix
- To be able to program a 18-pin PIC MCU

MATERIALS:

- PC-Based Microcontroller Programmer and Training Kit
- PC with Windows XP running PICSoft
- PIC16F88 18-Pin PIC MCU

INTRODUCTION:

Dot Matrix module

The Dot Matrix module includes an 8x8 dot matrix and a 74LS138 decoder IC that runs the cathodes of the matrix through multiplexing. The IC reduces the required number of MCU pins for interfacing. This module also contains 3 headers: the Module Supply header and two I/O headers.

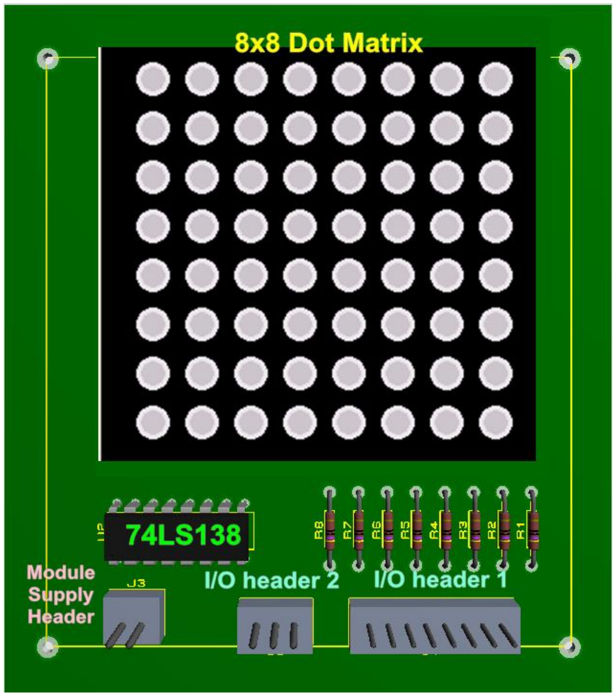


Figure 1 **Dot Matrix Module**

- **Dot Matrix and PIC MCU Ports**

  The dot Matrix LED to be used for the project is an 8x8 d5mm High Brightness Row anode- Column Cathode type. To light up any LED in the matrix, that LED's anode must have a 1 and it's cathode must have a 0. To form characters, each line of binary data that forms the pattern is sent to the column anodes, then the cathode of that entire column is enabled, each one for a brief period of time. For example, to display the letter A, we have to send 11111000 to the first column and enable that column by sending a 0 to their cathode. After a few milliseconds, the next line 11111100 is sent to the next and that line is enabled. When this process is repeated continuously in a very fast manner, the user sees as if all the lines are enabled simultaneously.

`

PORTA through 74LS138

PORTB

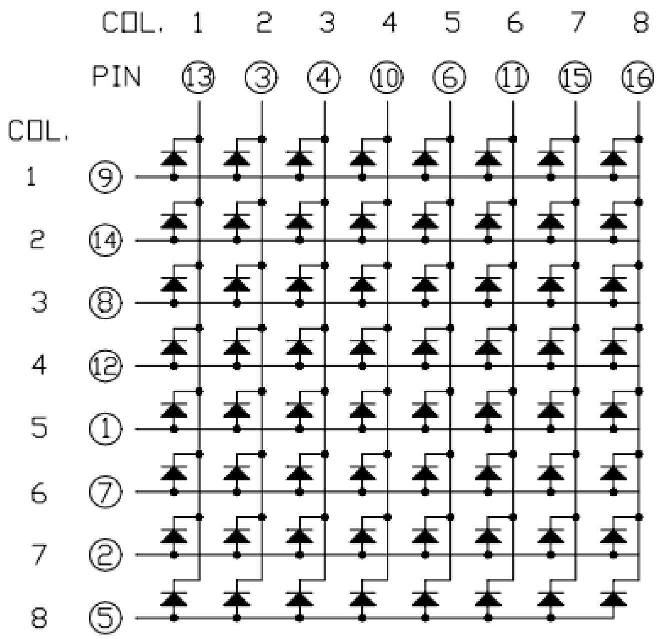


Figure 2 **8x8 Dot Matrix Pin Config**

DATA		DATA w/ 1s colored
0	11111000	11111000
1	11111100	11111100
2	00110110	00110110
3	00110011	00110011
4	00110011	00110011
5	00110110	00110110
6	11111100	11111100
7	11111000	11111000

Table 1 **8x8 Dot Matrix character ‘A’ Representation**

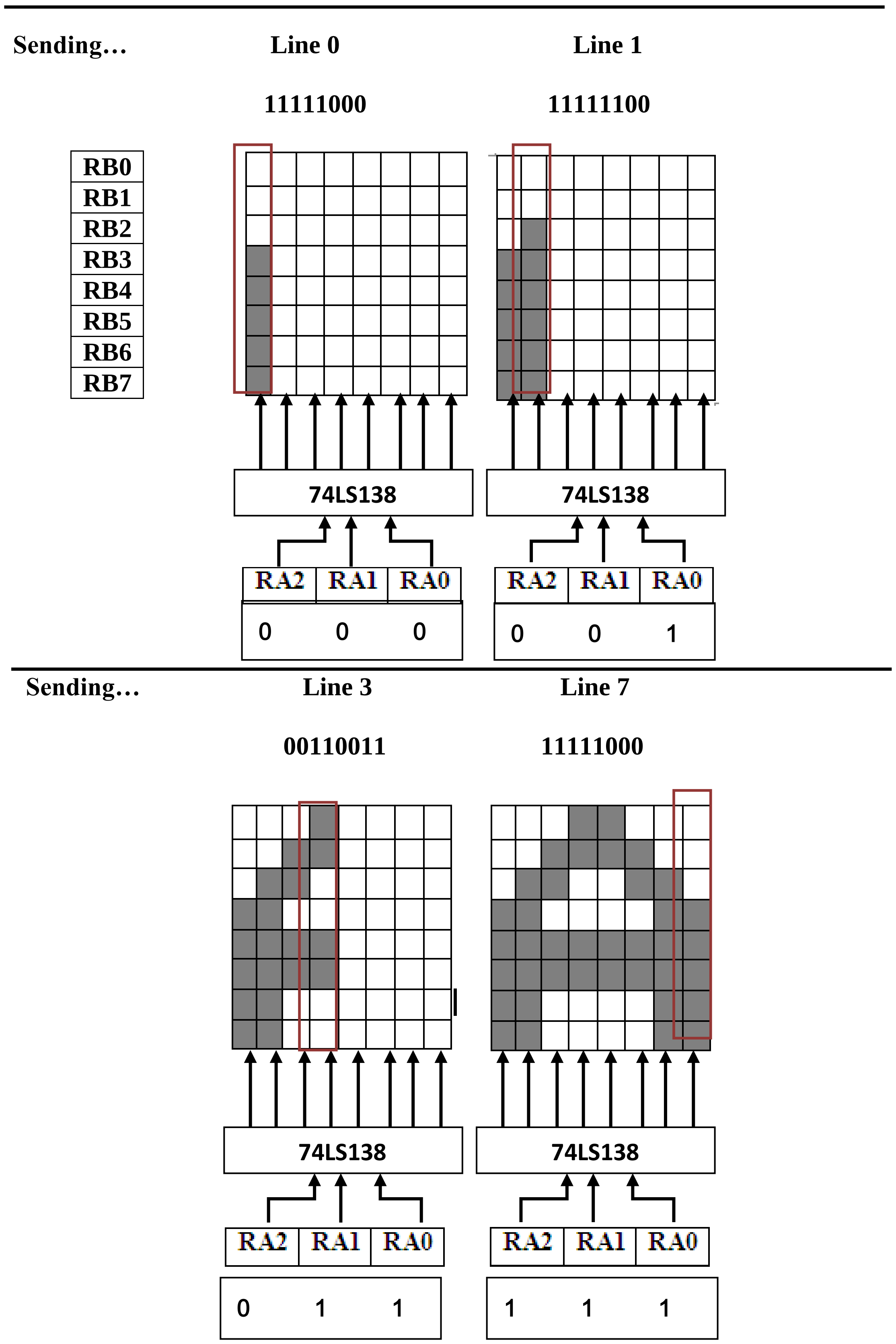


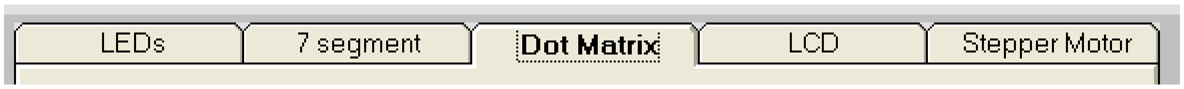
Figure 3 **Sending the lines of an 'A' to dot matrix**

PROCEDURES:

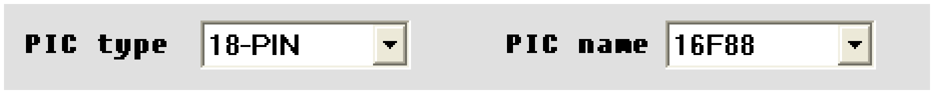
Before starting with the procedures, make sure you have already prepared the kit and installed the Device Driver and the Software. (See Chapter 2 for help).

A. Using the GUI and Virtual Emulator

1. Open PICSoft on the computer.
2. Select the Dot Matrix in the devices tab.

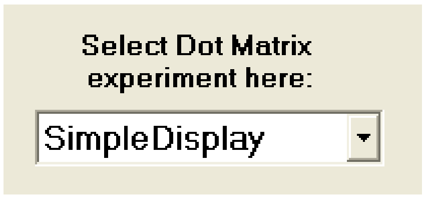


3. Select 18-PIN in the PIC type listbox and 16F88 in the PIC name listbox.



PIC type 18-PIN PIC name 16F88

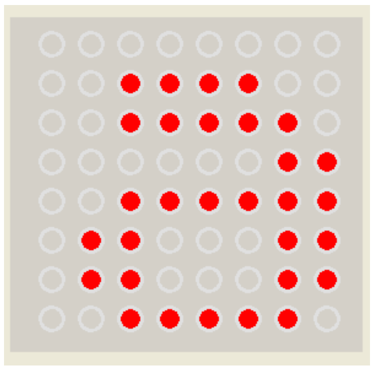
4. There are 3 available experiments in the Dot Matrix experiment listbox. For this time, click the “Simple Display”.



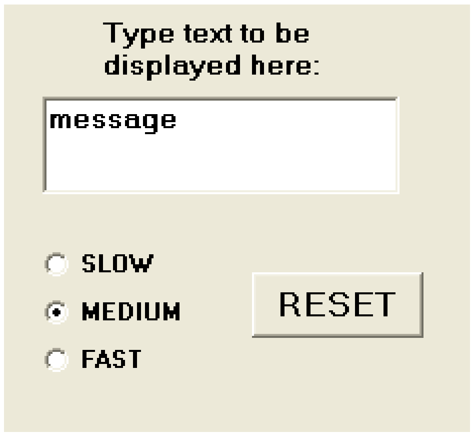
Select Dot Matrix experiment here:

SimpleDisplay

5. Notice the message displayed character by character. This is the Virtual Emulator for the Dot Matrix.



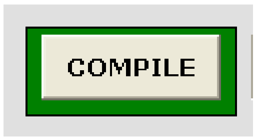
6. Change the message string to be displayed. Choose the speed (slow, medium, or fast). Click Reset after modifying to update the emulation.


Type text to be displayed here:

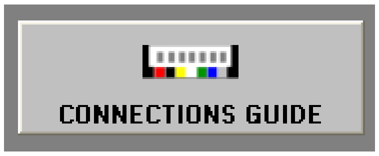
message

SLOW [✓ MEDIUM] FAST

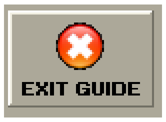
7. Click ‘Compile’.



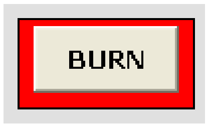
8. Click ‘OK’ when the Successful Compilation prompt appears.
9. At this point the device connections must be done before burning. Click the “Connections Guide” at the bottom- right side.



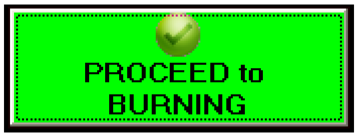
10. Follow the instructions and navigate through them by using the Prev and Next buttons. This guide will show you how to properly connect the headers for programming and testing of the compiled program.
11. Click Exit at the end of the guide (Connect USB page).



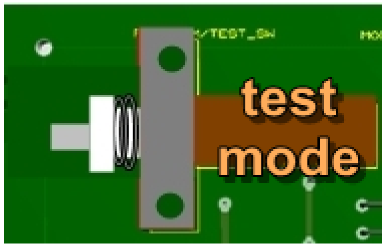
12. At this moment, click Burn.



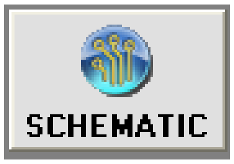
13. A confirmation window would appear. You can disregard this and click “Proceed to Burning” if you have read the connections guide earlier.



14. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Success prompt would appear. Close the window.
15. Push the Program/Test switch at the USB PIC Programmer to test the program.



16. Observe the program if it is running properly. Don’t close PICSoft.
17. Click “Schematic”. You should now see the schematic representation of the connections done in the kit. This could also help you in programming.



Question:

- In which PORT is the 74LS138 connected?

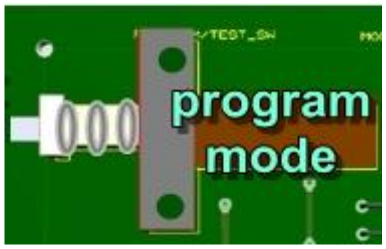
  ____

- How many inputs does the 74LS138 take from the PIC MCU? How many outputs to the Dot Matrix? Based on that, what kind of an IC is a 74LS138?

  ____

B. Using the Code Editor

1. Push the Program/Test switch back to go back into program mode.



2. Click "View Code".
3. The PICpad editor will appear with the C code of the SimpleDisplay experiment. It is advised to read section 2.3 "C Language Structure" beforehand to get acquainted with coding.
4. As you will notice, the code is long and contains some parts that can be difficult to understand. Make sure to read all of the comments to help you understand the purpose of each line.
5. Notice the `//############## includes, defines    ##############` block

   It contains include files and header files needed by the program.

```
#include <pic.h>
#include <string.h>

#ifndef _XTAL_FREQ          // This code is needed for delay routines
 #define _XTAL_FREQ 4000000  // Unless defined assume 4MHz frequency
```

```
23    #endif
24
25    //Dot Matrix Includes
26    #include "C:\PICSoft 4\charset_2.h"
27    #include "C:\PICSoft 4\switch_char.h"
```

6. Line 18 is necessary to obtain compile information on the PIC
7. Line 19 is used for string operations such as strlen() which gets the length of a string
8. Line 21-23 is used for defining the operation speed as 4MHz default, which is used for delay routines, such as __delay_ms()
9. Line 26-27 includes the character set files for the character representation of the alphabet, numerals, and symbols for the 8x8 Dot Matrix display
10. Examine the `//########### Global Variables and Arrays ##########` block.

    The one you can freely edit here is the string declaration. Supply the string of your choice.

    ```
    char message[] = "message";
    ```

11. Study the `// ############# program loop ##############` block.

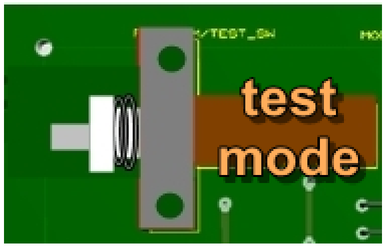
```
while(1 == 1)                  //  Loop Forever
    {
     for (ctr = 0; ctr <= length ; ctr++)     // process each character one by one up to the end of message
      {
        for  (Dlay  =  0;  Dlay  <  48;  Dlay++)       //  Delay  for  each character before next
         {
          for  (i  =  0;  i  <  8;  i++)           //  Loop  each  line  of  the charset
             {
                 j = char_set[(((translate(message,ctr))*8)+i)];
                 PORTB  =  j  &  0b11111111;      //  Load  pattern  to  the anodes
                 PORTA= i;                     // Enable current line

                 __delay_ms(2);                  //  Delay before next line
             }  //  rof i

        } // rof ylaD

```

```
101         }
102
103     __delay_ms(750);    // pause after displaying message then repeat again
104
105     }   // elihw
```

11. The code consists of several For loops inside one another. The outermost loop (Line 87) processes each character.
12. The middle loop (Line 89) provides a delay before next character. You can modify this delay freely to make the transition faster or slower.
13. The innermost loop (Line 91) processes 0 to 7 to take each line of a character set.
14. Line 94 loads the character set data to PORTB and into the DotMatrix anodes
15. Line 95 enables the current line to light it up.
16. Line 103 provides the pause before the message repeats. You can modify this freely.
17. This time, do not change the PIC type and PIC name so we do not have to change the connections.
18. To test the codes, click the Save button.
19. Click Compile button. If there are no errors, you would be prompted to burn the program. In case there are errors, compiling would not finish and the errors will be displayed.
20. Click Burn then click ‘Proceed with Burning’.

21. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Success prompt would appear. Close the window.

12. Push the Program/Test switch at the USB PIC Programmer to test the program. Observe the scrolling message in the Dot Matrix Module.

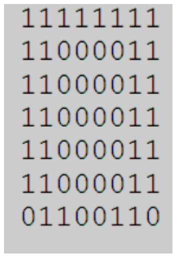


22. To modify the program, push it back to Program mode and follow the steps above in editing, saving, compiling, burning and testing back.

EXERCISES:

4. Create your 8x8 character representation of the following characters, the first one is done for you

    - ‘C’



```
11111111
11000011
11000011
11000011
11000011
11000011
01100110
```

    - ‘5’
    - ‘#’
    - ‘i’
    - ‘x’
    - ‘M’

5. Try the Horizontal Scroll and Vertical Scroll experiments in the GUI under the Dot Matrix tab. Don’t forget to customize it by changing the text and testing it at different speeds. You can also study and edit the program shown in the Visual Emulator by clicking the “View Code”. What are the differences in the code of Horizontal Scroll and Vertical Scroll experiments?

OBSERVATION:

CONCLUSION:

Interfacing with the LCD Module

OBJECTIVES:

- To be able to use the LCD Module of the PC-Based Microcontroller Programmer and Training Kit
- To be able to display a message in a 16x2 LCD
- To be able to interface a 18-pin PIC MCU with a LCD Display

MATERIALS:

- PC-Based Microcontroller Programmer and Training Kit
- PC with Windows XP running PICSoft
- PIC16F88 18-Pin PIC MCU

INTRODUCTION:

LCD Module and PIC MCU Ports

The LCD module features a 16x2 characters LCD with backlight. It is used in 4-bit mode to reduce the interface pins. This module contains 2 headers: the Module Supply header and one I/O header.

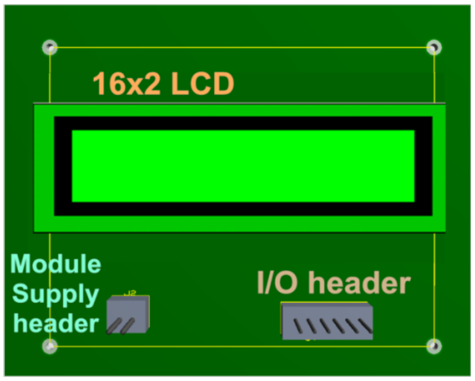


Figure 1 **LCD Module**

In this experiment, PORTB will be used. Only 6 pins is required (RB0 to RB5).

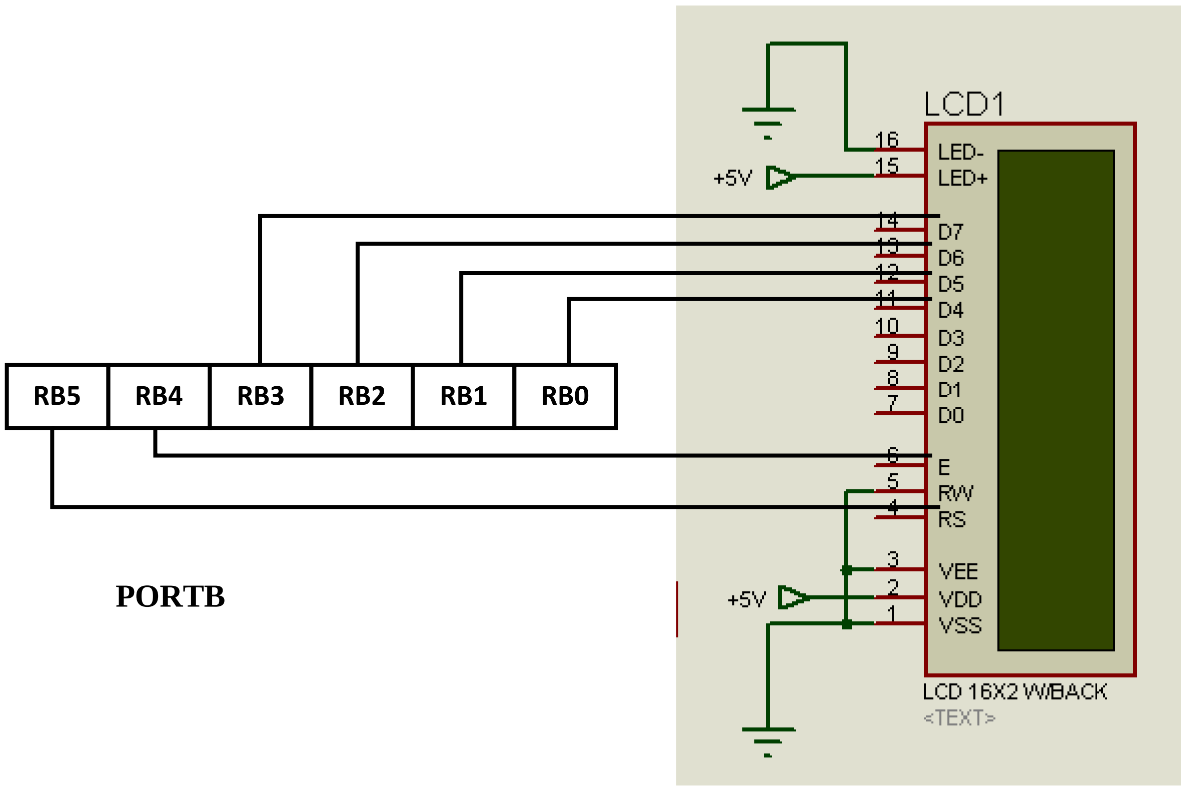


Figure 2 **LCD Module connections to PORTB**

- Vdd and Vss (pins 1 and 2) are the power pins and connected to the 5V and GND. RW (Pin 5) (Read Write) pin is connected to GND to continuously send commands and characters to the LCD. Vee (Pin 3) is also shorted to GND to keep the highest display contrast.
- LED- and LED+ (Pins 15 and 16) are the supply for the Backlight. This should also be connected to 5V and GND.
- D4 to D7 (Pins 11 to 14) are connected to RB0 to 3. Data is sent from the PIC MCU to the LCD in 4 bits. This has the advantage of using fewer I/O lines.
- Enable (E) (pin 6) is used to initiate the transfer of commands or data between the LCD module and the microcontroller.
- Register Select (RS) (pin 4) controls the data sent to the LCD. When this pin is LOW, data transferred to the display is treated as commands. When RS is HIGH, character data can be transferred to and from the module.

Notes:

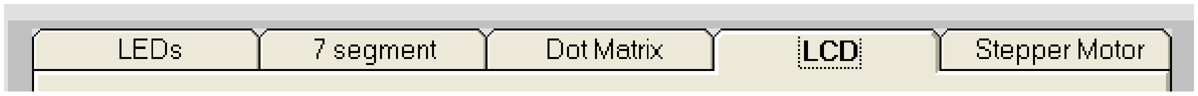
- Some reserved characters in programming cannot be used for the LCD display, such as & (ampersand), “ (double quotes), and \ (backslash).
- When the LCD is displaying incorrectly, use the green RESET button to refresh the program and the display itself.

PROCEDURES:

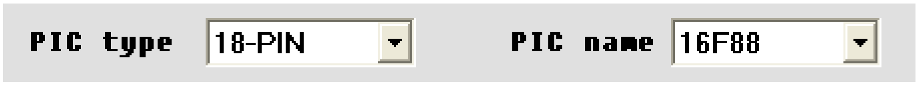
Before starting with the procedures, make sure you have already prepared the kit and installed the Device Driver and the Software. (See Chapter 2 for help).

A. Using the GUI and Virtual Emulator

1. Open PICSoft on the computer.
2. Select the LCD in the devices tab.

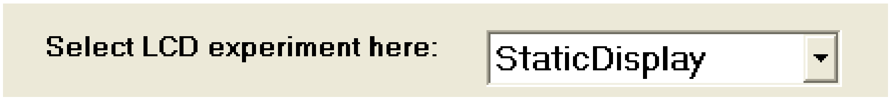


3. Select 18-PIN in the PIC type listbox and 16F88 in the PIC name listbox.

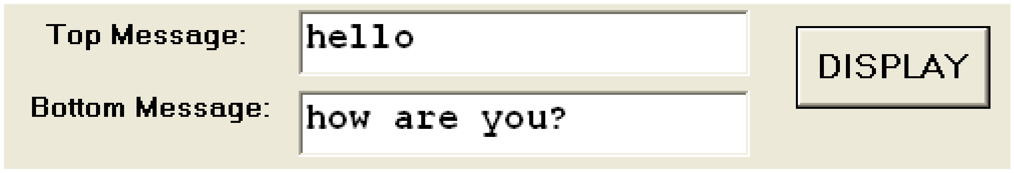

PIC type 18-PIN

PIC name 16F88

4. There are 3 available experiments in the LCD experiment listbox. For this time, click the "Static Display".

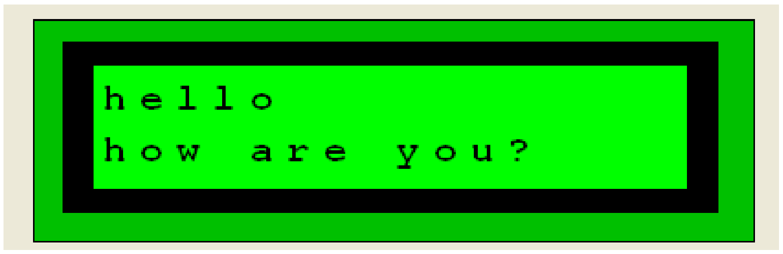

Select LCD experiment here: StaticDisplay

5. Type a string for the Top Message and another for the Bottom Message. Click 'Display' to preview how it would look in actual.

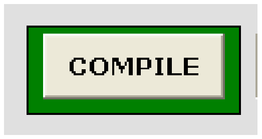

Top Message: hello

Bottom Message: how are you?

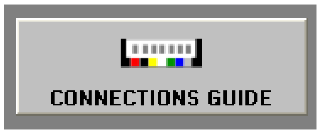
6. Notice the graphical animation of the LCD display. This is the Virtual Emulator for the LCD.



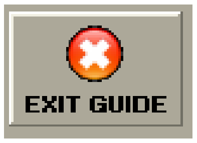
7. Click ‘Compile’.



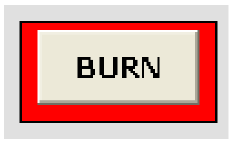
8. Click ‘OK’ when the Successful Compilation prompt appears.
9. At this point the device connections must be done before burning. Click the “Connections Guide” at the bottom- right side.



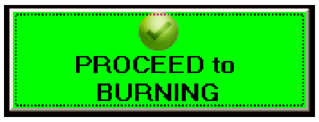
10. Follow the instructions and navigate through them by using the Prev and Next buttons. This guide will show you how to properly connect the headers for programming and testing of the compiled program.
11. Click Exit at the end of the guide (Connect USB page).



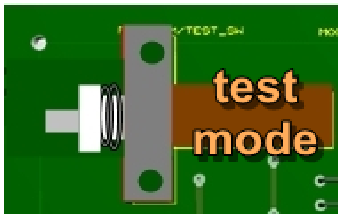
12. At this moment, click Burn.



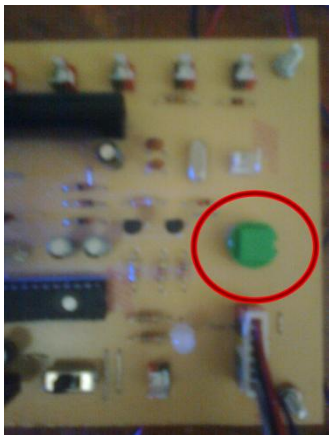
13. A confirmation window would appear. You can disregard this and click “Proceed to Burning” if you have read the connections guide earlier.



14. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Complete prompt would appear. Close the window.
15. Push the Program/Test switch at the USB PIC Programmer to test the program.



16. In some cases where the LCD displays incorrectly or displays nothing, press the Green Reset button in the USB PIC Programmer. This button can also be used to reset the count.



17. Observe the actual LCD Display. Don't close PICSoft.

18. Click "Schematic". You should now see the schematic representation of the connections done in the kit. This could also help you in programming.

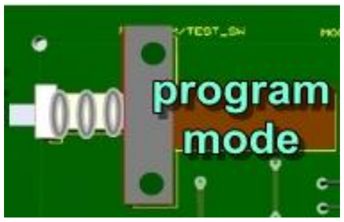
Question:

- In which PORT is the LCD connected?

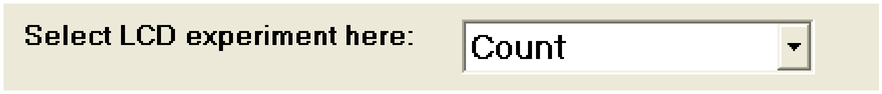
  ____

B. Using the Code Editor

1. Push the Program/Test switch back to go back into program mode.



2. Go back to the main window by clicking Exit at the Schematic Window (if it is still open).
3. Choose the other experiment in the LCD experiment listbox, named 'Count'.


Select LCD experiment here: Count

4. Click "View Code".

5. The PICpad editor will appear with the C code of the CountUp experiment. It is advised to read section 2.3 "C Language Structure" beforehand to get acquainted with coding.
6. As you will notice, the code contains several parts separated as blocks. Make sure to read all of the comments to help you understand the purpose of each block and line.
7. Notice the `//############## includes, defines    ##############` block

It contains include files and header files needed by the program.

```
17    //LCD Defines and Includes
18    #define LCD_D4 RB0
19    #define LCD_D5 RB1
20    #define LCD_D6 RB2
21    #define LCD_D7 RB3
22    #define LCD_EN RB4
23    #define LCD_RS RB5
24    #define LCD_DATA  PORTB
25    #define LCD_STROBE()    ((LCD_EN = 1),(LCD_EN=0))
26
27    #include "C:\PICSoft 4.8\lcd3.c"
```

8. Line 18-21 creates an alias as LCD data pins LCD_D4:7 for RB0 to RB3. Aliases simplify programming.
9. Line 22 creates an alias ‘LCD_EN’ for RB4.
10. Line 23 also creates an alias ‘LCD_RS’ for RB5.
11. Line 24 creates an alias ‘LCD_DATA’ for the entire PORTB.
12. Line 25 creates a command alias ‘LCD_STROBE’ for switching LCD_EN on and off.
13. Line 27 includes the lcd.c file that contains several functions needed to display text in the LCD. You can examine it by opening the file path in Notepad. Note that the generated path may not be the same in your computer. It depends on where PICsoft is installed.
14. Notice the `//             ############# program loop ##############` block

```
while(1 == 1)                 //  Loop Forever

   {
        for (ctr=0 ;ctr<= 9999;ctr++)
        {
        lcd_clear();                           // clear to refresh
        lcd_putsxy(0,0,"Count:");              // print top message
        lcd_writeintxy(0,1,ctr,4);               // print digit
        __delay_ms(1000);                  // delay before next
        }

    }   // elihw
```

15. Line 79 ensures that the code restarts counting when it is finished.
16. Line 82 is the for loop that sets the bounds of counting. You can edit this code to change the range or make it count backwards.
17. Line 85 uses the `lcd_putsxy()` command. It's syntax is

    ```
    lcd_putsxy(x position, y position, string to be printed)
    ```

    It enables a string to be printed in the exact XY position of the LCD.

    `lcd_putsxy(0,0,"Count:");` prints the "Count:" string in the first row (X=0), first column (Y=0)

18. Line 86 uses the `lcd_writeintxy()` command. It's syntax is

    ```
    lcd_writeintxy (x position, y position, string to be printed, length)
    ```

    It enables an integer to be printed in the exact XY position of the LCD.

    `lcd_writeintxy(0,1,ctr,4);` prints the ctr integer starting in the first row (X=0), second column (Y=1), reserve 4 spaces for the integer

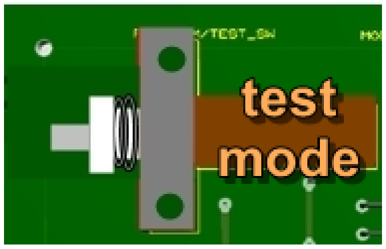
19. Feel free to edit the code as desired. You can adjust the count start and end. You can also adjust the delay by replacing the number inside the __delay_ms( ) function.
20. You can also modify the text displayed by the LCD by editing the string parameter ("Count:") of the lcd_putsxy() command (line 85). Make sure to not exceed 16 characters or the text will be truncated.
21. This time, do not change the PIC type and PIC name so we do not have to change the connections.
23. To test the codes, click the Save button.

24. Click Compile button. If there are no errors, you would be prompted to burn the program. In case there are errors, compiling would not finish and the errors will be displayed.

25. Click Burn then click ‘Proceed with Burning’.

26. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Complete prompt would appear. Close the window.

22. Push the Program/Test switch at the USB PIC Programmer to test the program. Observe the counting display in the LCD Module.



19. In some cases where the LCD displays incorrectly or displays nothing, press the Green Reset button in the USB PIC Programmer. This button can also be used to reset the count.

27. To modify the program, push it back to Program mode and follow the steps above in editing, saving, compiling, burning and testing back.

EXERCISES:

1. Try the 3rd Experiment for the LCD Module named ‘Scroll’. Observe the Left and Right Scroll in the Visual Emulator and in the actual device. View the code and note the differences between the left and right scroll algorithm.

2. View and edit the code of the StaticDisplay experiment. Manually replace the display string from the code editor.

3. View and edit the code of the Count experiment. Place various combinations of the `lcd_putsxy()` and `lcd_writeintxy()` commands of your choice.

4. Display another counting variable at another position.

   Declare another variable first at the variables section

```
//########### Global Variables and Arrays ##########

int ctr,ctr2;
```

   Then, in the program loop section, print the other variable after the first variable. Take this as an example.

```
lcd_writeintxy(0,1,ctr,4);                // print number in first row,
                                         2nd       // column

lcd_writeintxy(6,1,ctr2,4);                   // print 2nd number in 6th
row, 2nd
                                           // column
```

OBSERVATION:

CONCLUSION:

Interfacing with the Motor Module

OBJECTIVES:

- To be able to use the Motor Module of the PC-Based Microcontroller Programmer and Training Kit
- To be able to control a stepper motor to rotate an exact amount degrees and with the desired speed

  To be able to interface a 18-pin PIC MCU with a Stepper Motor

MATERIALS:

- PC-Based Microcontroller Programmer and Training Kit
- PC with Windows XP running PICSoft
- PIC16F88 18-Pin PIC MCU

INTRODUCTION:

Motor Module

The motor module employs a bipolar stepper motor. It can rotate for the exact number of steps based on the user program. A L293D driver chip was used to control the output signals from the MCU port to the motor coils. Another chip, the 74LS04 is used to invert the power signals to the required logic of the L293D chip. This module contains 2 headers: the Module Supply header and one I/O header.

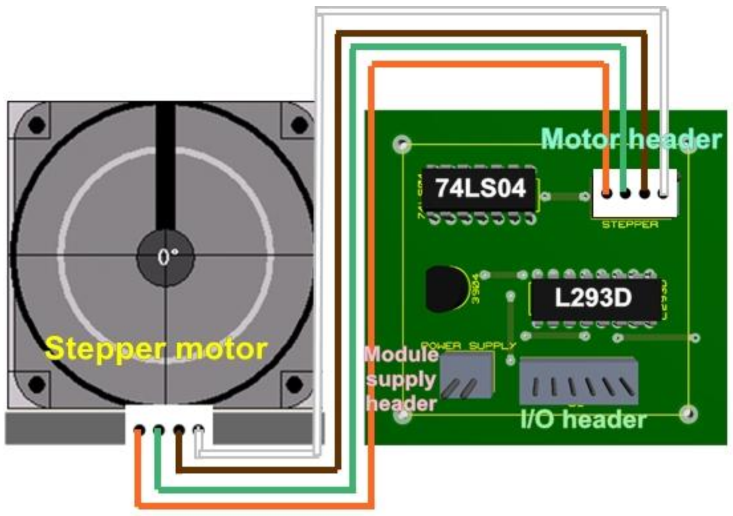


Figure 1 **Motor Module**

Stepper Motor

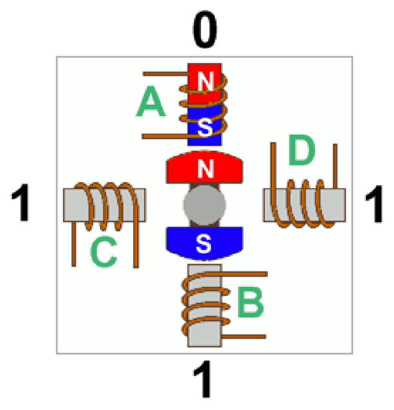
A stepper motors rotate in steps where each step is a fraction of a full circle. This fraction depends mostly from the mechanical parts of the motor, and from the driving method. They are driven usually with pulses. Each pulse is translated into a degree of rotation. The stepper motor included in this kit is a 1.8° stepper motor, which will revolve its shaft 1.8° on every pulse that arrives.

The stepper motors consists of 4 coils arranged perpendicularly to each other. These four coils surround a magnetized shaft, which will be either attracted or repelled when the coils are energized. To turn the stepper motor, the coils are energized in a pattern that will cause it to turn in one direction or another. The driving method used is called "Wave Drive" mode, where the coils are activated in a cyclic order, one by one. The sequence of steps, along with each corresponding 4-bit binary code is shown at the following page.

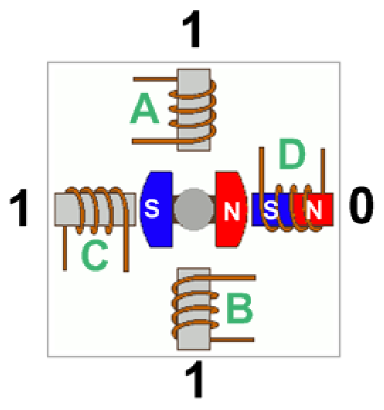
The coils are positioned to in a way that its pole closer to the shaft magnet will be South when given a logic 0, thus attracting the North Pole of the shaft magnet. For instance, in the first sequence, the coil A is given the logic 0 to match its South with the shaft's north. The logic 1 in coils B, C, and D are nulled.

The four codes are sent continuously to the coils to make the motor rotate.

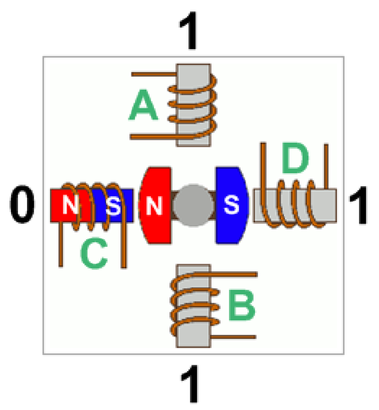
```
             ABCD
Sequence 1:  0111
```



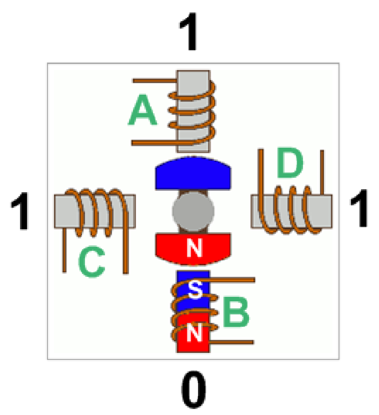
```
             ABCD
Sequence 2:  1110
```



```
             ABCD
Sequence 4:  1101
```



```
             ABCD
Sequence 3:  1011
```



In this experiment, PORTB will be used. Only 6 pins is required (RB0 to RB5).

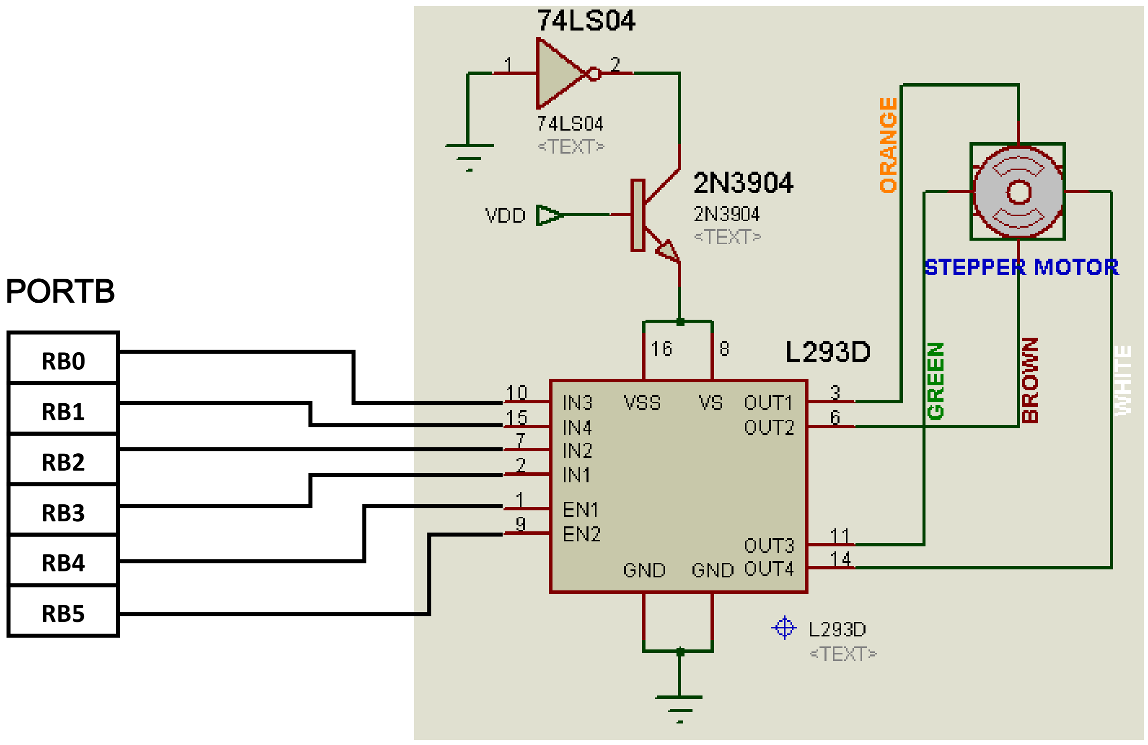


Figure 2 **Connections of PORTB, L293D and Stepper Motor**

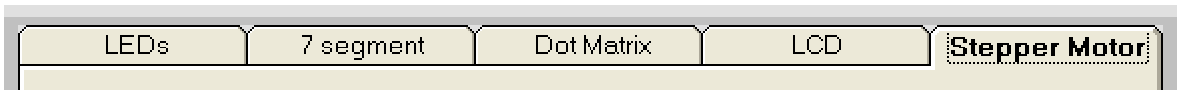
- PORTB is connected to the motor driver chip L293D. RB0 to 3 are connected to the 4 inputs and the outputs are connected to the Stepper Motor.
- The L293D chip simplifies programming for the user. The program on the PIC MCU has to only send the binary signals continuously through a loop and the L293D chip will handle the pulses of current- switching to energize the four coils accordingly.
- RB4 and 5 are connected to the coil enable pins of the chip. It provides control to the Stepper Motor coils.
- A 74LS04 inverter and 2N3904 NPN transistor is used to provide a switched supply to the Vss and Vs pins.

PROCEDURES:

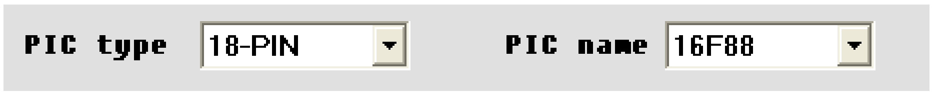
Before starting with the procedures, make sure you have already prepared the kit and installed the Device Driver and the Software. (See Chapter 2 for help).

A. Using the GUI and Virtual Emulator

1. Open PICSoft on the computer.
2. Select the Motor in the devices tab.

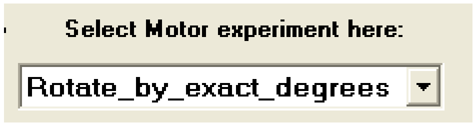


3. Select 18-PIN in the PIC type listbox and 16F88 in the PIC name listbox.

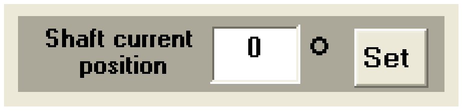

PIC type 18-PIN

PIC name 16F88

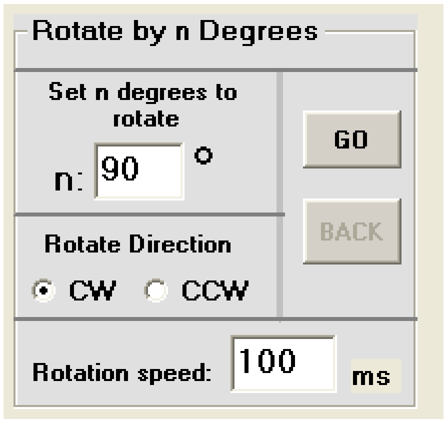
4. There are 3 available experiments in the Motor experiment listbox. For this time, click the “Rotate_by_exact_degrees”.


Select Motor experiment here: Rotate_by_exact_degrees

5. For the actual device to be consistent with the emulation and vice-versa, the actual motor shaft’s current position must be set.


Shaft current position 0 °

6. You can then supply the number of degrees to rotate, the direction (CW – clockwise or CCW – counter clockwise), and the rotation speed in millisecond.



Set n degrees to rotate

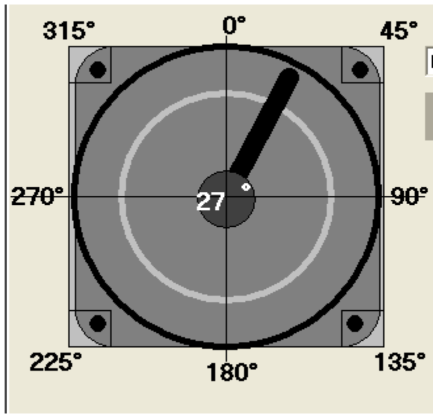
n: 90 °

Rotate Direction

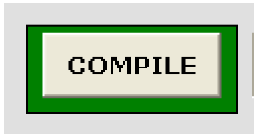
[✓ CW] CCW

Rotation speed: 100 ms

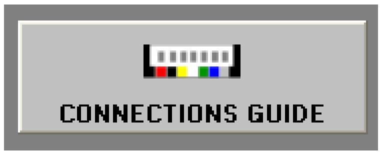
7. Click ‘Go’ to emulate.
8. Notice the graphical animation of the moving motor shaft. This is the Virtual Emulator for the Stepper Motor.



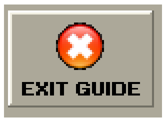
9. Click ‘Compile’.



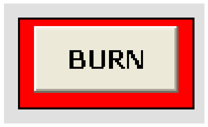
10. Click ‘OK’ when the Successful Compilation prompt appears.
11. At this point the device connections must be done before burning. Click the “Connections Guide” at the bottom- right side.



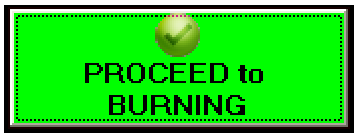
12. Follow the instructions and navigate through them by using the Prev and Next buttons. This guide will show you how to properly connect the headers for programming and testing of the compiled program.
13. Click Exit at the end of the guide (Connect USB page).



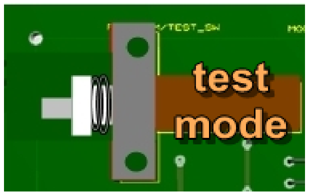
14. At this moment, click Burn.



15. A confirmation window would appear. You can disregard this and click "Proceed to Burning" if you have read the connections guide earlier.



16. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Complete prompt would appear. Close the window.
17. Push the Program/Test switch at the USB PIC Programmer to test the program.



18. Observe the actual motor if it is rotating appropriately similar to the emulation. Don't close PICSoft.

19. Click “Schematic”. You should now see the schematic representation of the connections done in the kit. This could also help you in programming.

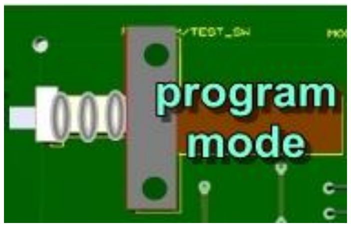
Question:

- In which PORT is the stepper motor connected?

  ____

B. Using the Code Editor

1. Push the Program/Test switch back to go back into program mode.



2. Go back to the main window by clicking “Exit Schem” at the Schematic Window (if it is still open).

3. Click “View Code”.

4. The PICpad editor will appear with the C code of the BlinkLED experiment. It is advised to read section 2.3 “C Language Structure” beforehand to get acquainted with coding.
5. As you will notice, the code contains several parts separated as blocks. Make sure to read all of the comments to help you understand the purpose of each block and line.
6. Notice the `//########### Global Variables and Arrays ##########` block

```
44    unsigned int i = 0, j;
45    const char StepperTable[] = {0b00110111, 0b00111110,
46                                 0b00111011, 0b00111101};
```

7. Line 44 initializes the i variable to 0.
8. Line 45 declares the array containing the four binary values that control the Stepper Motor coils. Focus on the 4 least significant bits of each pattern : 0111, 1110, 1011, and 1101. These are the values of the 4 sequences in the discussion earlier. These values are continuously loaded to the stepper motor coils through the L293D driver. Refer to the discussion for more details.
9. Notice the `//         ############# program loop ##############` block

```
66    for (j=0; j <50; j++)
67
68     {
69        __delay_ms(100);      //delay for each motor step
70
71        PORTB = StepperTable[i];     // step motor based from array
72
73        i = (i + 1) % 4;                    // next step
74
75     }  //  rof
```

10. Line 66 contains the loop of how many steps the motor is moving. 50 steps * 1.8 degrees /step = 90 degrees.

    `for (j=0; j <50; j++)` indicates that the motor must move 50 steps (variable j increments from 0 to 49) in Clockwise Rotation (j++)

Note: To reverse the rotation to Counter-Clockwise, the for loop has to be reversed, such as

`for (j=50; j >0; j--)` where the motor would move 50 steps (variable decrements from 50 to 1) in Counter-Clockwise Rotation (j--)

11. Line 69 contains the delay of every step
12. Line 71 supplies PORTB with a string value from the array.
13. Line 73 increments the array subscript for the next step.
14. Feel free to edit the code as desired. You can adjust the number of steps here manually. You can reverse the direction by decrementing the array counter .

```
i = (i - 1) % 4;
```

15. You can also adjust the delay by replacing the number inside the __delay_ms( ) function.
16. This time, do not change the PIC type and PIC name so we do not have to change the connections.
28. To test the codes, click the Save button, then the Compile button. If there are no errors, you would be prompted to burn the program. Click Burn then Proceed.
29. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Success prompt would appear. Close the window.
30. Push the Program/Test switch at the USB PIC Programmer to test the program.
31. To modify the program, push it back to Program mode and follow the steps above in editing, saving, compiling, burning and testing back.

EXERCISES:

1. Modify this for loop so that the motor would rotate 25 steps in Counter-Clockwise direction

   50 steps CW: `for (j=0; j <50; j++)`

   25 steps CCW: ____________________

2. Using the formula [no.of steps = no.of degrees / 1.8], create code for the motor to rotate for the following degrees:

   - 180°

     No. of steps:

     Code:

   - 270°

     No. of steps:

     Code:

   - 36°

     No. of steps:

     Code:

3. Try the other experiments in the GUI under the Motor tab. Don't forget to customize it by using the options provided. You can also study and edit the program shown in the Visual Emulator by clicking the "View Code". What are the differences in the "Rotate_by_exact_degrees" and "Go_to_exact_degrees" experiment?

OBSERVATION:

CONCLUSION:

Interfacing with the Temperature Sensor

OBJECTIVES:

- To be able to use the Temperature Sensor of the PC-Based Microcontroller Programmer and Training Kit
- To be able to use the ADC (Analog to Digital Converter) of a PIC MCU
- To make a simple digital thermometer using the Temperature Sensor and a 7-segment Display
- To be able to interface a 18-pin PIC MCU with a Temperature Sensor

MATERIALS:

- PC-Based Microcontroller Programmer and Training Kit
- PC with Windows XP running PICSoft
- PIC16F88 18-Pin PIC MCU

INTRODUCTION:

Temperature Sensor

The sensor used is a LM-35 precision integrated-circuit temperature sensor, whose output voltage is linearly proportional to the Celsius temperature. This 3-pin IC in a transistor package is directly connected to the power supply and to the ADC (Analog to Digital Converter) pin of the PIC MCU port.

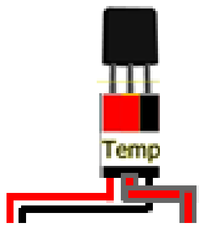


Figure 1 **LM-35 Sensor**

A special pin in PORTA must be used to connect the Temperature Sensor. The other pins of PORTA and entire PORTB will be connected to the 7-segment module.

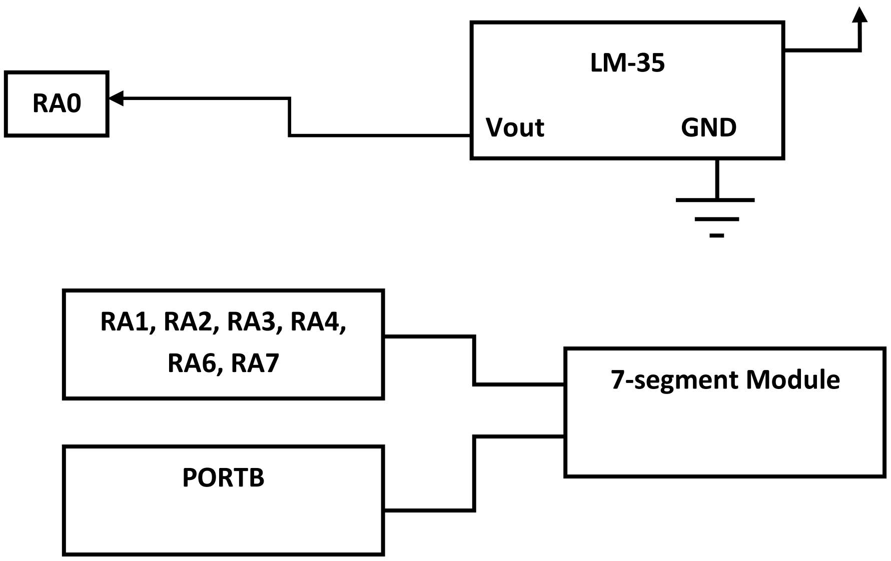


Figure 2 **PORTA and PORTB connections in LM-35 interfacing**

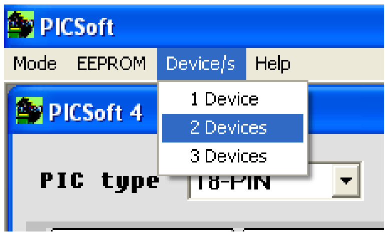
- RA0 is connected to the LM-35 sensor. RA0 has a special feature that can read analog voltages and convert it to a digital form that can be processed by the program.
- RA1, RA2, RA3, RA4, RA6, RA7 and entire PORTB are connected to the 7-segment module. A detailed description of connections required with interfacing with a 7-segment display is available at Laboratory Experiment 2.

PROCEDURES:

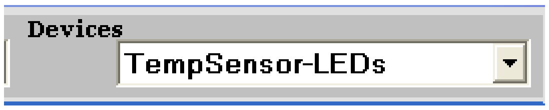
Before starting with the procedures, make sure you have already prepared the kit and installed the Device Driver and the Software. (See Chapter 2 for help).

A. Using the GUI and Virtual Emulator

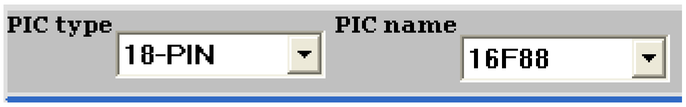
1. Open PICSoft on the computer.
2. Go to Menu Bar > Device/s > 2 Devices



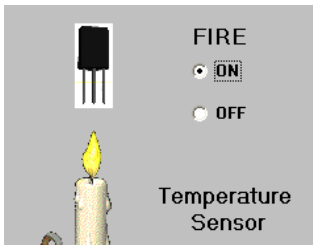
3. The form for Dual Devices would appear.
4. Select the 'TempSensor-7seg' in the Devices tab.



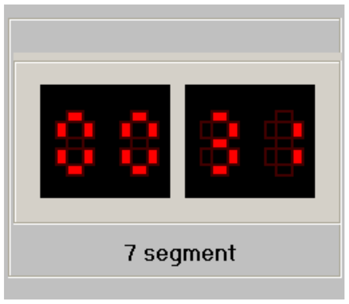
5. Select 18-PIN in the PIC type listbox and 16F88 in the PIC name listbox.



6. The emulation starts with the normal room temperature. Toggle the FIRE on/off to increase the temperature.



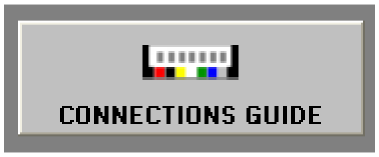
7. Temperature in the 7-segment display emulation increases.



8. Click ‘Compile’.

9. Click ‘OK’ when the Successful Compilation prompt appears.

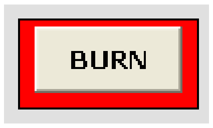
10. At this point the device connections must be done before burning. Click the “Connections Guide” at the bottom- right side.



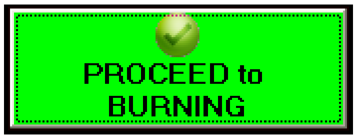
11. Follow the instructions and navigate through them by using the Prev and Next buttons. This guide will show you how to properly connect the headers for programming and testing of the compiled program.

12. Click Exit at the end of the guide (Connect USB page).

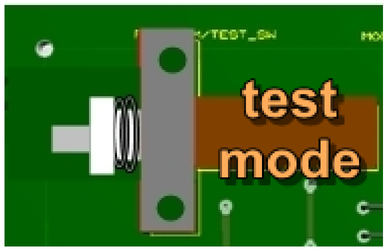
13. At this moment, click Burn.



14. A confirmation window would appear. You can disregard this and click “Proceed to Burning” if you have read the connections guide earlier.



15. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Complete prompt would appear. Close the window.
16. Push the Program/Test switch at the USB PIC Programmer to test the program.



17. Observe the actual 7-segment Display. It may take some seconds for the temperature to stabilize.
18. The LM-35 as default measures ambient temperature. When the program is executed, it should measure the room temperature and display the reading in the 7-segment display.
19. To properly test the Temperature Sensor, you can hold the plastic casing (not the pins) of the LM-35 with two fingers. Notice the temperature in the 7-segment display if it adapts with your body temperature.

20. It is also advised to try hotter or cooler objects with the sensor. You can lit a lighter or candle and put it near the sensor, without burning the plastic case.
21. You can also get an ice cube and let the plastic case of the sensor touch it. Do not submerge the sensor in liquid as it will shorten out the pins and will cause damage in the device.
22. Click "Schematic". You should now see the schematic representation of the connections done in the kit. This could also help you in programming.

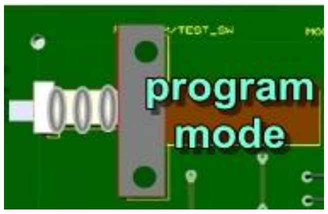
Questions:

- In which pin is the LM-35 connected in the experiment?

  ____

B. Using the Code Editor

1. Push the Program/Test switch back to go back into program mode.



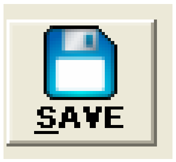
2. Go back to the main window by clicking Exit at the Schematic Window (if it is still open).

3. Click “View Code”.

4. The PICpad editor will appear with the C code of the BlinkLED experiment. It is advised to read section 2.3 “C Language Structure” beforehand to get acquainted with coding.
5. As you will notice, the code contains several parts separated as blocks. Make sure to read all of the comments to help you understand the purpose of each block and line.
6. Notice the `// ############# program loop ##############` block

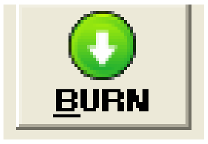
```
183    __delay_ms(1);            //acquisition delay
184    GO_DONE=1;               //Start conversion
185    __delay_ms(1);            //acquisition delay
186    input = ADRESL;         //  Read Value
187
188         t= (int) round(input * 0.48876);     //Convert to Degree
Celcius
```

7. Line 183 and 185 contains the delay for acquisition and conversion of temperature
8. Line 184 starts conversion of Analog Input in pin RA0 to digital format
9. Line 186 reads the converted value and places it in a variable named ‘input’
10. Line 188 converts the results to Degree Celsius
11. Several other blocks of codes exist to initialize the ADC and update the 7-segment display. Details on running a 7-segment display can be found in Laboratory Experiment 2.
12. This time, do not change the PIC type and PIC name so we do not have to change the connections.
13. To test the codes, click the Save button.



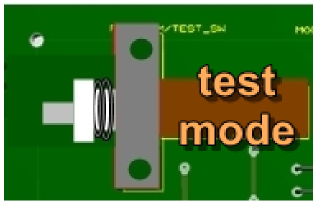
14. Click Compile button. If there are no errors, you would be prompted to burn the program. In case there are errors, compiling would not finish and the errors will be displayed.

15. Click Burn then click 'Proceed with Burning'.



16. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Success prompt would appear. Close the window.

32. Push the Program/Test switch at the USB PIC Programmer to test the program.



33. The LM-35 as default measures ambient temperature. When the program is executed, it will measure the room temperature.

34. To properly test the Temperature Sensor, you can hold the plastic casing (not the pins) of the LM-35 with two fingers. Notice the temperature in the 7-segment display if it adapts with your body temperature.

35. It is also advised to try extreme temperature with the sensor. You can lit a lighter or candle and put it near the sensor, without touching the plastic case.

36. You can also get an ice cube and let the plastic case of the sensor touch it. Do not submerge the sensor in liquid as it will shorten out the pins and will cause damage in the device.

37. To modify the program, push it back to Program mode and follow the steps above in editing, saving, compiling, burning and testing.

EXERCISES:

1. Try the other experiments in the 2 Devices form that involves the Temperature Sensor, such as the 'TempSensor-LEDs' and 'TempSensor-LCD'. You can also study and edit the program shown in the Visual Emulator by clicking the "View Code".

2. Change the unit of the displayed temperature by adding some additional conversion to the temperature variable. It can be done by modifying the line that does the conversion.

```
t= (int) round(input * 0.48876);  // Celsius equivalent

t= ((t*9) /5) + 32;           //convert to degree Fahrenheit
```

or

```
t= t + 273;                   //convert to degree Kelvin
```

OBSERVATION:

CONCLUSION:

Interfacing with the 7-Segment Module and LED Module

OBJECTIVES:

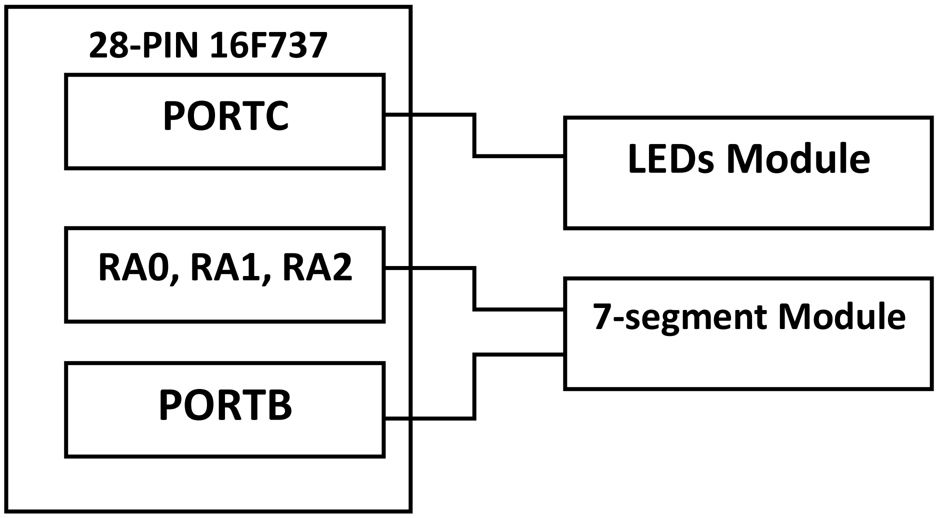
- To be able to use the 7-segment Module and LED Module of the PC-Based Microcontroller Programmer and Training Kit simultaneously
- To be able to create a program that employs number system conversion
- To be able to interface a 28-pin PIC MCU with 7-segment Display and LEDs

MATERIALS:

- PC-Based Microcontroller Programmer and Training Kit
- PC with Windows XP running PICSoft
- PIC16F737 28-Pin PIC MCU

INTRODUCTION:

PIC MCU PORTS



In this experiment involving two modules, the 7-segment Module will be connected to PORTA and PORTB of the 28-pin 16F737 PIC MCU. The LEDs Module will be connected to PORTC.

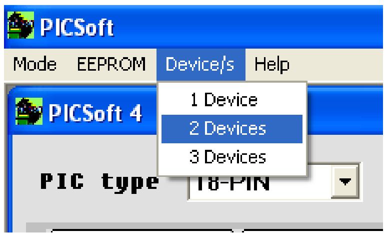
For detailed information regarding interfacing with 7-segment Module or LED module, refer to Laboratory Experiments 1 and 2.

PROCEDURES:

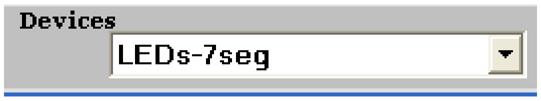
Before starting with the procedures, make sure you have already prepared the kit and installed the Device Driver and the Software. (See Chapter 2 for help).

A. Using the GUI and Virtual Emulator

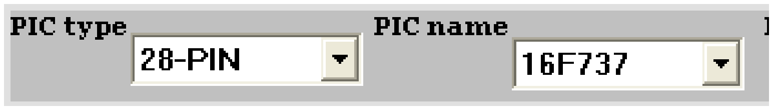
1. Open PICSoft on the computer.
2. Go to Menu Bar > Device/s > 2 Devices



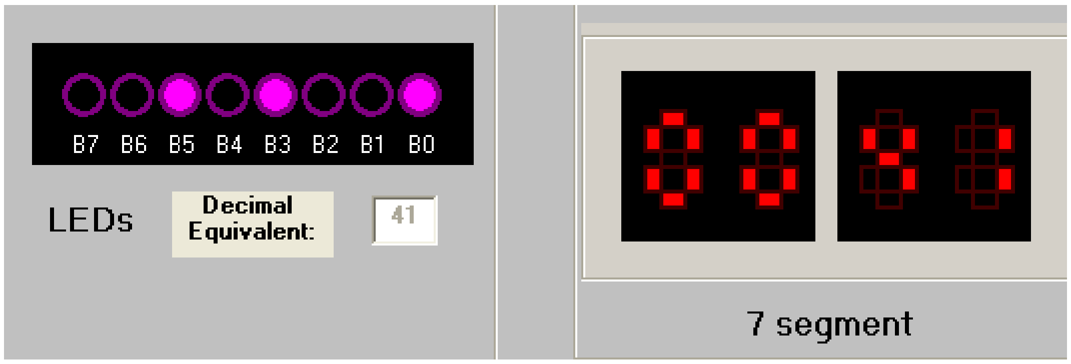
3. The form for Dual Devices would appear.
4. Select the ‘LEDs-7seg’ in the Devices tab.


Devices
LEDs-7seg

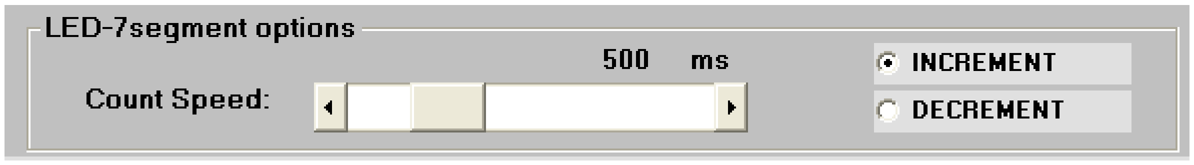
5. Select 28-PIN in the PIC type listbox and 16F737 in the PIC name listbox.


PIC type
28-PIN
PIC name
16F737

6. The virtual emulator runs both the LEDs and the 7-segment simultaneously. The 7-segment displays the counter in decimal while the LEDs display it in binary.



7. In the options, choose either Increment or Decrement. Also change the count speed by moving the slider.


LED-7segment options

Count Speed: 500 ms

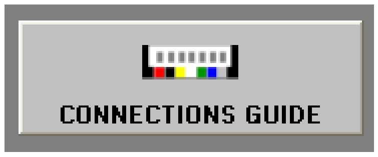
INCREMENT [✓]

DECREMENT

8. Click ‘Compile’.

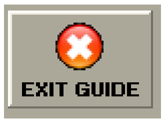
9. Click ‘OK’ when the Successful Compilation prompt appears.

10. At this point the device connections must be done before burning. Click the “Connections Guide” at the bottom- right side.

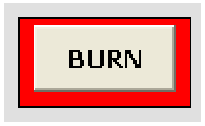


11. Follow the instructions and navigate through them by using the Prev and Next buttons. This guide will show you how to properly connect the headers for programming and testing of the compiled program.

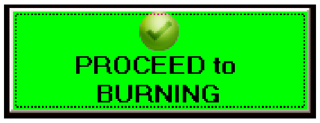
12. Click Exit at the end of the guide (Connect USB page).



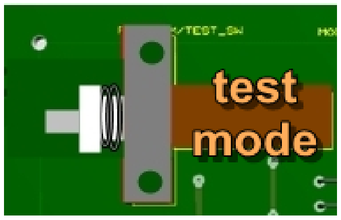
13. At this moment, click Burn.



14. A confirmation window would appear. You can disregard this and click “Proceed to Burning” if you have read the connections guide earlier.

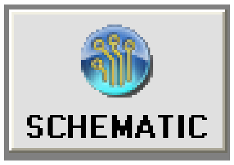


15. Click ‘Yes” at the Burning Confirmation message box.
16. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Complete prompt would appear. Close the window.
17. Push the Program/Test switch at the USB PIC Programmer to test the program.



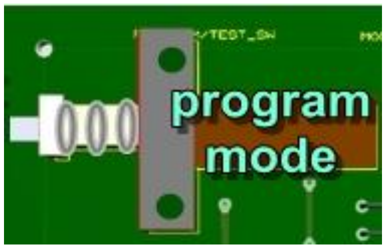
18. Observe the actual 7-segment and LEDs modules. See if the modules are running synchronously with each other. Also check the binary-decimal conversion if it is correct. Don’t close PICSoft.

19. Click “Schematic”. You should now see the schematic representation of the connections done in the kit. This could also help you in programming.

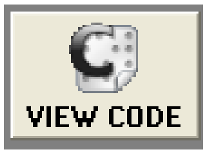


B. Using the Code Editor

1. Push the Program/Test switch back to go back into program mode.



2. Go back to the main window by clicking Exit at the Schematic Window (if it is still open).
3. Click “View Code”.



4. The PICpad editor will appear with the C code of the CountUp experiment. It is advised to read section 2.3 “C Language Structure” beforehand to get acquainted with coding.
5. As you will notice, the code is long and contains some parts that can be difficult to understand. Make sure to read all of the comments to help you understand the purpose of each line.

6. Notice the `//  ############# program loop ##############` block

```
178   while(1 == 1)                 //  Loop Forever
179       {
180          PORTC = ctr;   //load PORTC to light the LEDs up with the
binary

                              . . .

225         d1 = LEDChar[dig1] ^ 0b11111111 ;   // display Digit 1
226         d2 = LEDChar[dig2] ^ 0b11111111 ;   // display Digit 2
227         d3 = LEDChar[dig3] ^ 0b11111111 ;   // display Digit 3
228         d4 = LEDChar[dig4] ^ 0b11111111 ;   // display Digit 4

                              . . .
231        __delay_ms(222);        //delay before next count
232            }
233
234       } // elihw
```

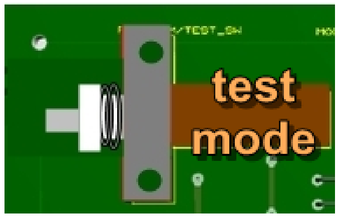
7. Line 180 loads the binary value to the LEDs through PORTC.
8. Lines 225-228 handle displaying of the decimal number to the 7-segment display. Each of the 4 variables (d1,d2,d3,d4) gets the pattern from the LEDChar[ ] array and XOR it with 0b11111111 to negate each bit. This is necessary because the 7-segment used is common-anode type, where the patterns are loaded in the cathodes.
9. `__delay_ms(500);` at line 231 creates the pause of 222 milliseconds before incrementing
10. Feel free to edit the code as desired. You can adjust the count start and end. You can also adjust the delay by replacing the number inside the __delay_ms( ) function.
11. This time, we will not change the PIC type and PIC name so we do not have to change the connections.
12. To test the codes, click the Save button.
13. Click Compile button. If there are no errors, you would be prompted to burn the program. In case there are errors, compiling

would not finish and the errors will be displayed.

14. Click Burn then click ‘Proceed with Burning’.

15. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Success prompt would appear. Close the window.

16. Push the Program/Test switch at the USB PIC Programmer to test the program. Observe the actual 7-segment and LEDs modules if the program is working appropriately.



17. To modify the program again, push it back to Program mode and follow the steps above in editing, saving, compiling, burning and testing back.

EXERCISES:

1. Under the LEDs_7seg device experiment, try Decrement option. Do not forget to customize it by using the options provided. You can also study and edit the program by clicking the “View Code”. What are the differences between the code of increment and decrement experiments?

OBSERVATION:

CONCLUSION:

Interfacing with the Motor and LCD Module

OBJECTIVES:

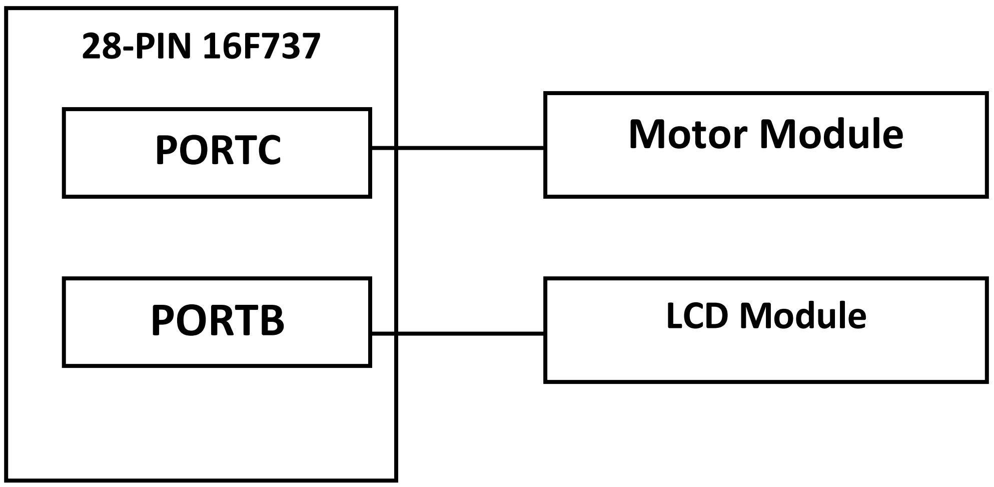
- To be able to use the Motor Module and LCD Module of the PC-Based Microcontroller Programmer and Training Kit simultaneously
- To be able to control a Stepper Motor and display its degrees position in the LCD Display
- To be able to interface a 28-pin PIC MCU with Stepper Motor and LCD

MATERIALS:

- PC-Based Microcontroller Programmer and Training Kit
- PC with Windows XP running PICSoft
- PIC16F737 28-Pin PIC MCU

INTRODUCTION:

PIC MCU PORTS



In this experiment involving two modules, the LCD Module will be connected to PORTB, and the Motor Module will be connected to PORTC.

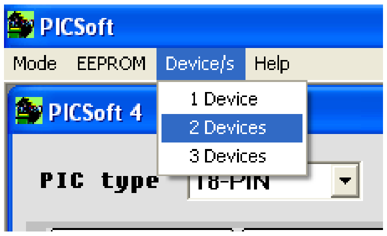
For detailed information regarding interfacing with Motor or LCD module, refer to Laboratory Experiments 4 and 5.

PROCEDURES:

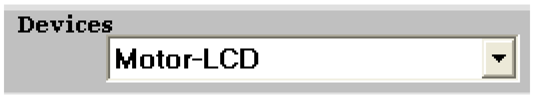
Before starting with the procedures, make sure you have already prepared the kit and installed the Device Driver and the Software. (See Chapter 2 for help).

A. Using the GUI and Virtual Emulator

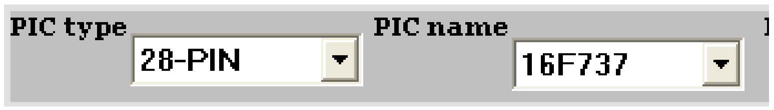
1. Open PICSoft on the computer.
2. Go to Menu Bar > Device/s > 2 Devices



3. The form for Dual Devices would appear.
4. Select the 'LEDs-7seg' in the Devices tab.

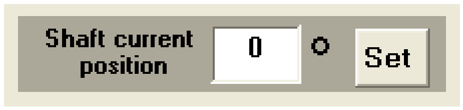


5. Select 28-PIN in the PIC type listbox and 16F737 in the PIC name listbox.

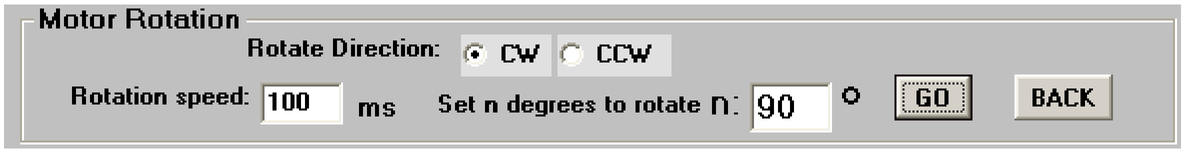


PIC type: 28-PIN
PIC name: 16F737

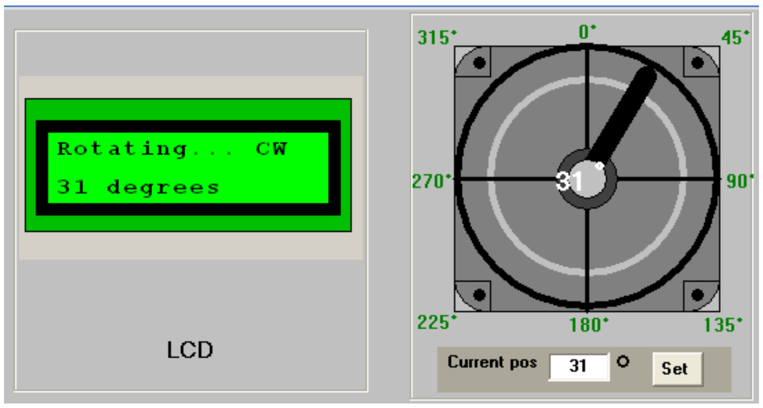
6. For the actual device to be consistent with the emulation and vice-versa, the motor shaft actual current position must be set.



7. You can then supply the number of degrees to rotate, the direction (CW or CCW) and the rotation speed in millisecond.


Motor Rotation
Rotate Direction: [✓ CW] CCW
Rotation speed: 100 ms
Set n degrees to rotate n: 90 °

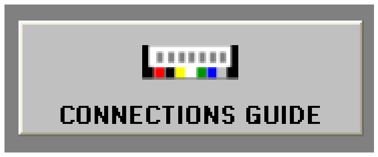
8. Click 'Go' to emulate.
9. The virtual emulator runs both the LCDs and the Stepper Motor simultaneously. The Stepper Motor will rotate for the specified degrees and the current position of the shaft will be displayed in the LCD.


Rotating... CW
31 degrees
Current pos 31 °

10. Click 'Compile'.

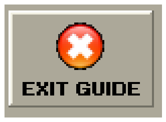
11. Click 'OK' when the Successful Compilation prompt appears.

12. At this point the device connections must be done before burning. Click the "Connections Guide" at the bottom- right side.

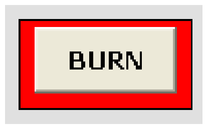


13. Follow the instructions and navigate through them by using the Prev and Next buttons. This guide will show you how to properly connect the headers for programming and testing of the compiled program.

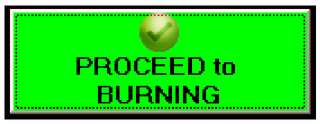
14. Click Exit at the end of the guide (Connect USB page).



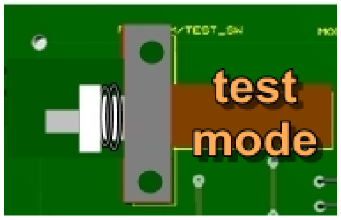
15. At this moment, click Burn.



16. A confirmation window would appear. You can disregard this and click "Proceed to Burning" if you have read the connections guide earlier.



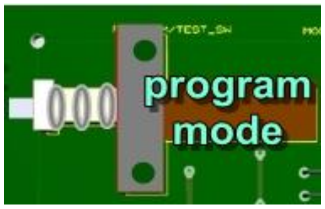
17. Click 'Yes" at the Burning Confirmation message box.
18. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Complete prompt would appear. Close the window.
19. Push the Program/Test switch at the USB PIC Programmer to test the program.



20. Observe the actual Motor and LCD modules. See if the modules are running synchronously with each other. Also check the accuracy of the LCD display with the position and direction of the motor shaft. Don't close PICSoft.
21. Click "Schematic". You should now see the schematic representation of the connections done in the kit. This could also help you in programming.

B. Using the Code Editor

1. Push the Program/Test switch back to go back into program mode.



2. Go back to the main window by clicking Exit at the Schematic Window (if it is still open).
3. Click "View Code".

4. The PICpad editor will appear with the C code of the CountUp experiment. It is advised to read section 2.3 “C Language Structure” beforehand to get acquainted with coding.
5. As you will notice, the code is long and contains some parts that can be difficult to understand. Make sure to read all of the comments to help you understand the purpose of each line.

6. Notice the `//  ############# program loop ##############` block

```
111   for (j=0; j <50; j++)
112
113       {
114         __delay_ms(100);      //delay for each motor step
115
116         PORTC = StepperTable[i];    // step motor based from the array
117
118         if (curDeg >= 360)
119           {
120               curDeg = 0;          // reset to 0 if reached 360 deg.
121           }
122
123
124         i = (i + 1) % 4;                  // next step
125
126         curDeg = curDeg + 1.8;            // increment degree
127
128         ctr = (int) curDeg;               // convert degree to int
129
130         LCD_writeintxy(0,1,ctr,4);  //Print the degrees in 1st column,
131                                     //      2nd row
132        }  //  rof
```

7. Line 111 contains the loop of how many steps the motor is moving. 50 steps * 1.8 degrees /step = 90 degrees.
8. Line 114 contains the 100 ms delay of every step
9. Line 116 supplies PORTC with a string value from the array. The logic signals will then be sent to the motor coils through the L293D IC driver.
10. Line 124 increments the array subscript for the next step.

11. Line 126 increments the degree counter ‘curDeg’ with 1.8° for every step. Line 128 converts ‘curDeg’ to an integer value that can be displayed by the LCD.
12. Line 130 displays the degree counter integer in position X:0 ,Y:1 (1st column, 2nd row) with a field length of 4 (four reserved character spaces)
13. Feel free to edit the code as desired. You can adjust the number of steps here manually. You can reverse the direction by decrementing the array counter .

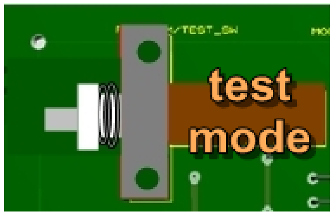
```
i = (i - 1) % 4;
```

14. You can also adjust the delay by replacing the number inside the __delay_ms( ) function.
15. This time, we will not change the PIC type and PIC name so we do not have to change the connections.
16. To test the codes, click the Save button.

17. Click Compile button. If there are no errors, you would be prompted to burn the program. In case there are errors, compiling would not finish and the errors will be displayed.

18. Click Burn then click ‘Proceed with Burning’.

19. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Success prompt would appear. Close the window.
20. Push the Program/Test switch at the USB PIC Programmer to test the program. Observe the actual Motor and LCDs modules if the program is working appropriately.



21. To modify the program again, push it back to Program mode and follow the steps above in editing, saving, compiling, burning and testing back.

EXERCISES:

1. Try the Motor_7seg device experiment under the Devices option. Do not forget to customize it by using the options provided. You can also study and edit the program by clicking the "View Code". What are the differences between the code of Motor_LCD and Motor_7seg experiments?

OBSERVATION:

CONCLUSION:

Pushbutton-controlled Stepper Motor with LCD and 7-segment Display

OBJECTIVES:

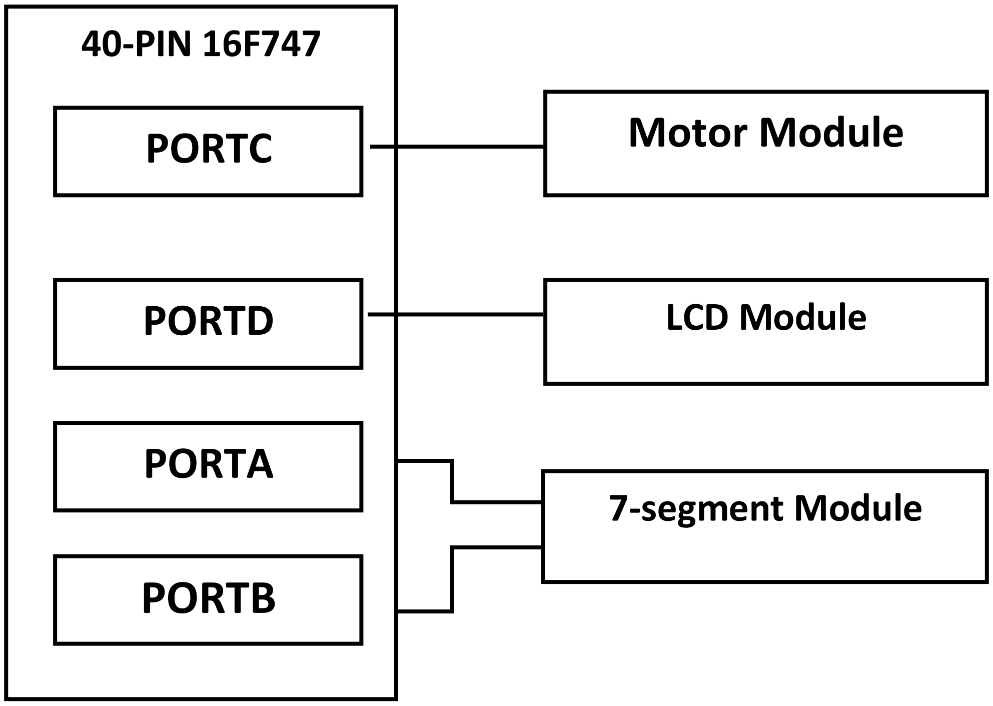
- To be able to use the Motor Module, LCD Module, and 7-segment Module of the PC-Based Microcontroller Programmer and Training Kit simultaneously
- To be able to control a Stepper Motor, display its degrees position in the 7-segment display, and display its direction and speed in the LCD Display
- To be able to use an input device in the form of a pushbutton to control a Stepper Motor
- To be able to interface a 40-pin PIC MCU with Stepper Motor, LCD, and 7-segment Display

MATERIALS:

- PC-Based Microcontroller Programmer and Training Kit
- PC with Windows XP running PICSoft
- PIC16F747 40-Pin PIC MCU

INTRODUCTION:

PIC MCU PORTS



In this experiment involving three modules, the LCD Module will be connected to PORTD, the Motor Module will be connected to PORTC, and the 7-segment Module will be connected to PORTA and PORTB.

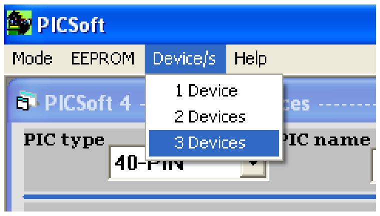
For detailed information regarding interfacing with Motor, LCD, or 7-segment module, refer to Laboratory Experiments 2, 4 and 5.

PROCEDURES:

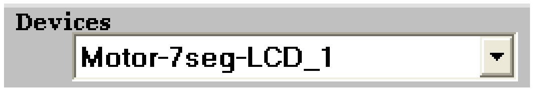
Before starting with the procedures, make sure you have already prepared the kit and installed the Device Driver and the Software. (See Chapter 2 for help).

A. Using the GUI and Virtual Emulator

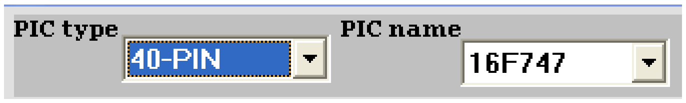
1. Open PICSoft on the computer.
2. Go to Menu Bar > Device/s > 3 Devices



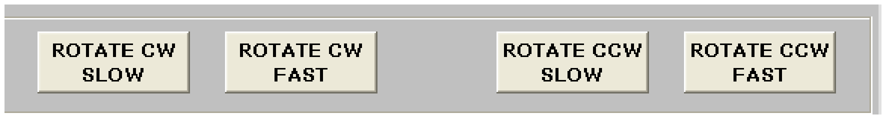
3. The form for 3 Devices would appear.
4. Select the ‘Motor-7seg-LCD_1’ in the Devices tab.



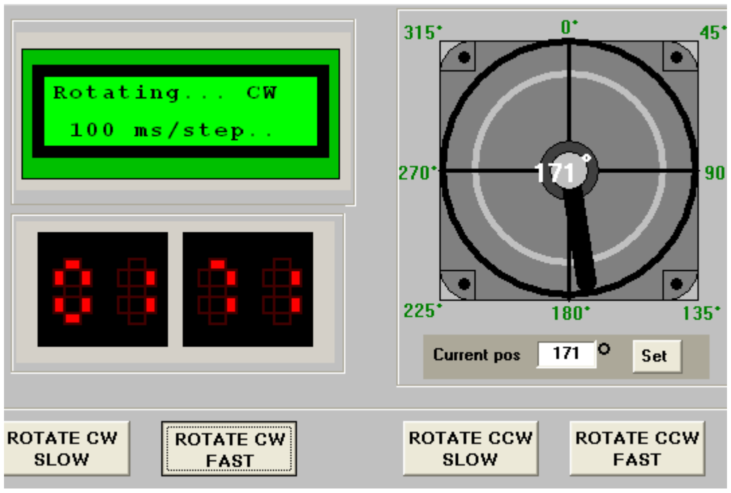
5. Select 40-PIN in the PIC type listbox and 16F747 in the PIC name listbox.



6. This experiment leaves the control of the Stepper Motor to the four pushbuttons and there are no other provided options. Each one rotates the motor for a distinct direction and speed. Click these buttons, hold for a moment, and release the mouse click to stop.



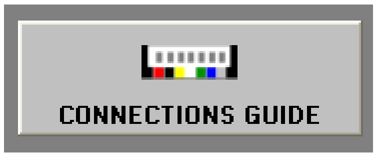
7. Notice how the motor moves, the position degrees on the 7-segment display and the status in the LCD. Try out these four controls and note their differences.



8. Click ‘Compile’.

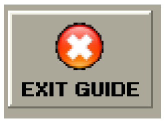
9. Click ‘OK’ when the Successful Compilation prompt appears.

10. At this point the device connections must be done before burning. Click the “Connections Guide” at the bottom- right side.

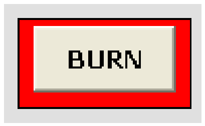


11. Follow the instructions and navigate through them by using the Prev and Next buttons. This guide will show you how to properly connect the headers for programming and testing of the compiled program.

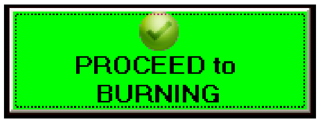
12. Click Exit at the end of the guide (Connect USB page).



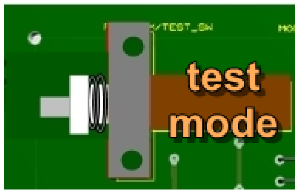
13. At this moment, click Burn.



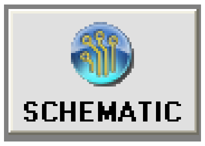
14. A confirmation window would appear. You can disregard this and click “Proceed to Burning” if you have read the connections guide earlier.



15. Click ‘Yes” at the Burning Confirmation message box.
16. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Complete prompt would appear. Close the window.
17. Push the Program/Test switch at the USB PIC Programmer to test the program.

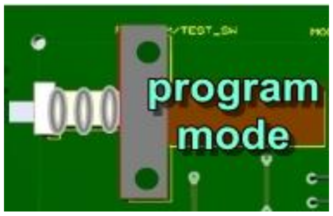


18. Try the actual circuit consisting of the three modules. Use the four pushbuttons located in the 7-segment module to control the motor. Check the accuracy of the degree position in the 7-segment and the motor’s status in the LCD display. Don’t close PICSoft.
19. Click “Schematic”. You should now see the schematic representation of the connections done in the kit. This could also help you in programming.



B. Using the Code Editor

1. Push the Program/Test switch back to go back into program mode.



2. Go back to the main window by clicking Exit at the Schematic Window (if it is still open).
3. Click “View Code”.

4. The PICpad editor will appear with the C code of the CountUp experiment. It is advised to read section 2.3 “C Language Structure” beforehand to get acquainted with coding.
5. As you will notice, the code is long and contains some parts that can be difficult to understand. Make sure to read all of the comments to help you understand the purpose of each line.
6. Notice the `//########### Global Variables and Arrays ##########` block in its middle section `//LCD declares`

```
char LCDT[17] ;
char LCDB[17] ;
          //123456789ABCDEFG
char CWF[] = "Rotating...  CW";
char CWS[] = "Rotating...  CW";
char CCWF[] = "Rotating...  CCW";
char CCWS[] = "Rotating...  CCW";
char FAST[] = " 100 ms/step.. ";
char SLOW[] = " 50 ms/step.. ";
char un[] = "  ............ ";
char wa[] = "waiting 4 input ";
```

These codes declare the string variables that hold the motor's status messages for the LCD Display. Line 101 and 102 reserves a string variable with 17 spaces each. The LCD has only 16 characters, plus 1 character for the string terminator. Lines 104-111 declared the string literals containing the texts that they contain.

7. Notice the `//  ############# program loop ##############` block

```
while(1 == 1)                    //  Loop Forever
  {

     if (RA5 == 0)                        //  CW-Fast

     {
      strcpy(LCDT,CWF);             //  Copy text to the Top LCD variable
      strcpy(LCDB,FAST);            //  Copy text to the Bottom LCD variable

      __delay_ms(75);               // delay for each step

      PORTC = StepperTable[s];   // step motor based from the Stepper Table

      s = (s + 1) % 4;               // increment step

      if (curDeg >= 360)
       {
         curDeg = 0;                // reset to 0 if reached 360 deg.
       }
      curDeg = curDeg + 1.8;     // increment degrees variable for display

      ctr = (int) curDeg;            // convert  float variable to int

     }
                      . . .
     else if (RA6 == 0)                   //  CW Slow


                      . . .
     else if (RA3 == 0)                   //   CCW Fast
                      . . .
     else if (RA4 == 0)                       // CCW Slow
                      . . .
     else                           // none pressed- waiting for input
                             . . .
     d1 = LEDChar[dig1] ^ 0b11111111 ;                    // display Digit 1
     d2 = LEDChar[dig2] ^ 0b11111111 ;                    // display Digit 2
     d3 = LEDChar[dig3] ^ 0b11111111 ;                    // display Digit 3
     d4 = LEDChar[dig4] ^ 0b11111111 ;                   // display Digit 4


     LCD_clear();

     LCD_goto(0);              // select first line
     LCD_puts(LCDT);           // print Top text
     LCD_goto(0x40);           // Select second line
     LCD_puts(LCDB);           // print bottom text

  } // elihw
```

Majority of the operation of the PIC MCU in this experiment is dependent on polling of the buttons. ‘Polling’ refers to waiting for the buttons to be pressed, knowing which is pressed, and executing code depending on which is pressed and how long it is pressed.

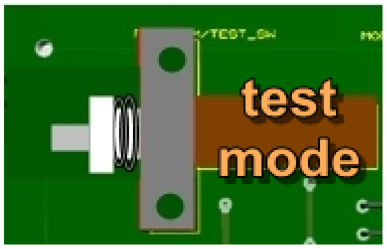
8. The code inside the ‘if’ block from 237 – 255 is what happens when the CW-Fast (Clockwise Fast) button (2nd to the left-connected to RA5) is pressed.
9. Line 237 copies the contents of the CWF variable to the LCDT variable. CWF is a declared variable that contains a string, as shown in Step 6. The LCDT variable is the one printed later in the Top LCD display line.
10. Line 240 contains the 75 ms delay of every step
11. Line 242 supplies PORTC with a string value from the array. The logic signals will then be sent to the motor coils through the L293D IC driver.
12. Line 244 increments the array subscript for the next step.
13. Lines 246-248 contains an if block that resets the degree count to 0 if it reaches or exceeds 360.. This keeps the degree count from getting large and out of range. Besides, 360° = 0° is theoretically true for circles.
14. Line 250 increments the degree counter ‘curDeg’ with 1.8° for every step. Line 252 converts ‘curDeg’ to an integer value that can be displayed by the LCD.
15. For the other buttons to be polled, the same concept goes for the code inside the if statements in lines 256, 278, and 301.
16. The ‘else’ statement in line 323 contains the code to be executed when there is no button pressed and the program is waiting for input.
17. Lines 377-380 handles displaying of the degree position in the 7-segment display. Each of the 4 variables (d1, d2, d3, d4) gets the pattern from the LEDChar[ ] array and XOR it with 0b11111111 to negate each bit. This is necessary because the 7-segment used is common-anode type, where the patterns are loaded in the cathodes.

18. Line 383 prepared the LCD display by clearing it.
19. Line 385 – 388 can be self-explanatory through the comments included. The LCDT and LCDB are the reserved string variables (explained in Step 6), contained text through the strcopy() command (explained in Step 9). Apparently, the LCDT and LCDB contents depend on whether a button is pressed and which one if there is. The LCD display reflects the algorithm on how the four buttons are polled.
20. Feel free to edit the code as desired. You can adjust the motor rotation speed by replacing the number inside the __delay_ms( ) function (Lines 240, 262, 284, and 307).
21. You can swap the button's functions or assigned them manually (Lines 234, 256, 278, 301, and 323). One way is to swap pins (e.g. RA4 for RA5).
22. You can also modify the text displayed by the LCD by editing the string variable contents (lines 104-111). Make sure to not exceed 16 characters or the text will be truncated.
23. You can also adjust the delay by replacing the number inside the __delay_ms( ) function.
24. This time, we will not change the PIC type and PIC name so we do not have to change the connections.
25. To test the codes, click the Save button.
26. Click Compile button. If there are no errors, you would be prompted to burn the program. In case there are errors, compiling would not finish and the errors will be displayed.

27. Click Burn then click ‘Proceed with Burning’.

28. Wait until burning is finished. If all the proper connections are made and if there are no errors met, the Burning Success prompt would appear. Close the window.

29. Push the Program/Test switch at the USB PIC Programmer to test the program. Observe the actual Motor and LCDs modules if the program is working appropriately.



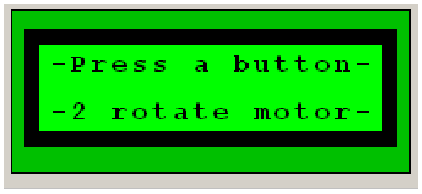
30. To modify the program again, push it back to Program mode and follow the steps above in editing, saving, compiling, burning and testing back.

EXERCISES:

1. Modify the program so that the four button functions are laid out like this. Write the pins assigned to each button in the blank provided.

ROTATE CCW FAST	**ROTATE CCW SLOW**	**ROTATE CW FAST**	**ROTATE CW SLOW**
____________	____________	____________	____________

2. Modify the program so that the LCD would display a message like the one below when waiting for input.

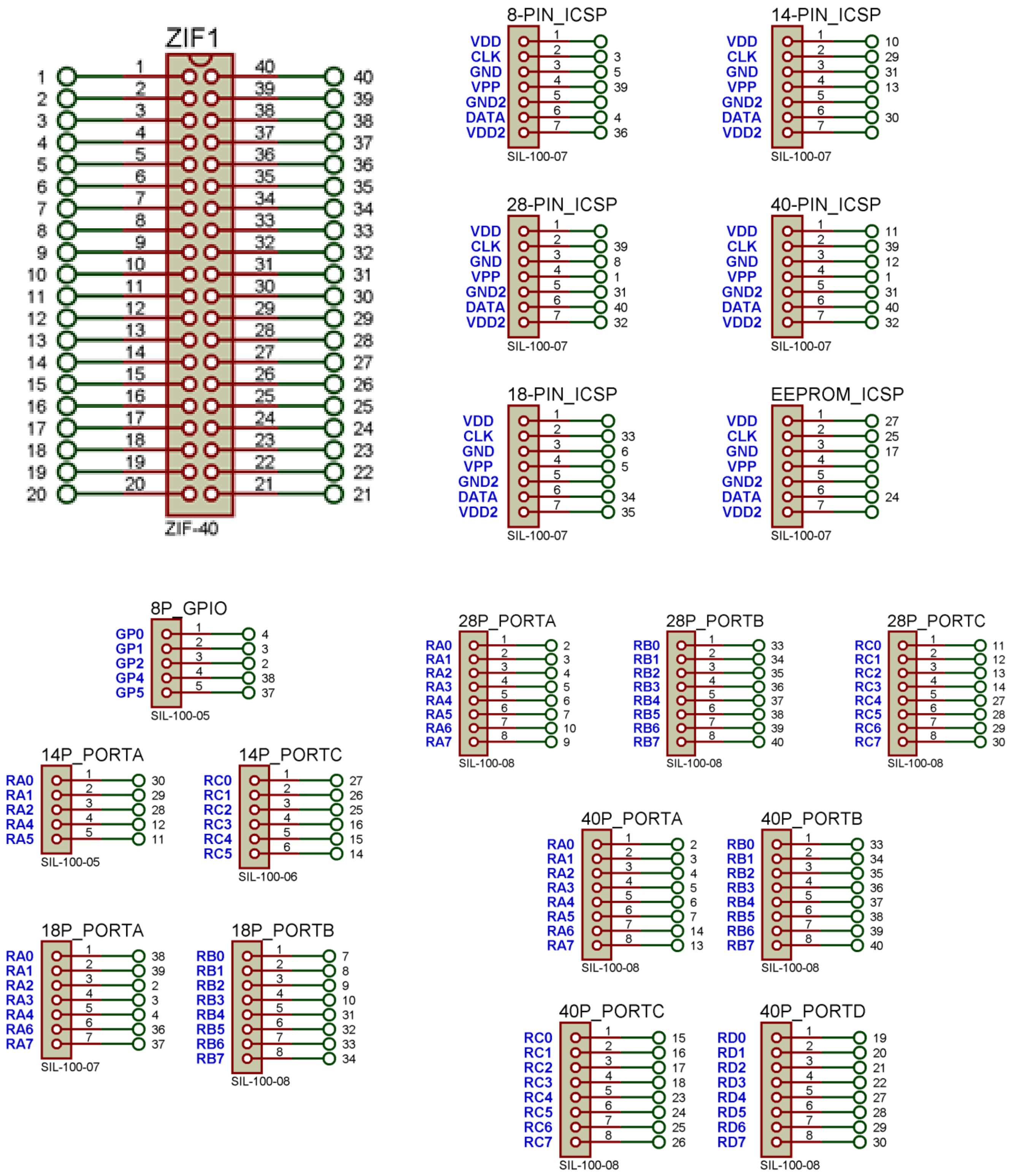


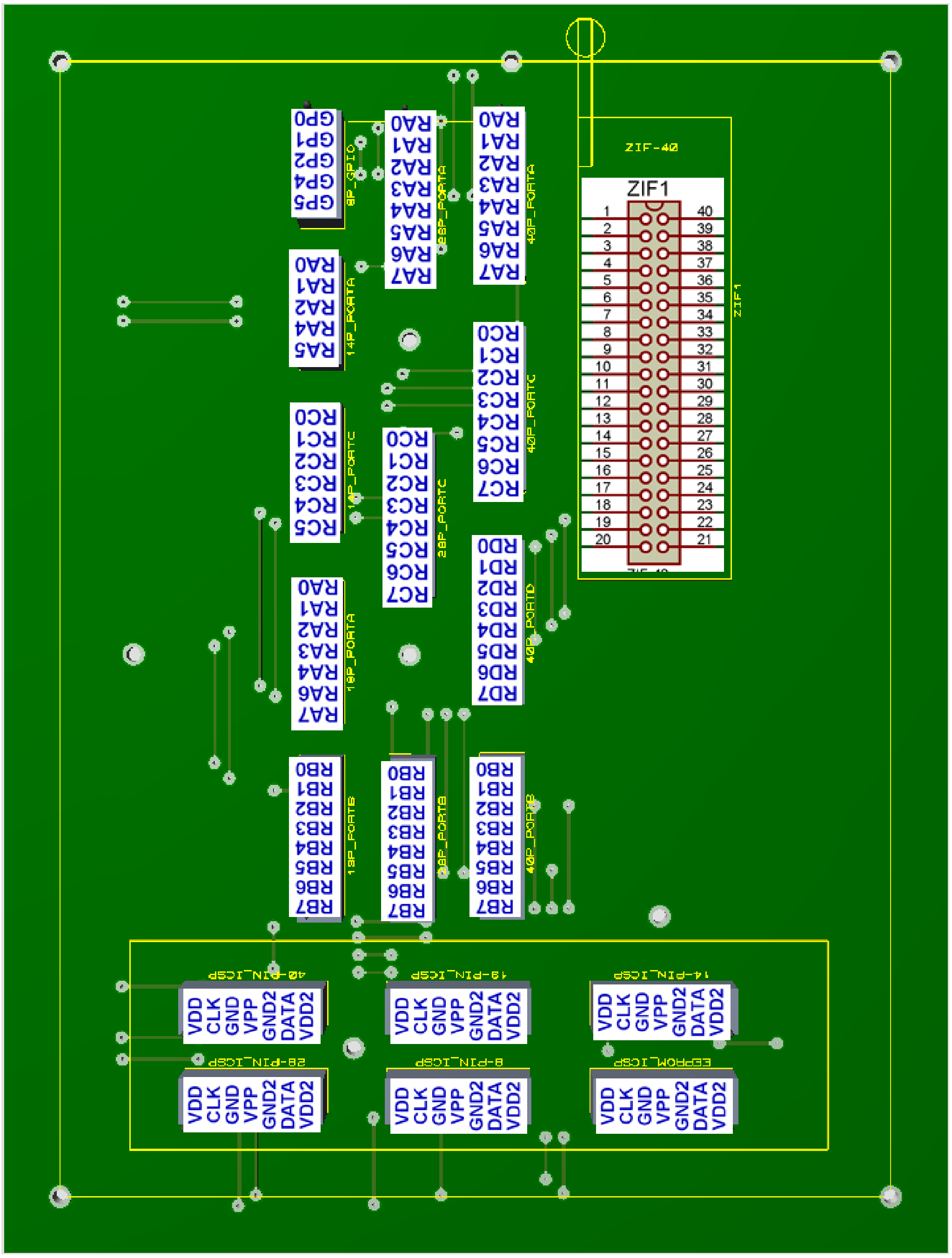
```
-Press a button-
-2 rotate motor-
```

OBSERVATION:

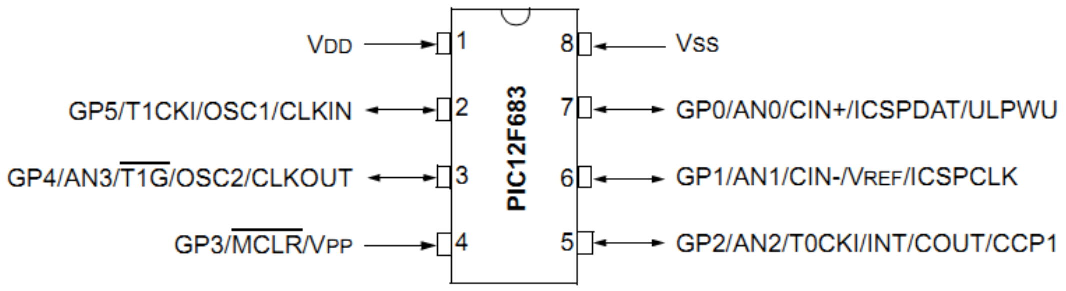
CONCLUSION:

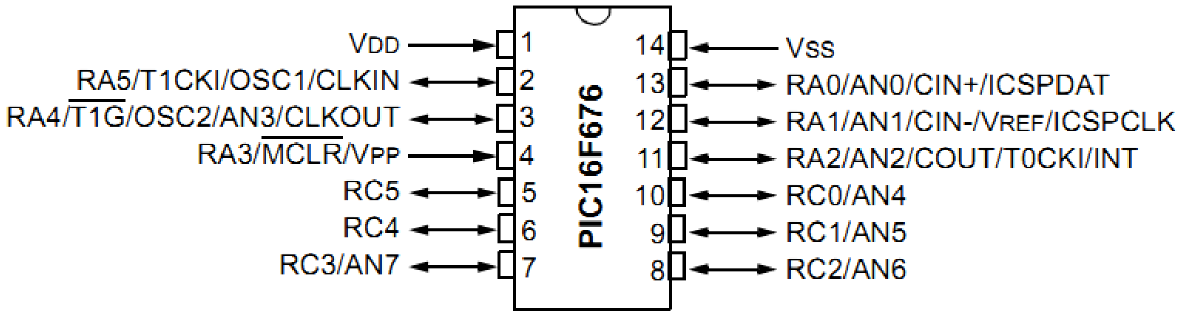
5 CIRCUIT CONNECTIONS

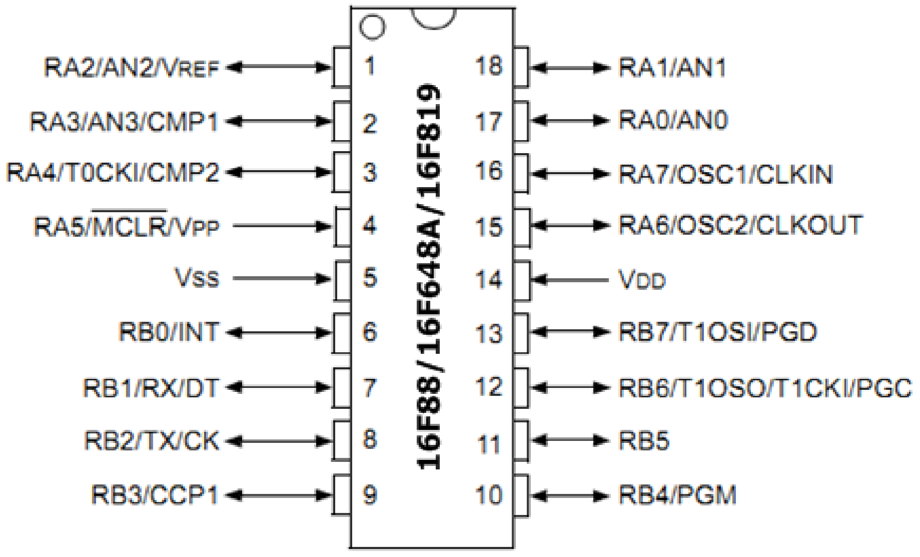


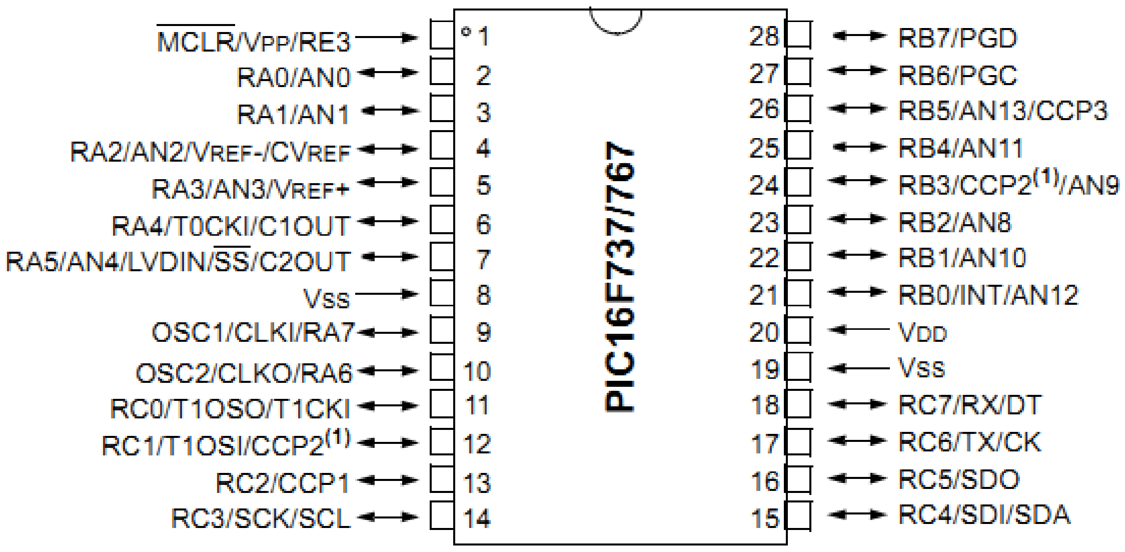

ZIF-40
ZIF1
VDD
CLK
GND
VPP
GND2
DATA
VDD2

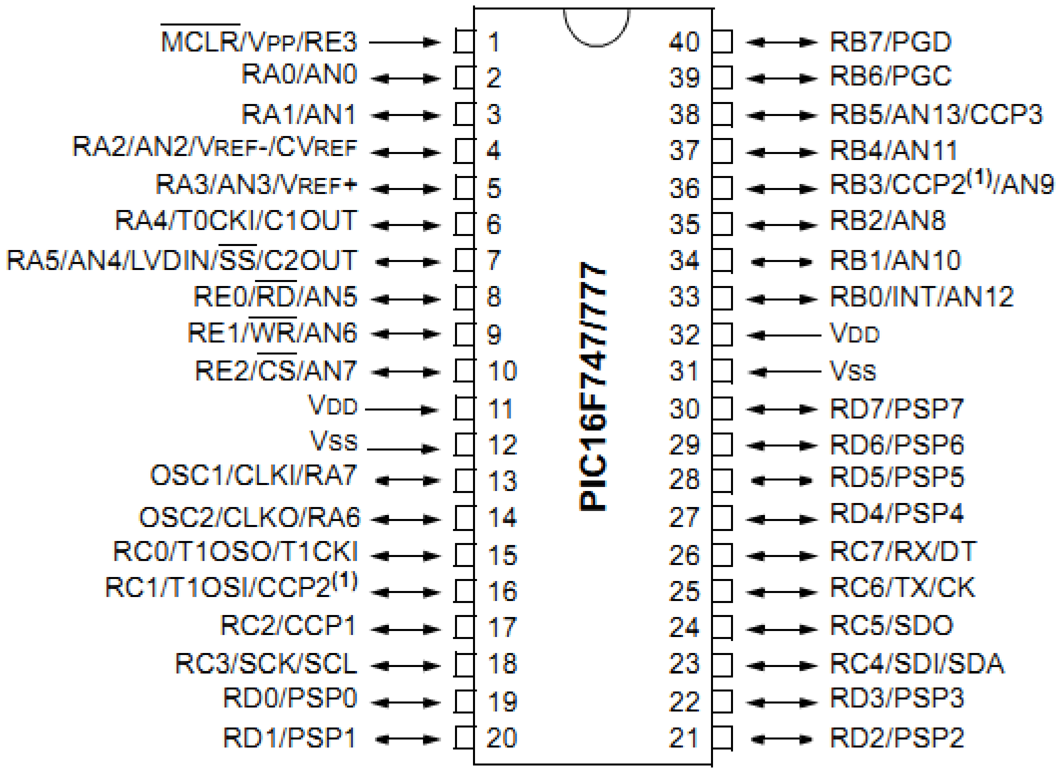
6 PIC MCU PINOUTS

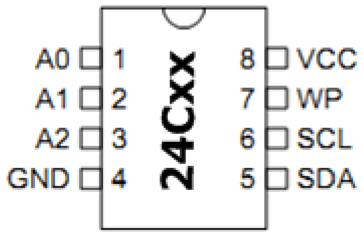

PIC12F683
VDD 1
GP5/T1CKI/OSC1/CLKIN 2
GP4/AN3/$\overline{T1G}$/OSC2/CLKOUT 3
GP3/$\overline{MCLR}$/VPP 4
8 VSS
7 GP0/AN0/CIN+/ICSPDAT/ULPWU
6 GP1/AN1/CIN-/VREF/ICSPCLK
5 GP2/AN2/T0CKI/INT/COUT/CCP1


PIC16F676
VDD 1
RA5/T1CKI/OSC1/CLKIN 2
RA4/$\overline{T1G}$/OSC2/AN3/CLKOUT 3
RA3/$\overline{MCLR}$/VPP 4
RC5 5
RC4 6
RC3/AN7 7
14 VSS
13 RA0/AN0/CIN+/ICSPDAT
12 RA1/AN1/CIN-/VREF/ICSPCLK
11 RA2/AN2/COUT/T0CKI/INT
10 RC0/AN4
9 RC1/AN5
8 RC2/AN6




PIC16F737/767
MCLR/VPP/RE3 1
RA0/AN0 2
RA1/AN1 3
RA2/AN2/VREF-/CVREF 4
RA3/AN3/VREF+ 5
RA4/T0CKI/C1OUT 6
RA5/AN4/LVDIN/SS/C2OUT 7
VSS 8
OSC1/CLKI/RA7 9
OSC2/CLKO/RA6 10
RC0/T1OSO/T1CKI 11
RC1/T1OSI/CCP2(1) 12
RC2/CCP1 13
RC3/SCK/SCL 14
28 RB7/PGD
27 RB6/PGC
26 RB5/AN13/CCP3
25 RB4/AN11
24 RB3/CCP2(1)/AN9
23 RB2/AN8
22 RB1/AN10
21 RB0/INT/AN12
20 VDD
19 VSS
18 RC7/RX/DT
17 RC6/TX/CK
16 RC5/SDO
15 RC4/SDI/SDA


PIC16F747/777
MCLR/VPP/RE3 1
RA0/AN0 2
RA1/AN1 3
RA2/AN2/VREF-/CVREF 4
RA3/AN3/VREF+ 5
RA4/T0CKI/C1OUT 6
RA5/AN4/LVDIN/SS/C2OUT 7
RE0/RD/AN5 8
RE1/WR/AN6 9
RE2/CS/AN7 10
VDD 11
VSS 12
OSC1/CLKI/RA7 13
OSC2/CLKO/RA6 14
RC0/T1OSO/T1CKI 15
RC1/T1OSI/CCP2(1) 16
RC2/CCP1 17
RC3/SCK/SCL 18
RD0/PSP0 19
RD1/PSP1 20
40 RB7/PGD
39 RB6/PGC
38 RB5/AN13/CCP3
37 RB4/AN11
36 RB3/CCP2(1)/AN9
35 RB2/AN8
34 RB1/AN10
33 RB0/INT/AN12
32 VDD
31 VSS
30 RD7/PSP7
29 RD6/PSP6
28 RD5/PSP5
27 RD4/PSP4
26 RC7/RX/DT
25 RC6/TX/CK
24 RC5/SDO
23 RC4/SDI/SDA
22 RD3/PSP3
21 RD2/PSP2


24Cxx
A0 1
A1 2
A2 3
GND 4
8 VCC
7 WP
6 SCL
5 SDA

7 TROUBLESHOOTING GUIDE

This section contains information on the common situations and problems that may be encountered when using the device, as well as the recommended solutions.

- **PICSoft won't open.**

  Make sure that PICSoft is installed. See Section 1.3 for instructions in installing the program.

- **There are some errors in compilation**

  In Advanced Mode / PICPad, compilation errors appear because of errors in the program code. Choose to view the error list to see information on the error, including its description and the lines of code that caused it. See section 2.3 for more details.

- **The program failed to burn**

  There are five main requirements to burn the program into the MCU:

  - Device driver – make sure device driver is installed. A green LED in the USB PIC Programmer indicates that it is installed. A red LED, however, indicates that it is not yet installed. See section 1.2 for instructions on device driver installation.
  - P/T (Program/Test) switch - make sure that the switch is in program mode before burning. Burning in Test mode is impossible for the programming lines are closed by the switch to give way for testing.
  - USB cable – make sure that the USB cable is connected to the host PC.
  - ICSP header – make sure that the white 7-pin ICSP header is connected to the target PIC type, either 8-pin, 18- pin, 28- pin, 40- pin, or EEPROM.
  - PIC MCU – make sure that the intended PIC MCU is inserted in the ZIF socket in its correct position.

Other solutions can be tried if burning still fails after following the five requirements correctly. The following steps can be tried in order until the program succeeds in burning:

- P/T reset – after burning failed, press the P/T switch shortly to test mode then press it back to program mode.
- Reconnect USB cable – if burning still fails, disconnect the USB cable from the host PC and connect it back again.
- Recompile – compile the program again.
- Try burning into other PIC type (e.g. 28-pin instead of 18-pin).
- Remove other USB loads from the PC (e.g. USB mouse, USB fan, card reader, etc)
- Restart PICSoft.
- Restart PC.

Other notes when burning:

When the Burn button is clicked / the burning process is ongoing:

- DO NOT press the P/T switch, either if it is in Program or Test mode. Let the program generate an error if it is in Test mode, then just Burn again with the switch in Program mode after closing the "Burning Failed" window. Pressing the P/T switch makes sudden changes to the programming lines and will cause some more errors.
- DO NOT disconnect the USB cable. In cases of burning with the USB disconnected, let the program generate an error, then just Burn again with the USB connected after closing the "Burning Failed" window.

• **The LCD displays some unrecognized characters or it displays incorrectly**

After burning, the LCD might display incorrectly or display some strange characters. Press the green Reset button in the USB PIC Programmer to refresh the display

8 FAQ (Frequently Asked Questions)

- What are the requirements in using the PC-Based Microcontroller Programmer and Training Kit?
    - The package includes all hardware components including the PIC MCUs and the USB cable. However, the host PC must be running any edition of Windows 2000/ME/XP. Only PICSoft is needed to be installed. The Hi-Tech PICC Compiler and WinPic are included in its installation files. In addition, a PDF viewer is also required for viewing the help files. A free PDF viewer can be downloaded from this link. [http://foxitsoftware.com/Secure_PDF_Reader/]

- Where does the device get power?
    - The device obtains the standard +5V from the USB port of the host PC. An external power supply header and power selector switch is also attached in the USB PIC Programmer that allows the device to run from an external +5V source. More details can be found in Section 2.1 - Familiarization with Hardware

- Will the device work without a host computer?
    - Yes, but only for testing the PIC MCU which already has code. While connected to an external +5V, the device can be used without the USB connection to a host PC to run the program inside the PIC MCU. However, the PIC MCU cannot be reprogrammed in this manner.

- I have my own C code written in Hi-Tech PICC. How can I compile this code?
    - First, open PICSoft. Then select the PIC type and PIC name that you're using. Afterwards, in the Menu Bar, click Mode > Advanced (PICpad). A

    blank code editor would appear, paste your code here and then save, compile, then burn.

- I have my own HEX object code. How can I burn this into the PIC MCU directly?
    - First, open PICSoft. Then, in the menu bar, click WinPic. This would open the WinPic interface which will allow you to load your hex and burn it directly to your PIC MCU. The list of all supported PIC MCUs for directly burning .hex files can also be found in WinPic.

- Where are the C files and their compiled HEX files located?
    - All of these files can be found in the “bin” folder inside the PICSoft directory.

- Where can I access more information about the device?
    - The Help section inside PICSoft contains articles, guides, and the workbook itself. The documents are in PDF format and would open in a separate window. The next section also describes references used in the development of the project.

9 References / Suggested Reading

123 PIC Microcontroller Experiments for the Evil Genius

By Myke Predko

McGraw-Hill/TAB Electronics **ISBN-13:** 978-0071451420
More than just hours of fun, these exciting experiments provide a solid grounding in PIC microcontrollers and the skills needed to program them -- from the ground up. Each experiment builds on those before it, so you develop a hands-on, practical understanding of microcontroller programming. You don't need any knowledge of programming to get started. But by the end, you'll be able to complete your own awesome projects!

GTP USB PIC PROGRAMMER (Open Source)

http://www.instructables.com/id/GTP-USB-PIC-PROGRAMMER-Open-Source/

This entry in the DIY (Do-It-Yourself) Instructables.com website discusses the open source design of the GTP USB PIC Programmer, which is a burner for the PIC Microcontroller. It includes the firmware, burning software (WinPic800), schematic, and board design for the circuit. It is an excellent resources for PIC enthusiasts and students, especially those looking for an improvised PIC burner using the USB port.

PIC Basic Projects: 30 Projects using PIC BASIC and PIC BASIC PRO

By Dogan Ibrahim

Newnes **ISBN-13:** 978-0750668798
Covering the PIC BASIC and PIC BASIC PRO compilers, PIC Basic Projects provides an easy-to-use toolkit for developing applications with PIC BASIC. Numerous simple projects give clear and concrete examples of how PIC BASIC can be used to develop electronics applications, while larger and more advanced projects describe program operation in detail and give useful insights into developing more involved microcontroller applications.
Including new and dynamic models of the PIC microcontroller, such as the PIC16F627, PIC16F628, PIC16F629 and PIC12F627, PIC Basic Projects is a thoroughly practical, hands-on introduction to PIC BASIC for the hobbyist, student and electronics design engineer.