USB Learning Module

by

Praveen Manaprath Mathews

Dissertation submitted in partial fulfillment of
the requirements for the
Bachelor of Engineering (Hons)
(Electrical and Electronic Engineering)

JUNE 2008

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

USB Learning Module

by

Praveen Manaprath Mathews

A project dissertation submitted to the
Electrical and Electronic Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL AND ELECTRONIC ENGINEERING)

Approved by,

[illegible]

DR. YAP VOOI VOON

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
JUNE 2008

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

PRAVEEN MANAPRATH MATHEWS

ABSTRACT

In Universiti Teknologi Petronas (UTP), it is essential for Electrical and Electronic engineering students to grasp the basis of C programming. Quite often, programming is taught through lectures and the lab exercises are more computer science related rather than Electrical and Electronic. Thus the main objective of this project is to produce course materials to aid in the teaching of programming in UTP. The main feature of these course materials will be a PIC microcontroller board and several interface modules. The PIC microcontroller board communicates with the PC via the USB port. Example worksheets are also produced to facilitate the teaching and learning process.

ACKNOWLEDGEMENT

I would like to thank my supervisor, Dr. Yap Vooi Voon for guiding me throughout the entire FYP programme. Thank you for guiding me as well as being patient, kind and helpful throughout the entire two semesters. To my co-supervisor, Mr. Patrick Sebastian, who also stepped in to help out with this project. To Mr. Evan Dudzik also for his advise and suggestions on USB interfacing.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background of Study

The main objective of this project is to produce course materials to aid in the teaching of C programming in Universiti Teknologi Petronas (UTP). It aims to train students on programming skills and problem solving from an engineering perspective. The device will accept simple instructions from students to execute tasks on application boards. This will also instill an engineering method of thinking while enhancing thinking skills for future endavours. As the ever-growing technology era continues, people need to have a basic understanding of technology and its methodologies to cope with the growing advancements.

1.2 Problem Statement of Study

i) With advancements in technology, people must be able to understand the fundamentals of programming.

ii) A suitable learning module is needed to teach students about computer skills and problem solving skills.

1.3 Objective of Study

i) To construct and test a USB interfacing circuit.

ii) To utilize the learning module for a programming course.

iii) To produce sample worksheet to facilitate learning and teaching

1.4 Scope of Study

This project covers a study of alternative teaching methods for teaching programming skills from an engineering perspective.

This project also covers the area of studying different USB interface circuits. This area is important as it provides a fast and easy connection that is recognized worldwide. The project also covers interfacing a user interface, using LOGO and C programming, with the module.

CHAPTER 2

LITERATURE REVIEW

Introduction

With new advancements in connectivity, the popular serial interface is phasing out and being replaced with other alternatives. The most distinct and successful of these is the Universal Serial Bus.

In this section, the literature review is divided into two parts for better understanding, USB and LOGO and C programming:

USB

The first version of USB, the USB 1.0 was created in 1996. Soon after, USB 1.1 was created in September 1998. Finally the current USB 2.0 was created on April 2000, it managed to support up to 48Megabits per second, which was 40 times faster than the speed of USB 1.1[5].

The Universal Serial Bus is a fast, flexible and ideal interface for connecting devices to computers. It was designed to be easy for users with no configurations required on both hardware and software.

USB Development

USB peripheral designing involves getting the peripheral running and developing the PC software to communicate with the peripheral. An assembler or compiler is necessary to create firmware inside the device's chip. A device driver is also needed for recognition by the computer. A monitor program, or protocol analyzer is also used to help in developing firmware and for debugging purposes[4].

USB Enumeration

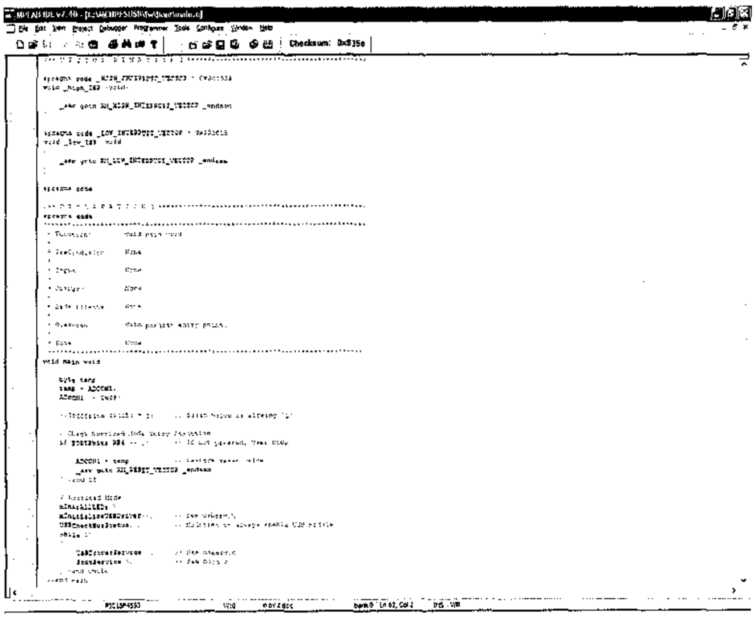
To determine the specific capabilities of each device, the peripheral and the host go through a series of data exchanges upon attachment that inform the operating system of the device and the maximum speed the two can communicate; low, full, or high. This is the first step in the enumeration process; a common process on Windows systems through which all peripherals are registered.

Each device on the USB is identified by a Vendor ID and a Product ID; VID & PID. VID numbers are managed by USB-IF and each number is licensed to a specific vendor; no two VID numbers are used by more than one vendor. On the other hand, the PID is used by the vendor to distinguish vendor products and is assigned without regard of other vendor PIDs. These numbers together constitute a unique key that identifies devices on the bus which the operating system uses during communication[3].

Every USB peripheral is required to implement one or more device descriptors that include information about the device class (type) and its capabilities; Endpoints and communication configurations. The host will request descriptors during the enumeration process, which it will use to register the device with the operating system using the VID and PID, load the appropriate drivers, and set the rules of communication[6].

USB Bootloading

A bootloader acts as a small program to load software for a system to start-up. It resides as the base of the system and allows other programs to boot on it. In the context of USB bootloading, the bootloader must be present on the system before the data is transferred to the device through the USB. Once the system is on bootloader mode, then only will the computer acknowledge the system and thus allow data transfer. When the system is then switched on, the bootloader must be initiated, then only will the data be read and executed. Data can then be efficiently erased and re-programmed on the system.


File Edit View Project Debugger Programmer Tools Configure Window Help
Checksum: 0x835e

```
#pragma code _HIGH_INTERRUPT_VECTOR = 0x[illegible]
void _high_ISR (void)

    _asm goto RM_HIGH_INTERRUPT_VECTOR _endasm


#pragma code _LOW_INTERRUPT_VECTOR = 0x[illegible]
void _low_ISR (void)

    _asm goto RM_LOW_INTERRUPT_VECTOR _endasm


#pragma code

[illegible]
#pragma code
[illegible]
 * Function:        void main(void)
 *
 * PreCondition:    None
 *
 * Input:           None
 *
 * Output:          None
 *
 * Side Effects:    None
 *
 * Overview:        Main program entry point.
 *
 * Note:            None
[illegible]
void main(void)

    byte temp;
    temp = ADCON1;
    ADCON1 |= 0x0F;

    [illegible]

    [illegible]
    if(PORTBbits.RB4 == 1)          [illegible]

        ADCON1 = temp;              [illegible]
        _asm goto RM_RESET_VECTOR _endasm
    [illegible]

    [illegible]
    mInitAllLEDs();
    mInitializeUSBDriver();         [illegible]
    USBCheckBusStatus();            [illegible]
    while(1)

        USBDriverService();         [illegible]
        BootService();              [illegible]
    [illegible]
[illegible]
```

LOGO Programming

The Logo Programming Language, a dialect of Lisp, was designed initially by a team from MIT headed by Wallace Feurzeig in 1967 as a tool for learning. It was designed to tutor children to encourage different thinking styles in where they can think mechanically or 'like a computer'. It is accessible to novices, including young children, and also supports complex explorations and sophisticated projects by experienced users. A Logo word is a string of characters. A Logo list is an ordered collection of words and or lists. Logo's data structures, words and lists are closely parallel to words, phrases, and sentences that make up spoken and written language[2]. For example, the code:

```
FORWARD 100
```

will move the module 100 steps forward

LOGO As A Teaching Tool

The design of the LOGO environment as a whole is strongly influenced by certain general ideas of which three are particularly relevant to work with young minds: Procedurization, anthropomorphization and debugging[7]. Procedurization describes creating an environment where sets of procedures are used which when knit together will help the person understand better. This will channel real-world procedural knowledge into mastering the computer and vice versa. Anthromopomorphization is described as ascribing human characteristics to non--human things. Debugging describes the process of making things work by hypothesizing, testing, revising, etc.
Ideas from computer science like naming, procedurization, and debugging become intermixed with anthropomorphic thinking to become tools in problem-solving situations. The interface between LOGO and the user can be changed by the teacher who knows LOGO, to better suit the user or task assigned. The teacher need not be a computer programmer even.

C Programming

C is a general-purpose, block structured, procedural, imperative computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system. It has since spread to many other platforms. Its design goals were for it to be compiled using a relatively straightforward compiler, provide low-level access to memory, provide language constructs that map efficiently to machine instructions, and require minimal run-time support. Today it is the most widely used programming language for both software and hardware. Thus it is essential that every electrical and electronic engineer be more than familiar with this language.

Teaching C Programming

Being the most widely used programming language, it is essential for every engineer to be familiar with the language. However some are still unclear and distant with it due to the lack of proper education and exposure of the topic at the beginning stages. It has been noted that traditional programming has been taught as the professors learnt it, via syntax, through the vehicle of a single language. With this students are bogged down in the specifics of the chosen form, that they see programming as "fighting the compiler" [8]. Other researchers found that programming should be taught through redundancy and instilling fundamental concepts first before heading on to complex programs [9]. It has also been found that the main problems with teaching beginners especially undergraduates, are that it is difficult for them to grasp the concept of arrays, functions, pointers and also the basic design and flow of the program [10]. Thus the overall layout and flow of a program must be taught first. This can be done through simple programs which students can play around with through trial and error and learn accordingly by viewing the outcomes on the monitor or application boards.

Summary

From the USB reviews, it states and reviews the basic USB functions and limitations which will aid in manipulating the USB for the purpose of this project. Through enumeration and development, the software side of the USB is defined; this helps in explaining the entire process flow of when the USB is plugged into a computer. USB bootloading is a key criterion for this project as the learning module will require programs to be downloaded onto it.

From the LOGO and C programming reviews, a basic knowledge of the languages is grasped and understood. More importantly, their effects as a teaching tool are deeply observed. In accordance with the objective of the project, methods of teaching these languages are also looked into with greater detail.

CHAPTER 3

METHODOLOGY

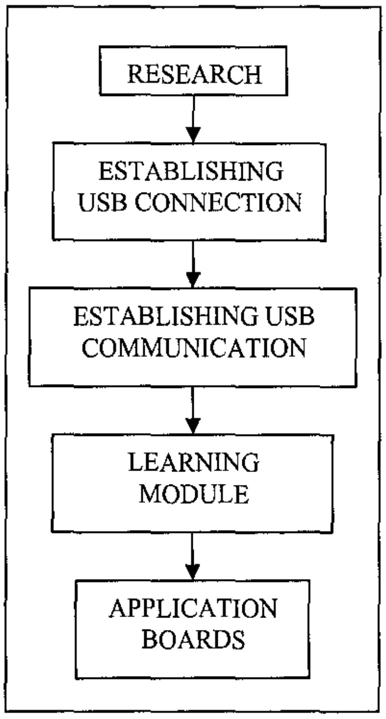


Figure 3.1 Methodology Flow Chart

This final year project will consist of various stages towards the goal of completion:

Research: The first stage was venturing into the nature of USB itself. Extensive research and study was conducted on the USB capabilities and functions as well as searching for alternative USB interface circuits. Each circuit was studied in detail and tested. The best circuit was then chosen and constructed.

Establishing USB connection: As the next stage, a simple USB Demo board using the 18F4550 PIC was constructed to communicate with the computer via USB. Hardware components for the board were obtained and assembled. A bootloader file was compiled and burnt onto the 18F4550 PIC. The appropriate driver was found and installed into the PC to recognize the USB Demo board. This stage was to ensure successful communication on a one way level first just to verify the USB functionality.

Establishing USB communication: The device was then configured to communicate two ways with the computer. The device was successfully recognized by the computer. The necessary driver was installed. A simple software program was then obtained, this program sent already existing hex files that were previously compiled on the computer to the USB Demo board. After which upon execution, that program ran on the USB Demo board. Necessary adjustments were made on the microcontroller and software to send information to the computer.

Learning module: The learning module was constructed and via USB, programs were downloaded to it. Originally the USB Demo board, the learning module was constructed on a Printed Circuit Board (PCB). Extensive testing was conducted to ensure reliability.

Application boards: As the last stage, application boards were made to coincide with the learning module to act as input and output devices. The user may observe the outcomes of their programming codes on these application boards. The application boards constructed are simple, durable and appealing. Each application board can be connected one at a time to avoid confusion and unnecessary extra configurations.

Operation Flow

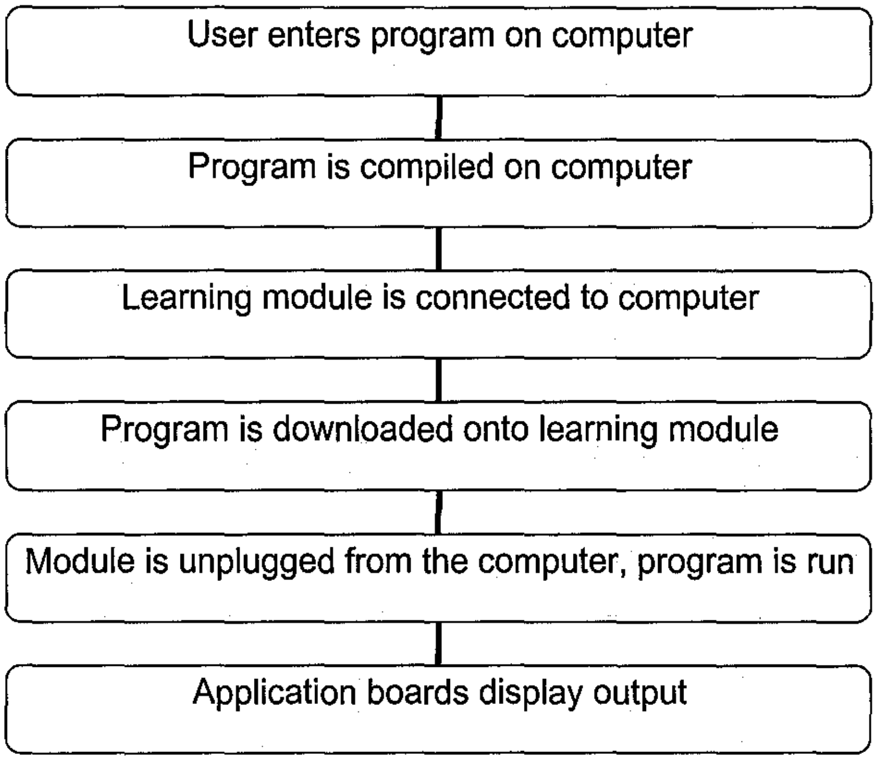

User enters program on computer
Program is compiled on computer
Learning module is connected to computer
Program is downloaded onto learning module
Module is unplugged from the computer, program is run
Application boards display output

Figure 3.2 Operation Flow Chart

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Research

For the first phase of the project, extensive research and study were conducted in two fields:

4.1.1 Programming Skills

Research was done to inquire on the effectiveness of the current programming course in engineering. A survey was done in a class of 10 final year electrical and electronic engineering students from Universiti Teknologi Petronas. The following data was obtained:

1. Students that felt that the C programming course (TAB1013 Structured Programming) provides sufficient knowledge to manipulate a small robot

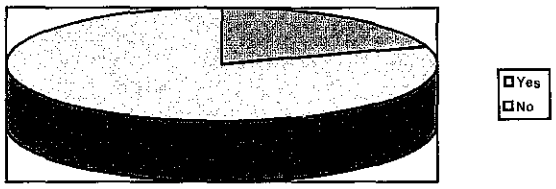


Figure 4.1: Pie Chart 1

2. Relevancy of the C programming course in the degree they are pursuing

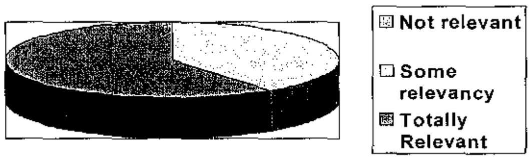


Figure 4.2: Pie Chart 2

3. Students that found it difficult learning C programming through lectures

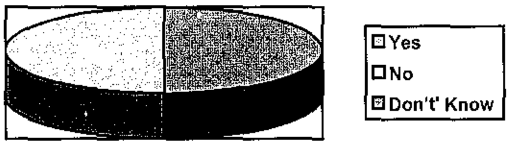


Figure 4.3: Pie Chart 3

4. Students that would rather learn C programming in the form of a lecture

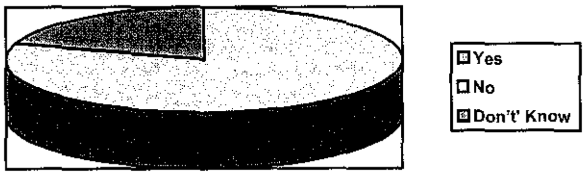


Figure 4.4: Pie Chart 4

4.1.2 USB

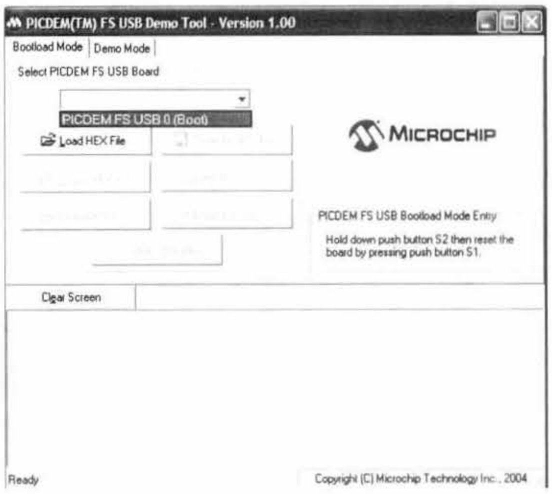
Extensive research was done on the functions and capabilities of the Universal Serial Bus (USB). The PIC chosen for the module was the 18F4550 PIC from MicroChip as it has USB 2.0 capabilities. A demo board was designed and built on a breadboard for testing.

4.2 Establishing USB Connection

The bootloader file was burnt onto the 18F4550 PIC on the demo board using a standard PIC burner. The demo board was then connected via the USB to the computer. Reset button S1 was pressed while holding down reset button S2 (bootloader reset) and bootloader mode was initiated. The computer recognized the demo board and prompted for a driver. Driver is obtained and installed. Device was recognized as PIC18F4550 Family Device. USB connection was successfully established.

4.3 Establishing USB Communication

The MicroChip PICDEM FS USB Tool was obtained from MicroChip. After connecting the demo board to the computer and initiating bootloader mode, the board was recognized as a demo board.

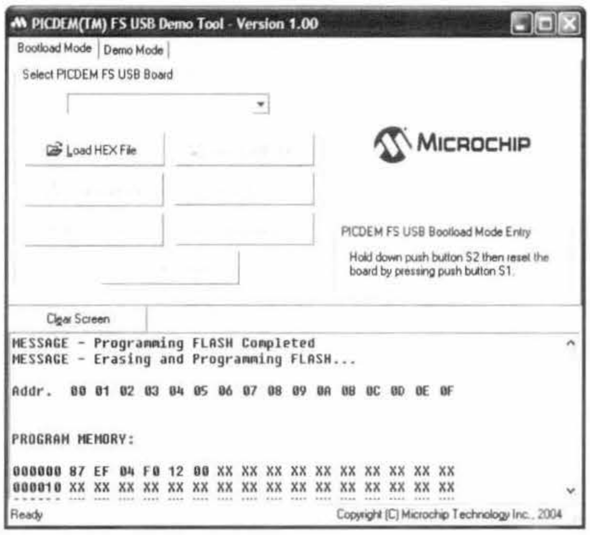

PICDEM(TM) FS USB Demo Tool - Version 1.00
Bootload Mode
Demo Mode
Select PICDEM FS USB Board
PICDEM FS USB 0 (Boot)
Load HEX File
MICROCHIP
Clear Screen
Ready
Copyright (C) Microchip Technology Inc., 2004

PICDEM FS USB Bootload Mode Entry

Hold down push button S2 then reset the board by pressing push button S1.

Figure 4.5: Board recognized by the PICDEM FS USB Tool

A sample program (.hex file) was programmed onto the demo board and the board is executed. Program was successfully downloaded and executed on the demo board. Communication successfully established.


PICDEM(TM) FS USB Demo Tool - Version 1.00
Bootload Mode
Demo Mode
Select PICDEM FS USB Board
Load HEX File
MICROCHIP
Clear Screen

PICDEM FS USB Bootload Mode Entry

Hold down push button S2 then reset the board by pressing push button S1.

```
MESSAGE - Programming FLASH Completed
MESSAGE - Erasing and Programming FLASH...

Addr.  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

PROGRAM MEMORY:

000000 87 EF 04 F0 12 00 XX XX XX XX XX XX XX XX XX XX
000010 XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX XX
```

Ready

Copyright (C) Microchip Technology Inc., 2004

Figure 4.6: PICDEM FS USB Tool programming and executing device

4.4 Learning Module

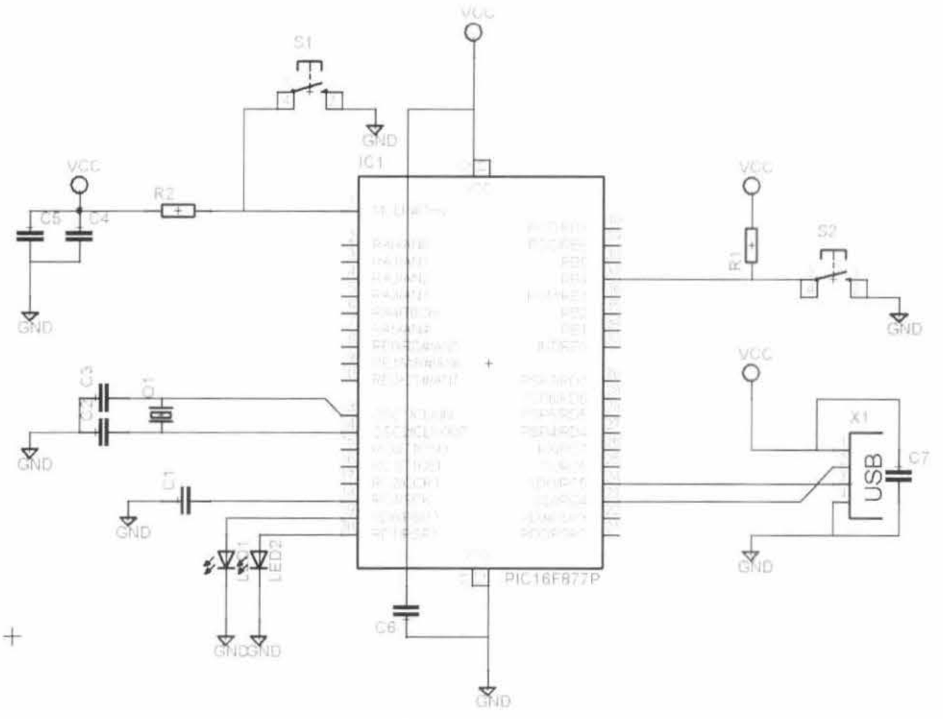
For the fourth phase of the project, the finalized base of the learning module was constructed and tested.

The base of the learning module consists of two segments: the USB interface and the microcontroller. The construction of the learning module base is primarily to verify that programs can be downloaded onto the microcontroller via USB and also to verify the proper programming techniques and definitions needed in each program to allow the microcontroller to function properly.

The learning module was tested on a veroboard and deemed feasible; it was then constructed on a printed circuit board (PCB).

4.4.1 Hardware Design

Below is the new design for the PCB base of the learning module:



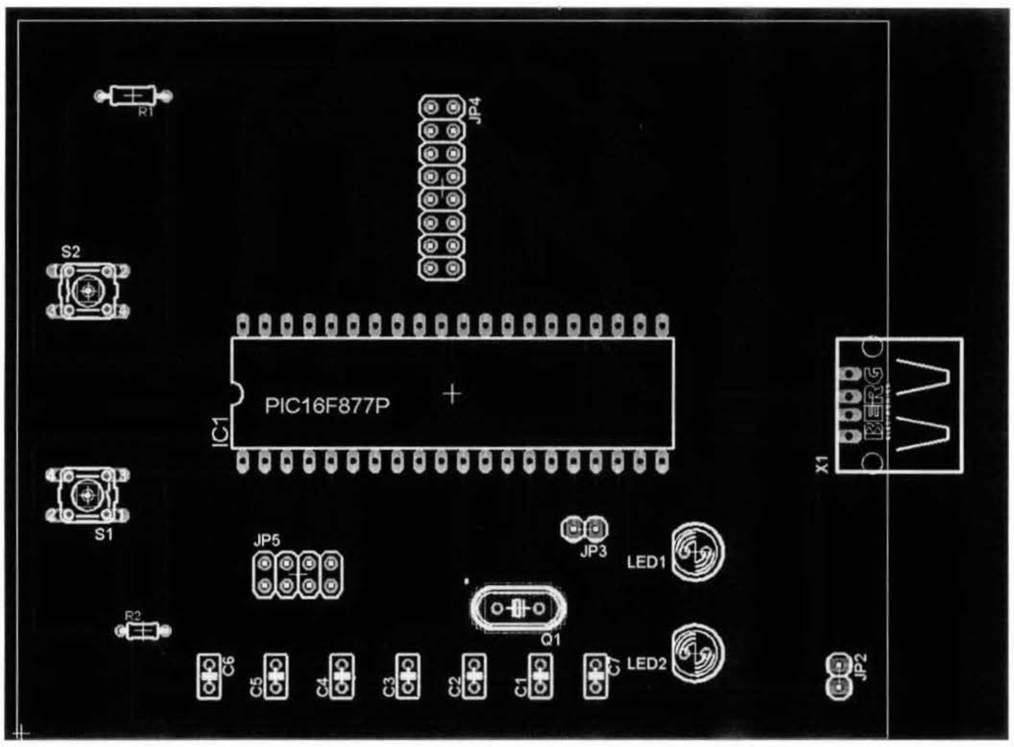


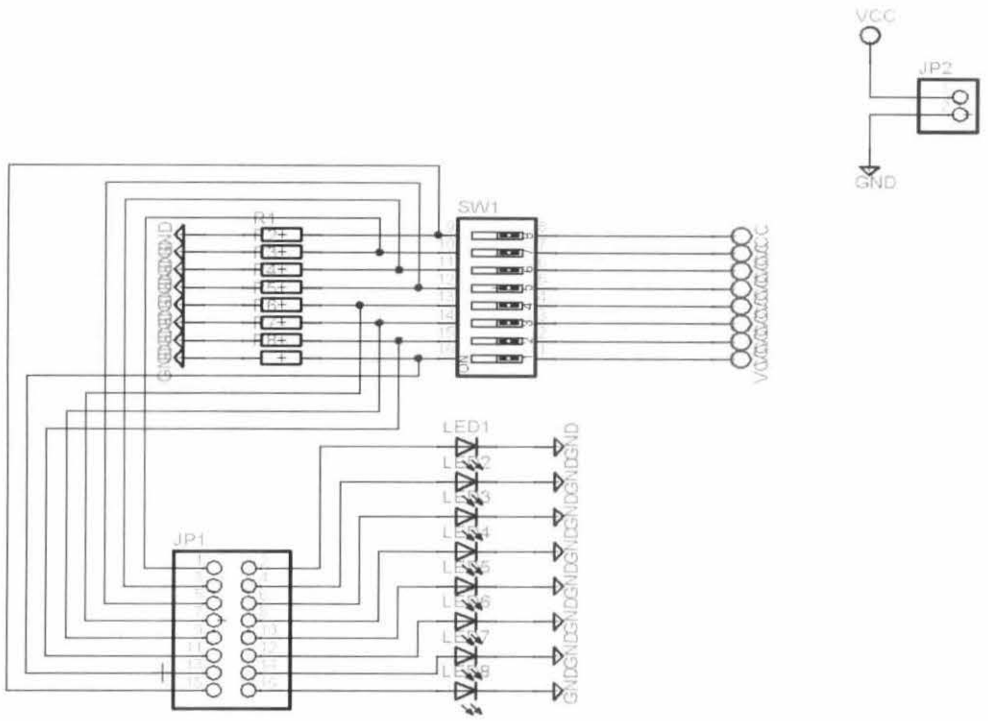
Figure 4.7: Learning Module Base schematic and diagram

4.5 Application Boards

Two application boards were made to be connected to the base of the learning module. The first consists of LEDs and switches and will output lit or flashing LEDs according to the program written. The second board is a two wheeled robot that will move forward, backward, left or right depending on the code produced.

These application boards can only be connected to the learning module one at a time.

Below is the design for the first application board:



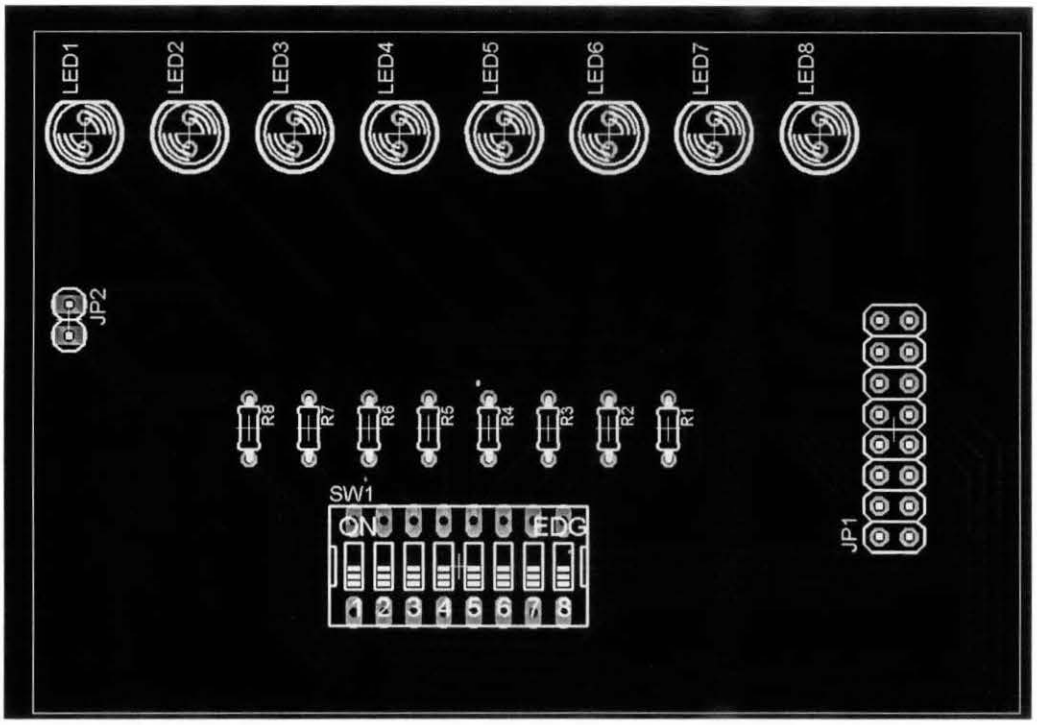


Figure 4.8: First Application Board schematic and diagram

Below is the design for the second application board:

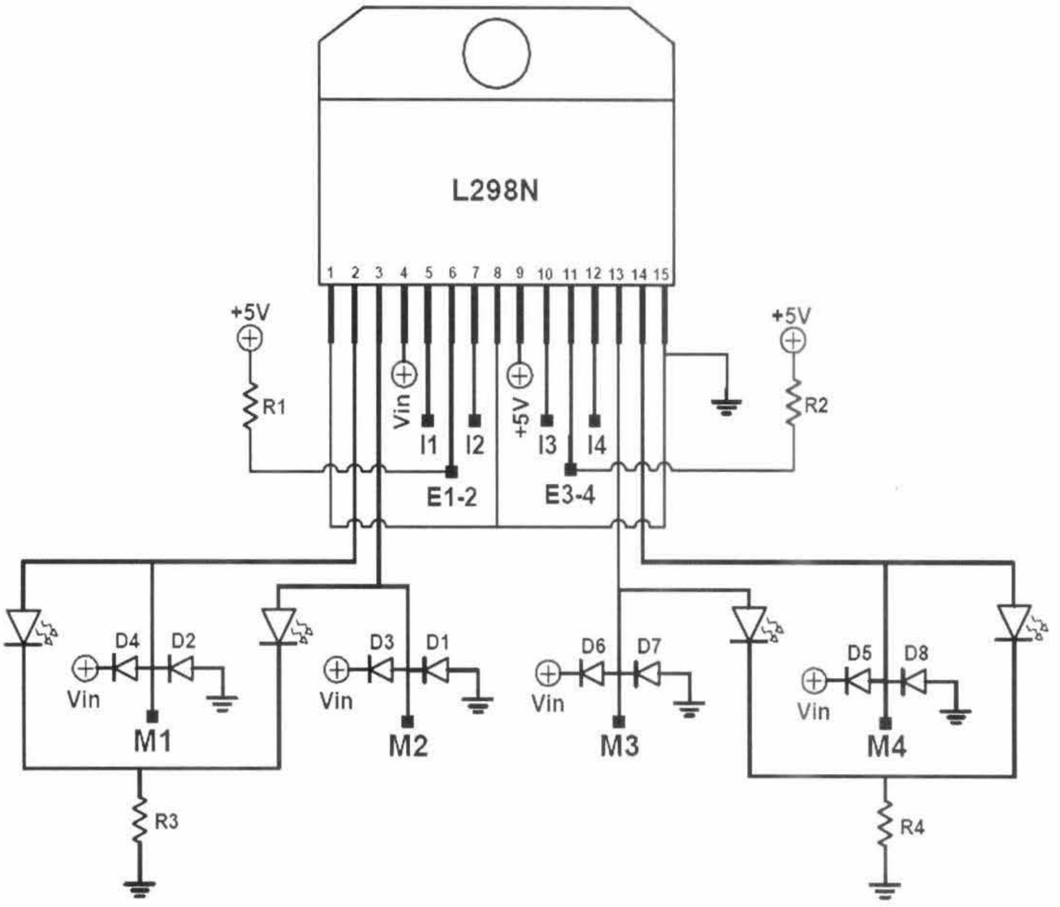


Figure 4.9: Second Application Board schematic

4.6 Results and Discussion

The learning module primarily consists of the USB interface and the microcontroller that runs the module. For this, the 18F4550 PIC from MicroChip was used.

The first application board primarily consists of LEDs and switches. This board will be connected to the learning module. The switches are inputs to the learning module, when the switches are switched on or off, a selected program will run on the learning module and the output LEDs will flash accordingly.

The second application board is a 2 wheeled robot. This board will be connected to the learning module as well. Depending on the code produced, the robot will move forward, backward, left or right.

Another application board will eventually be constructed such as an LCD screen.

4.6.1 Problems Encountered

The major problem encountered was determining the appropriate definitions and functions that needed to be included in the program for proper execution. Since these programs are transferred via the USB, many definitions and functions needed to be included in proper format before the program is able to execute. After much research, all necessary functions were determined.

CHAPTER 5

CONCLUSION AND RECCOMMENDATION

5.1 Conclusion

In conclusion, this project studies different programming languages available for teaching programming skills. Using a learning module, students will be able to acquire programming skills, as well as problem solving skills. The project aims to produce course materials both in hardware and software, to aid in the teaching and learning of C programming.

The project also aims to explore the use of the USB port to control devices. It looks for alternative USB interface circuits. Being a fast, easy and universally recognized standard, USB is a feasible and ideal connection for the purpose of this project and also for many other devices.

To date, the base of the learning module and two application boards have been constructed and function properly. A simple program was successfully downloaded onto the base and executed on the testing board. This coincides with the final stage of the project: 'Application Boards'. The given time dateline will be sufficient for the completion of the project as the methodologies have been planned out already and are deemed feasible.

The next phase carried out will be on constructing an additional application board to be connected to the base. Extensive construction and testing will be necessary.

5.2 Recommendation

For future enhancement of this project, other appealing application boards should be made. A board with an LCD screen can be made to display the output of the program. Users will then input a program to display characters on a 16 by 2 LCD screen.

REFERENCES

[1]. Seymour Papert, *Mindstorms*, Basic Books, 1980, pg 27

[2]. LOGO Foundation:
http://el.media.mit.edu/logo-foundation/logo/index.html
http://el.media.mit.edu/logo-foundation/logo/programming.html

[3]. Jan Axelson, *USB Complete* Second edition, Lakeview Research, 2001, pg 19, pg147

[4]. Don Anderson, *USB Systems Architecture* Second Edition, MindShare Inc, Addison Wesley, 2001, pg 19

[5]. *Universal Serial Bus Implementers Forum [Internet]. Beaverton(OR): c2006. Available from* ***www.usb.org/.***

[6]. Eric Brown, *USB Instrumentation*, 2006, Yale University

[7]. Cynthia J. Solomon, *Teaching Young Children To Program In A LOGO Turtle Computer Culture,* ACM Sigcue Bulletin, July 1978

[8] Fincher S., *What Are We Doing When We Teach Programming?*, Proceedings 1999 Frontiers in Education Conference Nov.1999, San Juan PR, p.p 12a4-1 – 12a4-5.

[9] Jermann W.H., *The Freshman Programming Course: A new direction*, Proceedings 1996 ASEE Annual Conference, June 1996, Washington D.C.

[10]. Dan Budny, Laura Lund, Jeff Vipperman, John L. Patzer II, *Four Steps To Teaching C Programming*, 2002, 32nd ASEE/IEEE Frontiers in Education Conference

APPENDIX I

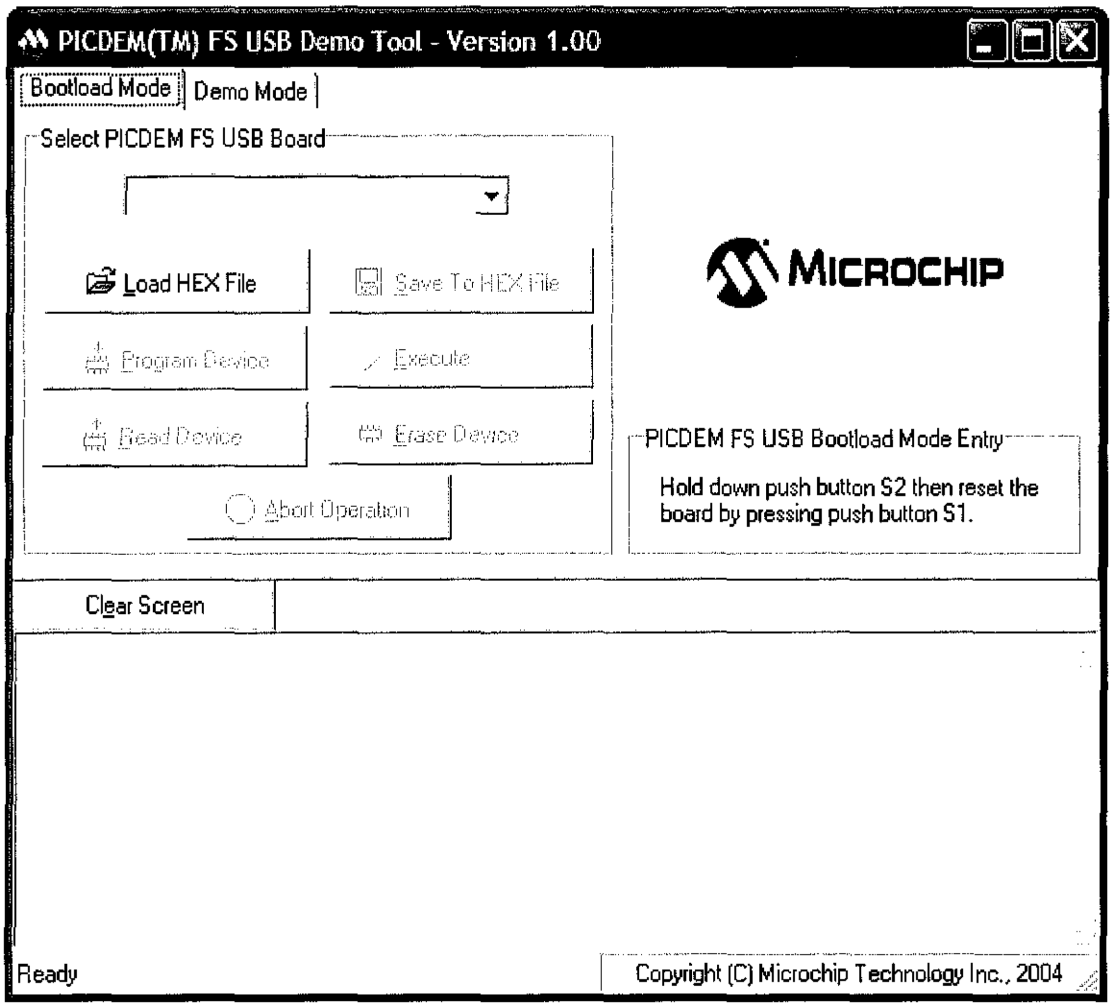
Bootloader main c file

```
/** I N C L U D E S **********************************************************/
#include <p18cxxx.h>
#include "system\typedefs.h"                // Required
#include "system\usb\usb.h"                 // Required
#include "io_cfg.h"                        // Required

#include "system\usb\usb_compile_time_validation.h" // Optional

/** V A R I A B L E S ********************************************************/
#pragma udata

/** P R I V A T E  P R O T O T Y P E S ***************************************/

/** V E C T O R  R E M A P P I N G *******************************************/

#pragma code _HIGH_INTERRUPT_VECTOR = 0x000008
void _high_ISR (void)
{
   _asm goto RM_HIGH_INTERRUPT_VECTOR _endasm
}

#pragma code _LOW_INTERRUPT_VECTOR = 0x000018
void _low_ISR (void)
{
   _asm goto RM_LOW_INTERRUPT_VECTOR _endasm
}

#pragma code

/** D E C L A R A T I O N S **************************************************/
#pragma code
void main(void)
{
   byte temp;
   temp = ADCON1;
   ADCON1 |= 0x0F;

   //TRISBbits.TRISB4 = 1;    // Reset value is already '1'

   //Check Bootload Mode Entry Condition
   if(PORTBbits.RB4 == 1)     // If not pressed, User Mode
   {
      ADCON1 = temp;          // Restore reset value
      _asm goto RM_RESET_VECTOR _endasm
```

```
    }//end if

    //Bootload Mode
    mInitAllLEDs();
    mInitializeUSBDriver();   // See usbdrv.h
    USBCheckBusStatus();      // Modified to always enable USB module
    while(1)
    {
        USBDriverService();   // See usbdrv.c
        BootService();        // See boot.c
    }//end while
}//end main

#pragma code user = RM_RESET_VECTOR

/** EOF main.c ***************************************************************/
```

APPENDIX II

Downloading Software



PICDEM FS USB Bootload Mode Entry

Hold down push button S2 then reset the board by pressing push button S1.

Ready

Copyright (C) Microchip Technology Inc., 2004

APPENDIX III

Student Survey Form

1. Do you feel that after having completed the "TAB1013 Structure Programming" course C that you would be able write C programs to manipulate a small robot?
   Yes No
   Circle the appropriate answer.

If no, please state reasons

2. How would you rank the relevancy of "TAB1013 Structure Programming" in the degree course that you are pursuing?
   1- Not relevant at all
   2- Some relevancy
   3- Totally relevant

3. Do you think you understand C programming through lectures difficult?
   1- Yes
   2- No
   3- Don't know

Circle the appropriate answer.

4. Would you rather learning C programming in the form of a lecture?
   1- Yes
   2- No
   3- Don't know

Circle the appropriate answer.

APPENDIX IV

Demo program 1 (Demo02.c)

```
/** I N C L U D E S **********************************************************/
#include <p18cxxx.h>

/** V A R I A B L E S ********************************************************/
#pragma udata

/** V E C T O R  R E M A P P I N G *******************************************/

extern void _startup (void);        // See c018i.c in C18 compiler dir
#pragma code _RESET_INTERRUPT_VECTOR = 0x000800
void _reset (void)
{
   _asm goto _startup _endasm
}
#pragma code

#pragma code _HIGH_INTERRUPT_VECTOR = 0x000808
void _high_ISR (void)
{
   ;
}

#pragma code _LOW_INTERRUPT_VECTOR = 0x000818
void _low_ISR (void)
{
   ;
}
#pragma code

/** D E C L A R A T I O N S **************************************************/
```

```
#pragma code

/** L E D ***********************************************************/
#define mInitAllLEDs()     LATD &= 0xF0; TRISD &= 0xF0;

#define mLED_1          LATDbits.LATD0
#define mLED_2          LATDbits.LATD1
#define mLED_3          LATBbits.LATB0
#define mLED_4          LATBbits.LATB1
#define mLED_5          LATBbits.LATB2
#define mLED_6          LATBbits.LATB3
#define mLED_7          LATBbits.LATB5
#define mLED_8          LATBbits.LATB6
#define mLED_9          LATBbits.LATB7

#define mLED_1_On()      mLED_1 = 1;
#define mLED_2_On()      mLED_2 = 1;
#define mLED_3_On()      mLED_3 = 1;
#define mLED_4_On()      mLED_4 = 1;
#define mLED_5_On()      mLED_5 = 1;
#define mLED_6_On()      mLED_6 = 1;
#define mLED_7_On()      mLED_7 = 1;
#define mLED_8_On()      mLED_8 = 1;
#define mLED_9_On()      mLED_9 = 1;

#define mLED_1_Off()     mLED_1 = 0;
#define mLED_2_Off()     mLED_2 = 0;
#define mLED_3_Off()     mLED_3 = 0;
#define mLED_4_Off()     mLED_4 = 0;
#define mLED_5_Off()     mLED_5 = 0;
#define mLED_6_Off()     mLED_6 = 0;
#define mLED_7_Off()     mLED_7 = 0;
#define mLED_8_Off()     mLED_8 = 0;
#define mLED_9_Off()     mLED_9 = 0;

#define mLED_1_Toggle()   mLED_1 = !mLED_1;
#define mLED_2_Toggle()   mLED_2 = !mLED_2;
```

```
#define mLED_3_Toggle()    mLED_3 = !mLED_3;
#define mLED_4_Toggle()    mLED_4 = !mLED_4;
#define mLED_5_Toggle()    mLED_5 = !mLED_5;
#define mLED_6_Toggle()    mLED_6 = !mLED_6;
#define mLED_7_Toggle()    mLED_7 = !mLED_7;
#define mLED_8_Toggle()    mLED_8 = !mLED_8;
#define mLED_9_Toggle()    mLED_9 = !mLED_9;

/** S W I T C H *****************************************************/
#define mInitAllSwitches()
TRISBbits.TRISB4=1;TRISBbits.TRISB0=0;TRISBbits.TRISB1=0;TRISBbits.TRISB2=0;TRISBbits.TRI
SB3=0;TRISBbits.TRISB5=0;TRISBbits.TRISB6=0;TRISBbits.TRISB7=0;
#define mInitSwitch2()    TRISBbits.TRISB4=1;
#define mInitSwitch3()
TRISBbits.TRISB5=0;TRISBbits.TRISB0=0;TRISBbits.TRISB1=0;TRISBbits.TRISB2=0;TRISBbits.TRI
SB3=0;TRISBbits.TRISB6=0;TRISBbits.TRISB7=0;
#define sw2           PORTBbits.RB4
#define sw3           PORTBbits.RB5

void main(void)
{
  ADCON1 |= 0x0F;           // Default all pins to digital
  mInitAllSwitches();
  mInitAllLEDs();
  while(1)
  {
    if(sw2 == 0)
    {
      mLED_1_On();
                      mLED_2_On();
                      mLED_3_On();
                      mLED_6_On();
                      mLED_7_On();
                      mLED_8_On();
                      mLED_9_On();


    }
```

```
        else
        {
          mLED_1_Off();
                              mLED_2_Off();
                              mLED_3_Off();
                              mLED_4_Off();
                              mLED_5_Off();
                              mLED_6_Off();
                              mLED_7_Off();
                              mLED_8_Off();
                              mLED_9_Off();

        }//end if else
    }//end while
}//end main

/** EOF Demo02.c *************************************************************/
```

APPENDIX V

Demo program 2 (triall.c)

```
/** I N C L U D E S **********************************************************/
#include <p18cxxx.h>
#include <delays.h>


/** V A R I A B L E S ********************************************************/
#pragma udata

/** P R I V A T E  P R O T O T Y P E S ***************************************/

/** V E C T O R  R E M A P P I N G *******************************************/

extern void _startup (void);       // See c018i.c in your C18 compiler dir
#pragma code _RESET_INTERRUPT_VECTOR = 0x000800
void _reset (void)
{
    _asm goto _startup _endasm
}
#pragma code

#pragma code _HIGH_INTERRUPT_VECTOR = 0x000808
void _high_ISR (void)
{
    ;
}

#pragma code _LOW_INTERRUPT_VECTOR = 0x000818
void _low_ISR (void)
{
    ;
}
#pragma code

/** D E C L A R A T I O N S **************************************************/
#pragma code

/** L E D ********************************************************************/
#define mInitAllOutputs()    LATB &= 0x00; TRISB &= 0x00;

#define mOut_1          LATBbits.LATB5
#define mOut_2          LATBbits.LATB6
#define mOut_3          LATBbits.LATB7
#define mOut_4          LATBbits.LATB3

#define mOut_1_High()       mOut_1 = 1;
```

```
#define mOut_2_High()       mOut_2 = 1;
#define mOut_3_High()       mOut_3 = 1;
#define mOut_4_High()       mOut_4 = 1;

#define mOut_1_Low()      mOut_1 = 0;
#define mOut_2_Low()      mOut_2 = 0;
#define mOut_3_Low()      mOut_3 = 0;
#define mOut_4_Low()      mOut_4 = 0;

#define delay_ms()                          Delay100TCYx(1000)

#define mOut_1_Toggle()   mOut_1 = !mOut_1;
#define mOut_2_Toggle()   mOut_2 = !mOut_2;
#define mOut_3_Toggle()   mOut_3 = !mOut_3;
#define mOut_4_Toggle()   mOut_4 = !mOut_4;

/** S W I T C H *****************************************************/
#define mInitAllInputs()  TRISDbits.TRISD4=1;TRISDbits.TRISD5=1;
#define mInitSwitch2()    TRISDbits.TRISD4=1;
#define mInitSwitch3()    TRISDbits.TRISD5=1;
#define sw2            PORTDbits.RD4
#define sw3            PORTDbits.RD5

void main(void)
{
  ADCON1 |= 0x0F;            // Default all pins to digital
  //mInitAllInputs();
  mInitAllOutputs();
  while(1)
  {

        mOut_3_Low();
        mOut_1_High();
        mOut_2_Low()`;

  }//end while
}//end main

/** EOF Demo02.c ****************************************************************/
```

APPENDIX VI

Sample Worksheet

Moving forward

Upon downloading a program, a robot will execute according to its written program. Did you ever stop to wonder why the robot was moving? Now you will get a chance to look at the code more closely, and using the code, create a Forward, Right, or Left movement with the robot.

In this section, you will learn what each command does in the sample program. You will also learn to move your robot forward, right and left.

Below is a line explanation of the sample program:

Program 1: LED on

```
void main(void)   ◄──────────
{

        mLED_1_On();

        mLED_2_Off();

}
```

Every C program contains a **main()** task. **main()** is usually at the beginning.

```
void main(void)
{

            mLED_1_On();

            mLED_2_Off();

}
```

void main forms a structure that starts with an open brace and ends with a closing brace.

```
void main(void)
{

            mLED_1_On();

            mLED_2_Off();

}
```

This line is a statement that will turn on an LED (LED 1).

Program 2: Motor on

```
void main(void)
{

            mOut_1_Low();
            mOut_2_High();
            mOut_3_Low();

}
```

This line is a statement that will turn on the appropriate motor (motor 2)