AC 2011-1613: TEACHING MICROCONTROLLERS THROUGH SIMULATION

Nikunja Swain, South Carolina State University

Dr. Swain is currently a Professor at the South Carolina State University. Dr. Swain has 25+ years of experience as an engineer and educator. He has more than 50 publications in journals and conference proceedings, has procured research and development grants from the NSF, NASA, DOT, DOD, and DOE and reviewed number of books on computer related areas. He is also a reviewer for ACM Computing Reviews, IJAMT, CIT, ASEE, and other conferences and journals. He is a registered Professional Engineer in South Carolina.

Teaching Microcontrollers through Simulation

Abstract

There are numerous uses of simulation, starting from simulation of simple electric circuits to complex tasks such as electromagnetic fields, heat transfer through materials, networking, computer circuits, game programming, electron flow in semiconductors, or beam loading with the ultimate objective of providing illustrations of concepts that are not easily visualized and difficult to understand. Simulators are also used as an adjunct to and, in some cases such as distance learning courses, as a substitute for actual laboratory experiments. In many instances, students are required to verify their theoretical design through simulation before building and testing the circuit in the laboratory. Studies show that students who used simulation prior to conducting actual experiments performed better than the students who conducted the laboratory experiments without conducting simulation first. Also, simulation is used to model large and complex systems. There is no doubt that simulation cannot replace the physical hands-on experience, but simulation can enhance the teaching and learning experience. The objective of this paper is to discuss microcontroller simulation software packages for 8051 and PIC microcontroller and its effect on education and research.

Introduction

Automation is becoming part and parcel of every industry, and industries need a trained workforce to manage this new development. Engineering and technology graduates must have a comprehensive background covering a wider range of technical subjects. The graduates must be proficient in the use of engineering and scientific equipment, conducting experiments, collecting data, and effectively presenting the results [1, 2, 3, 4]. Furthermore, these graduates must be well-trained in courses and laboratories such as electric and electronic circuits; digital systems and microprocessors; computer programming; computer aided design; computer organization and architecture; electronic and data communications; networking; control and robotics; electric machines and power systems; PLC and virtual instrumentation; microprocessors and microcontrollers and others. One cost-effective way of achieving this is through the use of simulation software programs, and a number of simulation software packages are available for these purposes. These software packages play an important role in education and are used to deliver training for all kinds of activities, from piloting sophisticated aircraft or ships to operating nuclear power plants or complex chemical processing facilities.

Packages like PSPICE, Multisim, MatLab, Simulink, LogicWorks, RSLogix, Debug, MASM, and LabVIEW are widely used by engineering and technology programs at other institutions and there is sufficient information on these in textbooks and on the web. Packages like PIC 18 Simulator IDE and EDSIM 51 are not that well known and may not be widely used but both these packages have tremendous potential in enhancing student learning in Microcontroller course. We will be discussing some of the features of these software packages and the instructional modules developed using these packages below.

EDSIM51 Instructional Module

Features of EDSIM 51

The EdSim51 [5] Simulator is a free simulator for the popular 8051 microcontroller. In EDSIM51, a virtual 8051 is interfaced with virtual peripherals such as a keypad, motor, display, UART, etc. The student can write 8051 assembly code, step through the code and observe the effects each line has on the internal memory and the external peripherals. The following is a list of virtual peripherals in EDSIM51:

- Analogue-to-Digital Converter (ADC)
- Comparator
- UART
- 4 Multiplexed 7-segment Displays
- 4 X 3 Keypad
- 8 LEDs
- DC Motor
- 8 Switches
- Digital-to-Analogue Converter (DAC) - displayed on oscilloscope

Example of EDSIM51 Simulation Module

This simulation module introduces the student to LCD Module in EDSIM51. The LCD module connections in EDSIM51 are shown in Figure 1. As shown in Figure 1, the EdSim51 8051 simulator Port 1 is connected to an LCD module as seen below. The LCD module is a simulation of the Hitachi HD44780. The LCD module is configured in 4-bit mode, with DB7-DB4 connected to P1.7-P1.4. P1.3 is connected to the register-select pin and P1.2 is connected to the enable pin.

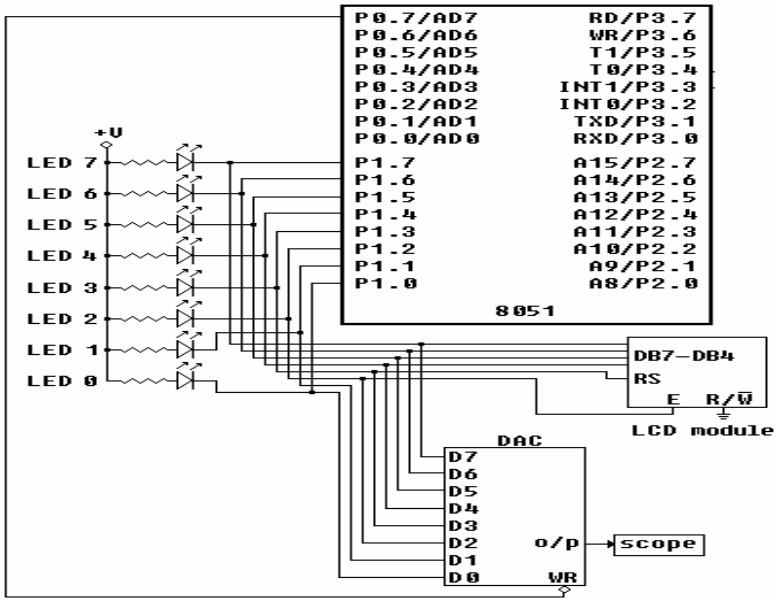

Figure 1 – LCD configuration in EDSIM51

To display the text on the LCD module, Port 1 bits will have to be set and cleared to set the bit mode (4

or 8), function, entry mode, and display control of the LCD module. Then the values to be sent to the display can be moved from the RAM of the 8051. The instruction sets and different settings can be found in the user guide of the HD 44780. To write a program that will send the text 8051 PROGRAM to the LCD display, the values must be stored in the RAM of the 8051. Also, the LCD display must be set to properly display the data. Because the LCD module is configured in 4 bit mode, the settings and data must be sent 4 bits at a time. At the completion of each instruction, P1.2 must be set and then cleared. This will control the enable bit, which will send the instruction to the LCD module. Once the function, entry mode, and display on/off control settings have been set, the data can be moved from RAM to the LCD module in a similar manner (4 bits at a time). The assembly language program to send the text 8051 PROGRAM to the LCD display for this module is shown below and the corresponding simulation screenshot is shown in Figure 2.

Assembly Language Program

```
 ORG 0

 ; MOVE DATA INTO RAM

 MOV 30H, #'8' ; PUT DATA IN RAM
 MOV 31H, #'0' ; PUT DATA IN RAM
 MOV 32H, #'5' ; PUT DATA IN RAM
 MOV 33H, #'1' ; PUT DATA IN RAM
 MOV 34H, #' ' ; PUT DATA IN RAM
 MOV 35H, #'P' ; PUT DATA IN RAM
 MOV 36H, #'R' ; PUT DATA IN RAM
 MOV 37H, #'O' ; PUT DATA IN RAM
 MOV 38H, #'G' ; PUT DATA IN RAM
 MOV 39H, #'R' ; PUT DATA IN RAM
 MOV 3AH, #'A' ; PUT DATA IN RAM
 MOV 3BH, #'M' ; PUT DATA IN RAM
 MOV 3CH, #0 ; END OF DATA MARKER

 CLR P1.3              ; CLEAR RS (P1.3) TO INDICATE                          ;
INSTRUCTIONS BEING SENT TO MODULE

 CLR P1.7 ; SET FUNCTION 1ST TIME TO TELL
 CLR P1.6  ; MODULE TO GO TO 4-BIT MODE
 SETB P1.5  ; N = 1 (4 BIT MODE)
 CLR P1.4 ; DL = 0

 SETB P1.2             ; SET NEGATIVE EDGE ON E SO MODULE
 CLR P1.2              ; WILL READ HIGH NIBBLE
 CALL DELAY            ; DELAY TO WAIT FOR BF TO CLEAR

 SETB P1.2             ; SET NEGATIVE EDGE ON E SO MODULE
 CLR P1.2              ; WILL READ HIGH NIBBLE FOR 2ND TIME

 SETB P1.7             ; SET LOW NIBBLE OF FUNCTION SET

 SETB P1.2             ; SET NEGATIVE EDGE ON E SO MODULE
 CLR P1.2              ; WILL READ LOW NIBBLE OF FUNCTION SET
 CALL DELAY            ; DELAY TO WAIT FOR BF TO CLEAR

 CLR P1.7              ; SET HIGH NIBBLE OF ENTRY MODE
```

```
 CLR P1.6              ; 0
 CLR P1.5              ; 0
 CLR P1.4              ; 0
 SETB P1.2             ; SET NEGATIVE EDGE ON E SO MODULE
 CLR P1.2              ; WILL READ HIGH NIBBLE OF ENTRY MODE
 SETB P1.6             ; SET LOW NIBBLE OF ENTRY MODE
 SETB P1.5             ; BITS 5 AND 6 ONLY NEEDED CHANGE
 SETB P1.2             ; SET NEGATIVE EDGE ON E SO MODULE
 CLR P1.2              ; WILL READ LOW NIBBLE OF ENTRY MODE
 CALL DELAY            ; DELAY TO WAIT FOR BF TO CLEAR
 CLR P1.7              ; SET HIGH NIBBLE OF DISPLAY CONTROL
 CLR P1.6              ; 0
 CLR P1.5              ; 0
 CLR P1.4              ; 0

 SETB P1.2             ; SET NEGATIVE EDGE ON E SO MODULE
 CLR P1.2              ; WILL READ HIGH NIBBLE OF DISPLAY

 SETB P1.7             ; SET LOW NIBBLE OF DISPLAY CONTROL
 SETB P1.6             ; 1 = DISPLAY ON
 SETB P1.5             ; 1 = CURSOR ON
 SETB P1.4             ; 1 = BLINKING ON

 SETB P1.2             ; SET NEGATIVE EDGE ON E SO MODULE
 CLR P1.2              ; WILL READ THE LOW NIBBLE OF DISPLAY
 CALL DELAY            ; DELAY TO WAIT FOR BF TO CLEAR

 SETB P1.3             ; SEND DATA FROM RAM TO MODULE
 MOV R1, #30H          ; MOVE DATA TO BE SENT FROM RAM TO R1

LOOP: MOV A, @R1           ; MOVE DATA POINTED TO BY R1 TO A
 JZ FINISH             ; IF A IS 0, JUMP OUT OF LOOP
 CALL SENDDATA         ; SEND DATA TO LCD
 INC R1                ; INCREMENT R1 TO POINT TO NEXT DATA
 JMP LOOP              ; REPEAT UNTIL DONE
FINISH: JMP $               ; FINISH PROGRAM ONCE DATA IS WRITTEN

SENDDATA: MOV C, ACC.7         ; SEND DATA FROM HIGH BITS OF A
 MOV P1.7, C           ; TO LCD MODULE
 MOV C, ACC.6          ; MOVE BIT 6 TO C
 MOV P1.6, C           ; MOVE C TO P1.6
 MOV C, ACC.5          ; MOVE BIT 5 TO C
 MOV P1.5, C           ; MOVE C TO P1.5
 MOV C, ACC.4          ; MOVE BIT 4 TO C
 MOV P1.4, C           ; MOVE C TO P1.4

 SETB P1.2             ; SET NEGATIVE EDGE ON E SO MODULE
 CLR P1.2              ; WILL READ THE HIGH NIBBLE OF DATA

 MOV C, ACC.3          ; SEND DATA FROM LOW BITS OF A
 MOV P1.7, C           ; TO LCD MODULE
 MOV C, ACC.2          ; MOVE BIT 2 TO C
 MOV P1.6, C           ; MOVE C TO P1.6
 MOV C, ACC.1          ; MOVE BIT 1 TO C
 MOV P1.5, C           ; MOVE C TO P1.5
 MOV C, ACC.0   ; MOVE BIT 0 TO C
 MOV P1.4, C  ; MOVE C TO P1.4
```

```
SETB P1.2              ; SET NEGATIVE EDGE ON E SO MODULE
CLR P1.2               ; WILL READ LOW NIBBLE OF DATA
CALL DELAY             ; DELAY TO WAIT FOR BF TO CLEAR
DELAY: MOV R0, #50     ; MOVE 50 TO R0
DJNZ R0, $             ; KEEP DECREMENTING R0
RET
```

Simulation Screen Shot for LCD

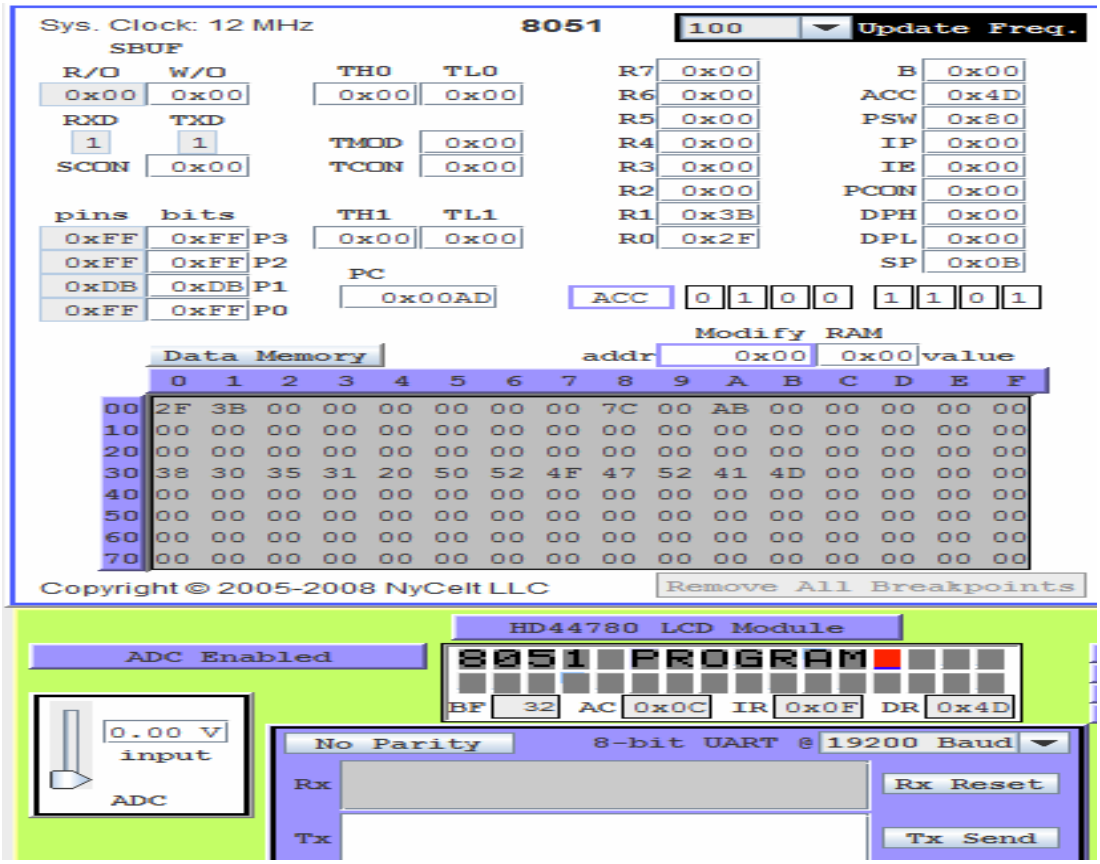

Figure 2 – Simulation screenshot for EDSIM51 LCD Simulation Module

PIC Microcontroller

A Peripheral Interface Controller (PIC) Microcontroller is a small integrated computer with a complete control system on a single physical chip, combining EEPROM program memory, data memory (RAM), data ROM, I/O ports, processor, Analog/Digital convertor, bus controllers and some other devices to run a simple application for a specific task. The Microchip PIC family microcontrollers are very popular and selected by more and more designers [6, 7]. The PIC family can break up into 8-Bit, 18-Bit, and 32-Bit groups as illustrated in Figure 3.

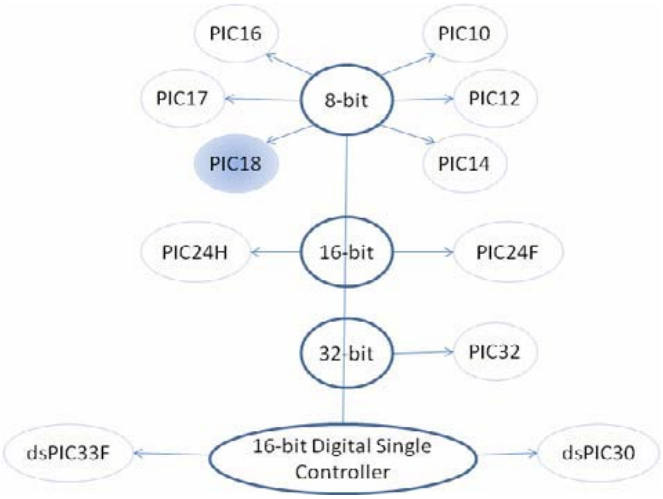

Figure 3 – PIC Microcontroller Family

PIC Microcontrollers are designed using the Harvard Architecture. A typical PIC Microcontroller unit consists of Microprocessor unit, program memory for instructions, I/O ports, memory for data, serial and parallel port communication, Analog to Digital convertor, and it supports different devices such as Timers. Figure 4 illustrates the PIC Microcontroller Architecture.

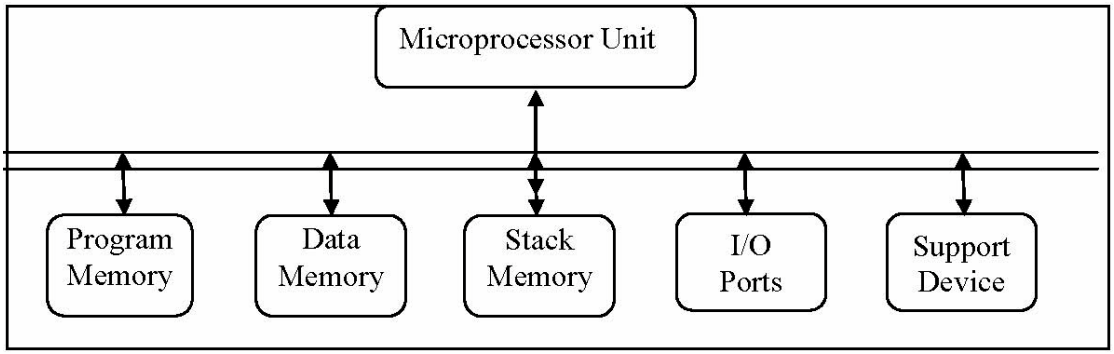

Figure 4 – PIC Microcontroller Architecture

PIC18 Family Features

PIC18 is the last version of 8-bits Microcontrollers family. The earlier versions of this family are 10, 12, 14, 16, and 17. Some peripherals of PIC18F and some of their feature as listed on Microchip's website [8] are:

- Power Control PWM (PCPWM)
- Quadrature Encoder Interface (QEI)
- Input Capture (IC)
- High Speed Analog-to-Digital Converter (ADC)

PIC Simulator IDE

PIC18 Simulator IDE [9] is a powerful simulator to simulate programs (assembly and Basic) for PIC18 microcontrollers. This simulator has number of features such as editing, compiling, assembling, debugging, linking, loading, simulation, and display of I/O module, memory, registers and others. Limited version (30 starts) of this simulator can be freely downloaded from www.oshonsoft.com. The following is a brief description of various features of this simulator:

Editor (Basic Compiler and Assembler)

The PIC Simulator IDE supports a Basic Compiler for Basic Language Programming and an Assembler for Assembly Language programming. Both the Basic compiler and Assembler compile the program and generate the hex code. The Hex code is then loaded and executed in the simulator and the results are displayed in the respective display devices.

I/O Modules (Displays)

I/O devices are essential components of Microcontroller based systems and they are classified into input and output devices. PIC18F Microcontroller communicates with five I/O Ports, including PORTA through PORTB. Each I/O port is associated with the special function registers (SFR) to setup different function. LEDs, LCD, 7-Segment LED Displays are the common output devices and switches and Keypad Matrixes are the common input devices in PIC18.

8 x LED (Light-Emitting Diode)

LEDs are the simplest displays. Common Cathode and Common Anode are two different ways of configuring the LEDs. In common cathode configuration, cathodes are grounded and anodes are connected to Power Supply. The common cathode and common anode configurations are shown in figures 5(a) and 5(b), respectively. Figure 6 shows screen shot of the LED to display AA(hex) 10101010 in binary.

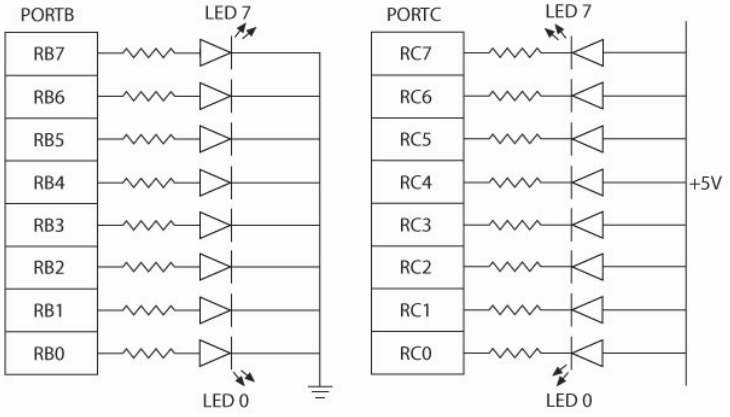

Figure 5-a: Common Cathode Figure 5-b: Common Anode

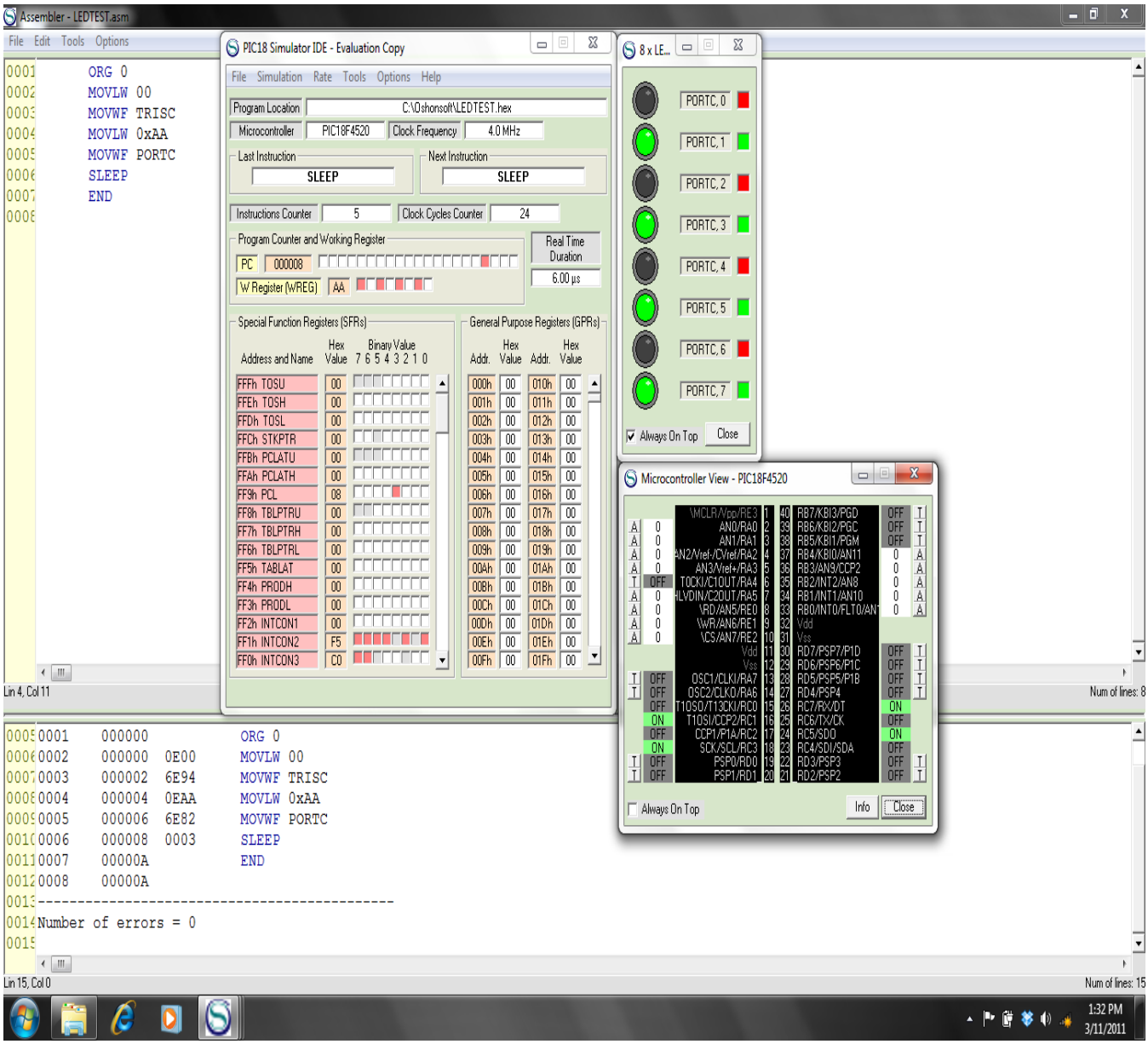

Figure 6: The simulation result of a LED Displayer

LCD (Liquid Cristal Display)

LCD represents ASCII characters. It is varied from 1 to 4 lines and at most represents 80 characters. It has a display Data Registers (ASCII characters) which has its own address that communicates with its location on the line and stores data in 8-Bit character code. Figure 7 illustrates how registers and power are connected to the LCD with driver HD44780 and PIC18F452/4520 Microcontroller.

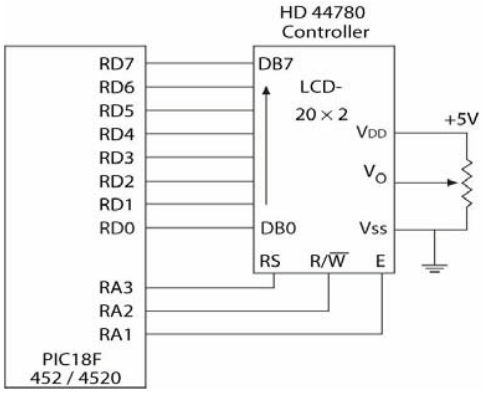

Figure 7: LCD with HD44780 driver and PIC18F452/4520

Figure 8 shows the simulation result of interfacing a 2 line x 16 characters LCD module on PIC18 simulator. The Basic program is designed to display is designed to display Analog Input AN0 Value and the data on AN0/AN1 line in the microcontroller [9].

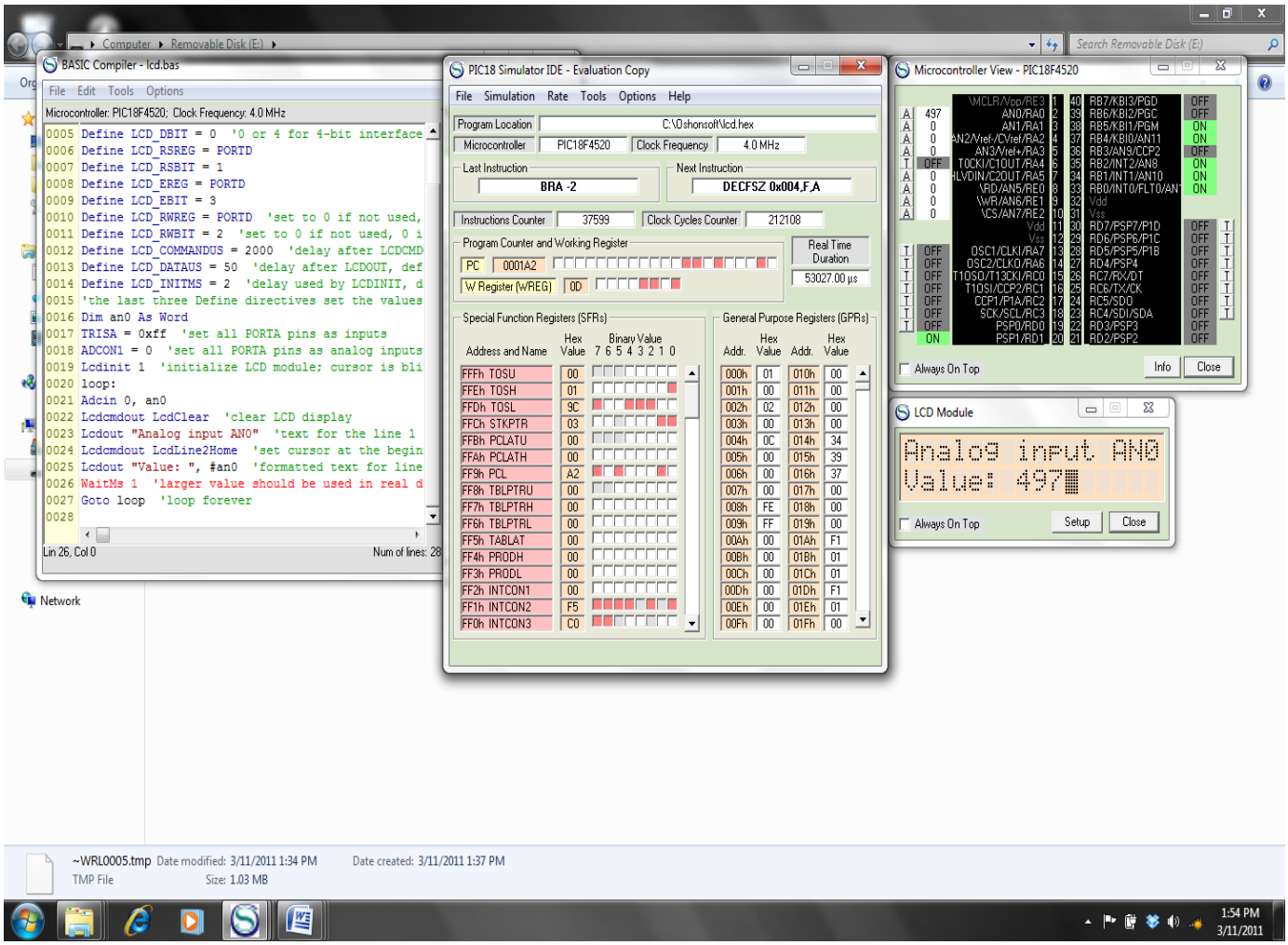

Figure 8: LCD Simulation result

7-Segment LED Displays Panel

Seven-Segment LED is another type of output module of I/O Port. It is a group of 7 LEDs (segments) physically built up in the form of number 8 and a decimal point as illustrated in figure 9. It is used to show decimal number 0 through 9 and alphabets A through F.

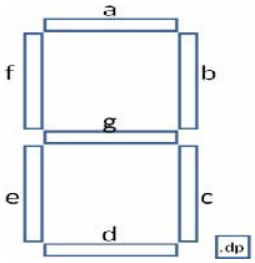

Figure 9: 7-Segment LEDs panel

The Basic program to simulate a seven segment display is shown below and the corresponding simulation result is shown in Figure 10 [9]. The basic program displays numbers from 0 to 99 on the two 7-segment LED displays with parallel connection and two enable lines using TMR0 interrupt multiplexing procedure. The basic file was generated using integrated Basic compiler. The hex file was generated using integrated assembler.

```
Dim digit As Byte 'input variable for GETMASK subroutine
Dim digit1 As Byte 'current high digit
Dim digit2 As Byte 'current low digit
Dim mask As Byte 'output variable from GETMASK subroutine
Dim mask1 As Byte 'current high digit mask
Dim mask2 As Byte 'current low digit mask
Dim i As Byte
Dim phase As Bit
Symbol d1enable = PORTC.0 'enable line for higher 7-segment display
Symbol d2enable = PORTC.1 'enable line for lower 7-segment display
TRISB = %00000000 'set PORTB pins as outputs
TRISC.0 = 0 'set RC0 pin as output
TRISC.1 = 0 'set RC1 pin as output
d1enable = False
d2enable = False
mask1 = 0
mask2 = 0
phase = 0
INTCON.T0IE = 1 'enable Timer0 interrupts
INTCON.GIE = 1 'enable all un-masked interrupts
OPTION_REG.T0CS = 0 'set Timer0 clock source to internal instruction
cycle clock
loop:
```

```
For i = 0 To 99
digit1 = i / 10 'get current high digit
digit2 = i Mod 10 'get current low digit
TMR0 = 0 'reset Timer0 to prevent its interrupt before both masks are
determined
digit = digit1
Gosub getmask 'get mask for high digit
mask1 = mask
digit = digit2
Gosub getmask 'get mask for low digit
mask2 = mask
Gosub show1 'display new mask
Gosub show2 'display new mask
WaitUs 500 'delay interval suitable for simulation
'use large delay for the real device, say WAITMS 500
Next i
Goto loop
End
On Interrupt 'Timer0 interrupt routine
'continuously switch between high and low digit displays
If phase = 0 Then
phase = 1
Gosub show1
Else
phase = 0
Gosub show2
Endif
INTCON.T0IF = 0 'enable new TMR0 interrupts
Resume
getmask: 'get appropriate 7-segment mask for input digit
mask = LookUp(0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f), digit
Return
show1: 'show high digit on its display
d2enable = False
PORTB = mask1
d1enable = True
Return
show2: 'show low digit on its display
d1enable = False
PORTB = mask2
d2enable = True
Return
```

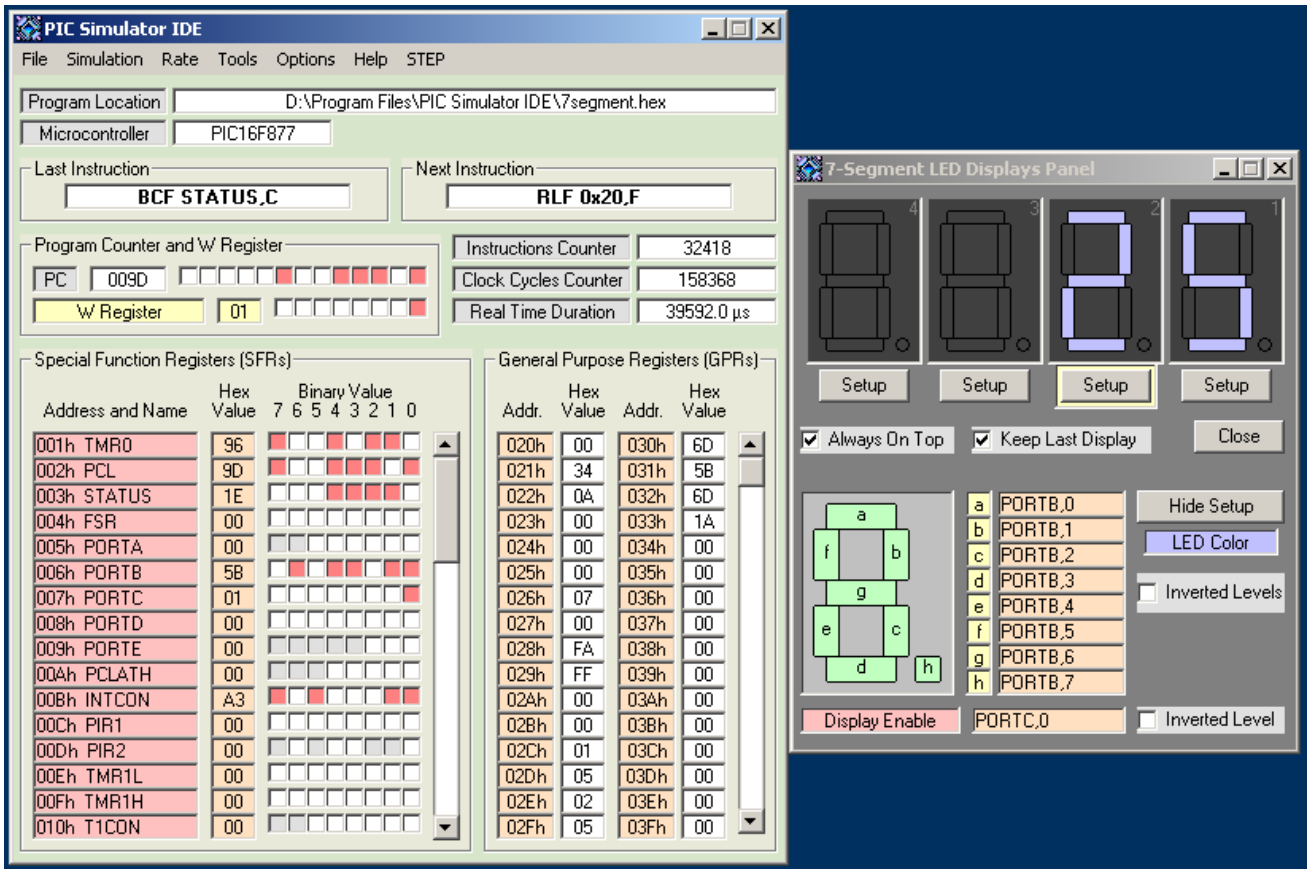

Figure 10 - The Seven Segment LCD simulation result

Summary and Conclusions

The sample modules presented above are user friendly and performed satisfactorily under various input conditions. These and other modules helped the students to understand the concepts in more detail. The students were able to compare their theoretical calculation (machine code) with the machine code generate by the simulator. They were also able to observe various register and memory by using single stepping. Simulator was also helpful to explain difficult concepts such as interfacing LCD, Multiplexing in Seven Segment Displays, Timers and A/D conversion to students. The simulators were also used in an online course and were well liked the students. The PIC Simulator IDE is powerful application that supplies PIC developers with user-friendly graphical development environment for Windows with integrated simulator (emulator), Basic compiler, assembler, disassembler and debugger. PIC Simulator IDE currently supports the PIC 12, PIC 16 and PIC 18 microcontrollers from the Microchip. The student version of the simulator is approximately $30.00 and the department license is approximately $200.00. The EDISM 51 simulator is for 8051 microcontroller and it is available free of charge. These simulators can be used in conjunction with other teaching aids to enhance student learning in various courses and will provide a truly modern environment in which students and faculty members can study engineering, technology, and sciences at a level of detail.

Acknowledgement

This work was funded in part by a grant from the NSF-HBCU-UP/RISC grant. We are thankful to the NSF for providing us with this help.

References

1. Swain, N. K., Korrapati, R., Anderson, J. A. (1999) “Revitalizing Undergraduate Engineering, Technology, and Science Education Through Virtual Instrumentation”, NI Week Conference, Austin, TX..
2. Elaine L., Mack, Lynn G. (2001), “Developing and Implementing an Integrated Problem-based Engineering Technology Curriculum in an American Technical College System” Community College Journal of Research and Practice, Vol. 25, No. 5-6, pp. 425-439.
3. Buniyamin, N, Mohamad, Z., 2000 “Engineering Curriculum Development: Balancing Employer Needs and National Interest--A Case Study” – Retrieved from ERIC database.
4. Kellie, Andrew C., And Others. (1984), “Experience with Computer-Assisted Instruction in Engineering Technology”, Engineering Education, Vol. 74, No. 8, pp712-715.
5. URL: http://www.EDSIM51.com
6. Brey, Barry B. “Applying PIC18 Microcontrollers Architecture, Programming, and Interfacing Using C and Assembly”, Pearson Education, Inc. 2008.
7. Katzen, Sid. “The Quintessential PIC Microcontroller”, 1st edition. Springer-Verlag, 2000. http://padabum.com/data/.pdf
8. http://www.microchip.com
9. http://www.oshonosoft.com