Report
on

REII 321
PRACTICAL 2

LCD Interfacing

SCHOOL FOR ELECTRICAL, ELECTRONIC & COMPUTER ENGINEERING

Mr. Henri Marais

by

CR van Zyl 21492204

11 August 2013

NORTH-WEST UNIVERSITY
YUNIBESITI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT
POTCHEFSTROOMKAMPUS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. GOAL

The goal of the practicum is to interface a LCD screen in 4-bit mode with the PIC 16F1937 microcontroller.

After successful initialization of the screen elementary programming technique shall be implemented to ultimately write a string of characters to the LCD screen.

Proper use of timing diagrams shall ensure a successful interface between the LCD screen and the PIC microcontroller.

2. BACKGROUND

When developing software for an embedded system, the programmer is seldom in a position to physically see if the code is doing what it is supposed to do. LCD interfacing is an exception to this generalization; the programmer can test code on-the-fly by means of visible gestures.

The LCD screen can be described as a form of visible RAM where its current state is observed on the screen itself.

Proper implementation of a microcontroller-to-LCD based system has countless applications and is therefore a good exercise for students studying towards a degree in Computer and Electronic Engineering.

3. HARDWARE DESIGN

The first step in successful LCD interfacing is hardware design. Since the LCD screen has adjustable contrast, it is possible to write working code and still be unable to observe any output.

With the above in mind, a trimpot was inserted between the V0 pin of the LCD screen and V_{DD}. When a acceptable contrast was observed, the resistance on the trimpot was measured and a resistor close to the trimpot measurement was inserted (in this case 330Ω) [1].

Next the LCD was wired to the PIC microcontroller. Figure 1 shows the complete schematic that was used to interface the LCD:

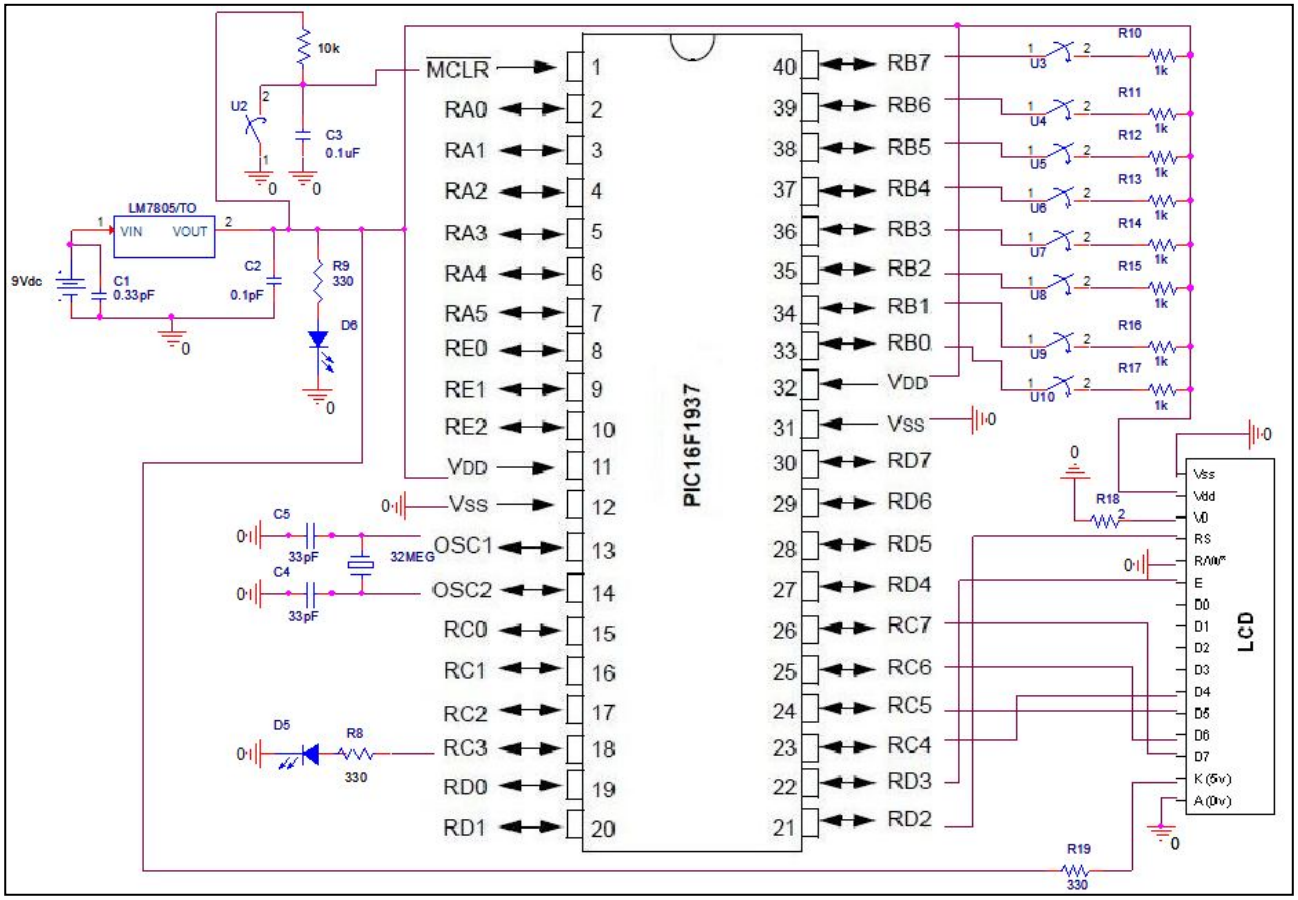

Figure 1: Complete LCD interface schematic

Notes:

The 8-pin DIP switch connected to Port B is used to shift the LCD screen, move the cursor and to reset the screen.

The LED on pin RC3 indicates whether or not the one of the digital inputs are currently active (active low).

The R/W* pin on the LCD screen is grounded since there will only be written to the screen and not read from it.

Only the upper four bits (D4 - D7) are connected to the PIC, hence 4-bit operating mode.

Figure 2 shows the final breadboard layout of the practical:

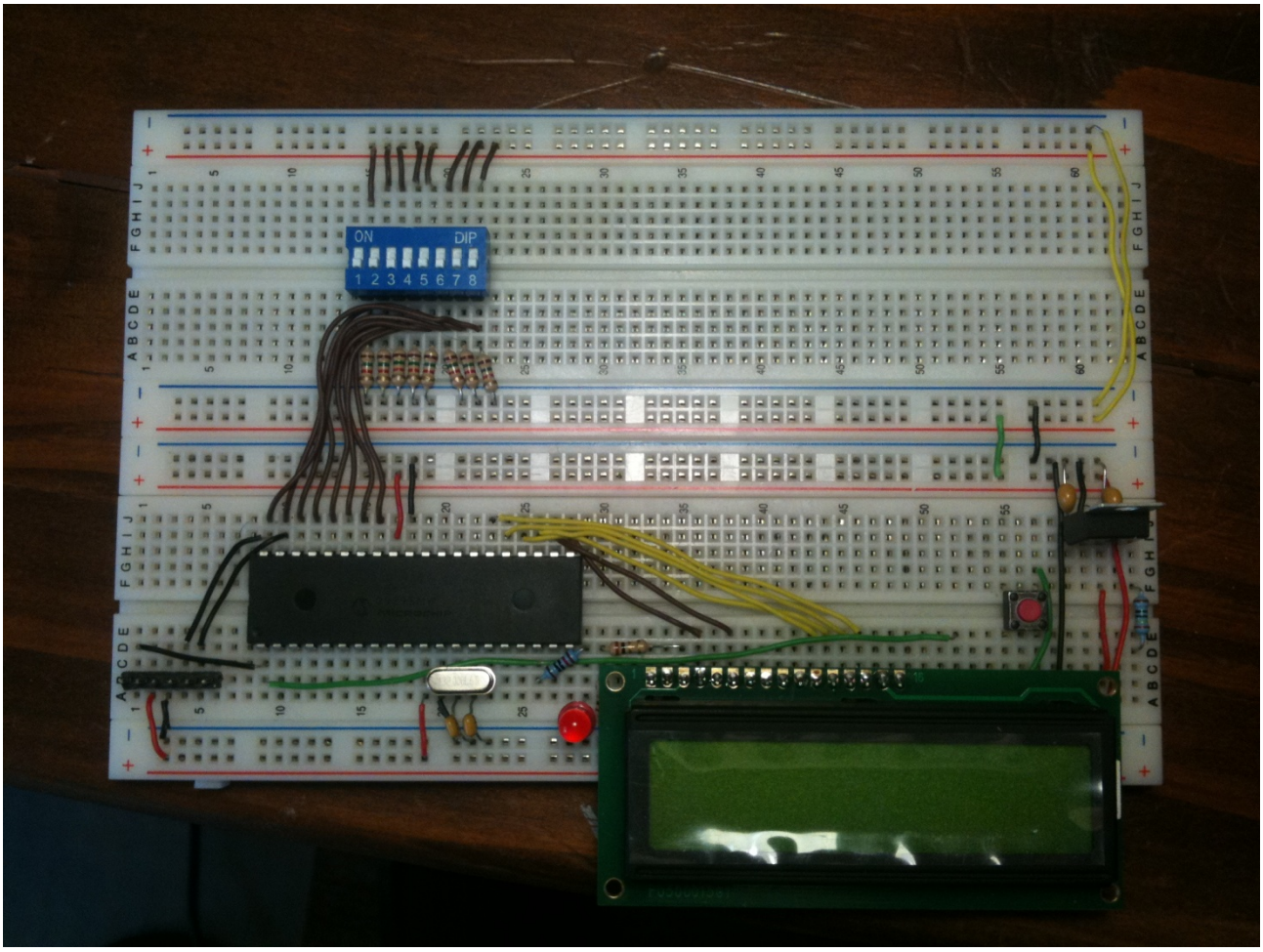

Figure 2: Final breadboard layout of the practical

4. SOFTWARE DESIGN

The first step in successful LCD interfacing is to properly initialize the screen by means of hexadecimal codes found in the datasheet [2].

Table 1 shows the codes used to initialize the screen along with brief descriptions of each:

Description	Code
Function Set: 2 line display, 4-bit mode	0x28
Display ON/OFF: Display on, cursor on	0x0E
Entry Mode Set: Entry mode enabled, advance cursor, shift screen right	0x06
Clear display: Display cleared	0x01

Table 1: Initialization Codes

Next, frequently used code was written into functions with argueably the most important being the command_LCD and character_LCD functions.

Refer to Appendix A for the source code, a discussion of important functions continues below.

command_LCD allows us to write codes to the LCD module and will typically be implemented during initialization. The function accepts a variable of type char which is then AND'ed with 0xF0 and set equal to PORTC. Next, the E-pin of the LCD is strobed from high to low before shifting the char left by 4 bits and, once again AND'ing the shifted result with 0xF0, the result is then set equal to PORTC. In order to complete the command_LCD function, the E-pin needs to be strobed again, a time delay is also necessary between command calls. The students chose a time delay of 250ms to create a typewriter-like effect when characters are written to the LCD module.

character_LCD is a direct implementation of the command_LCD function. It works by accepting a char* (char pointer) and then pulling the RS-pin of the LCD high (indicating that display data is to be written) before writing the string character-by-character. The C-language does not have a string variable, hence the use of char* (char pointer) to write individual characters to the LCD module to form strings.

All the shift functions work by writing relevant control bytes to the LCD module when requested. The students implemented this extended functionality by means of digital I/O. When, for instance, switch number 8 on the DIP is toggled from low to high, the shift (int) function is called with an integer value equal to 0. The last mentioned function then writes a control nibble periodically to shift the display right by one space at a time.

The remaining shift functions work in the exact same way, with each writing a different code depending on what we want to achieve with them. All shifting functions are implemented via a switch since the only difference between them is the hexadecimal code to be written.

Note:
All delays used in the software was implemented with Hi-Tech C's built in delay routines [3]. It was (in this case) not necessary to use a interrupt driven delay routine, the processor's only task is to drive the LCD so it can afford to simply waste time during delays.

5. TESTING

Throughout development various aspects of the design was constantly tested to isolate faults that may arise due to malfunctioning hardware as well as logical (programming) errors.

Firstly the power supply circuit was tested with a multimeter before commencing development.

Next, all pins that are to be connected to the LCD module was pulled high in software, and tested with a multimeter to ensure that they are in fact able to be pulled into a high state.

In order to ensure that the digital inputs are in working order, a LED was toggled.

Once working code had been written, output was directly observed on the screen.

Figure 3 shows the LCD module displaying the strings 'REII 321' and '21492204 & 21807965' on the first and second lines respectively:

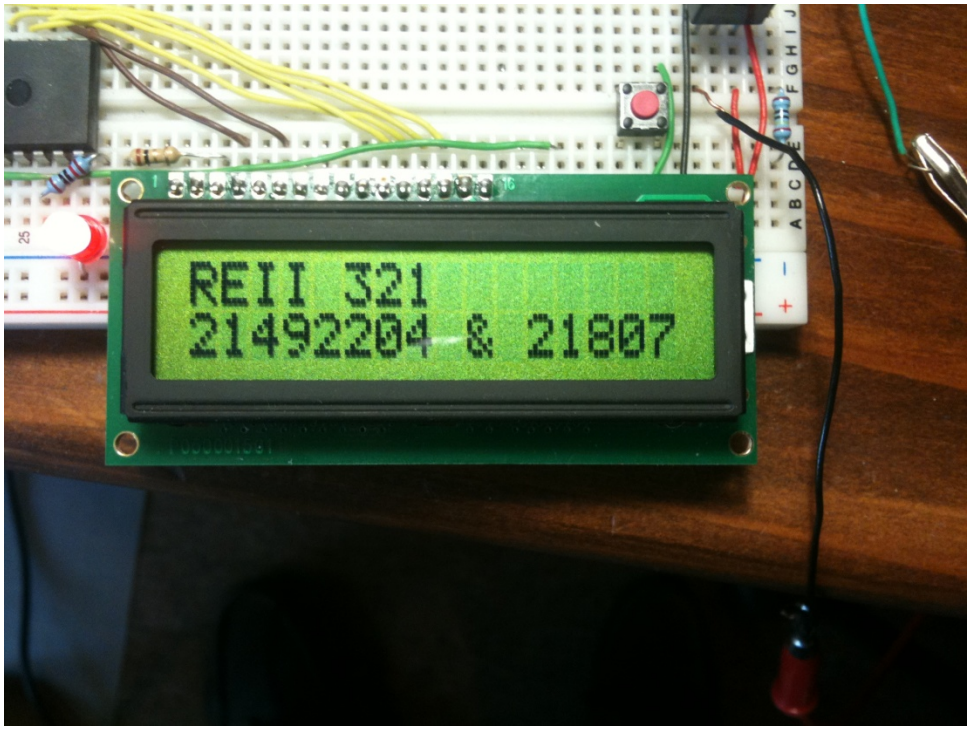

Figure 3: Graphic representation of the working LCD interface

6. CONCLUSION

The practicum was completed successfully and the specifications have been met. We were able to interface the LCD screen in 4-bit mode and eventually wrote a string of characters to the screen.

The use of timing diagrams in the design of embedded systems is understood and shall be implemented in future designs with relative ease.

7. REFERENCES

[1] "Interfacing A PIC To LCD." Internet: http://www.pyroelectro.com/tutorials/pic_lcd/index.html [Sept. 14, 2011].

[2] Shenzhen TOPWAY Technology, "LMB162ABC LCD Module User Manual," LCD datasheet, 2005.

[3] Microchip, "HI-TECH C® for PIC10/12/16 User's Guide," Compiler user guide, 2010.

8. APPENDIX A

```
//***Lcd.h***
#define LCD_RS      RD2                 //Register select; pinn RD2
#define LCD_E       RD3                 //Enable; pinn RD3
#define LED_0       RC3                 //To indicate if a digital input is active
(active low)

#ifndef _XTAL_FREQ                      //Initialize delay calls; __delay_ms(x)
                                        and __delay_us(x)
#define _XTAL_FREQ 32000000             //Oscillator frequency
#endif

void initialize_LCD ();
void command_LCD (unsigned char);
void second_Line ();
void clear_LCD ();
void shift (int);
void character_LCD (const char *);
void toggle_E();
```

```
//***Lcd.c***
#include <htc.h>
#include "Lcd.h"

    void initialize_LCD ()
    {
    LCD_RS = 0;                         //Write control bytes

    toggle_E();
    __delay_ms(10);

    command_LCD (0x28);                 //Function Set: 2 line display, 4-bit
    command_LCD (0x0E);                 //Display on, cursor ON
    command_LCD (0x06);                 //Entry mode enable advance cursor, shift
                                        screen right
    command_LCD (0x01);                 //Clear display and reset cursor
    }

    void command_LCD (unsigned char b)
    {
    PORTC = b & 0xF0;                   //Write first nibble, move, write second
    toggle_E();
    PORTC = (b << 4) & 0xF0;
    toggle_E();
    __delay_ms(250);                    //Typewriter effect when writing strings
    }
```

```
void clear_LCD ()
{
LCD_RS = 0;

command_LCD (0x1);
__delay_ms(2);
}

void second_Line ()
{
LCD_RS = 0;                    //Write control bytes
command_LCD (0xC0);            //Moves cursor to first place on the second
                               line
}
```

```
void shift(int command)
{
        LCD_RS = 0;

        switch(command)
        {
                case 0:
                command_LCD (0x1C);
                break;

                case 1:
                command_LCD (0x1B);
                break;

                case 2:
                command_LCD (0x10);
                break;

                case 3:
                command_LCD (0x14);
                break;
        }
}
```

```
void character_LCD (const char *c)          //Write characters
{
LCD_RS = 1;

while(*c)
command_LCD (*c++);
}

void toggle_E()
{
LCD_E = 1;
__delay_us(1);
LCD_E = 0;
}
```

```
//***Main.c***

#include <htc.h>
#include "Lcd.h"

void main()
{
        ANSELD = 0x00;                  //Set PORTD to digital I/O
        TRISD = 0x00;                   //Set PORTD output
        ANSELB = 0x00;                  //Set PORTB to digital I/O
        TRISB = 0xFF;                   //PORTB digital input
        TRISC = 0x00;                   //Set PORTC output

         LED_0 = 0;

        initialize_LCD();

        character_LCD ("REII 321");

        __delay_ms(1000);

        second_Line();

        character_LCD ("21492204 & 21807965");
```

```
        while(1)
        {
        if(RB0 == 1)
        {
        shift(0);                       //Shift screen right
        }

        if(RB1 == 1)
        {
        shift(2);                       //Shift cursor left
        }

        if(RB2 == 1)
        clear_LCD();                    //Clears display and sets the cursor to position 1

        if(RB6 == 1)
        {
        shift(3);                       //Shift cursor right
        }

        if(RB7 == 1)
        {
        shift(1);                       //Shift screen left
        }

        LED_0 = 1;                      //Command indicator LED
        }
}
```