Lab 6: Handshaking and LCD Control

Contents

[Goal and Background Information 2](#_Toc85489619)

[Implementation 4](#_Toc85489620)

[Testing and Verification 8](#_Toc85489621)

[Table 1: Measured Serial Communication Timings 8](#_Toc85489622)

[Figure 1 8](#_Toc85489623)

[Figure 2 8](#_Toc85489624)

[Figure 3 9](#_Toc85489625)

[Figure 4 9](#_Toc85489626)

[Figure 5: DDRAM Address Read Timing 9](#_Toc85489627)

[Conclusion 9](#_Toc85489628)

[Listings 9](#_Toc85489629)

[Listing 1: LCD\_putc() 9](#_Toc85489630)

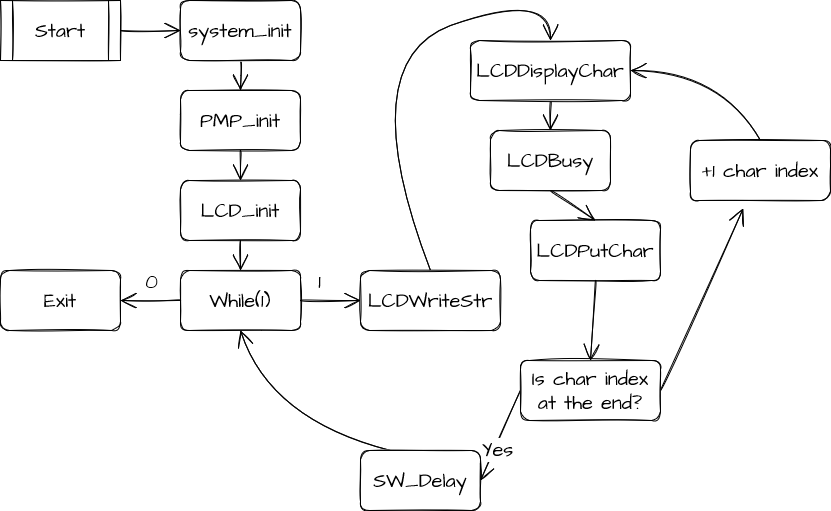
[Listing 2: LCDInit 10](#_Toc85489631)

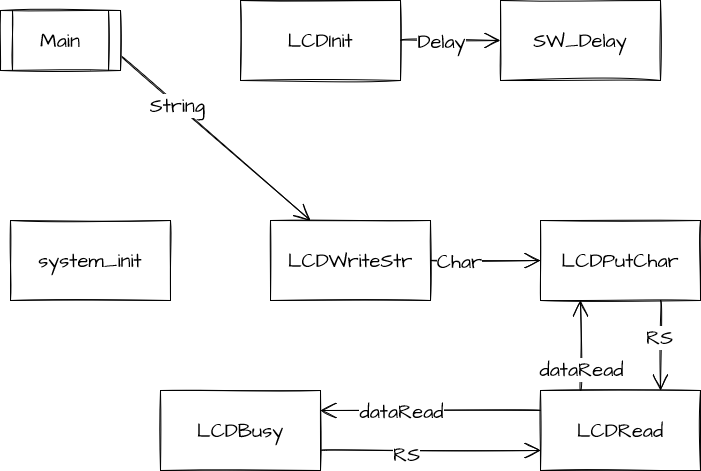
# Goal and Background Information

The goal of this lab is to implement a parallel, asynchronous handshake between the PIC32 and the LCD control module to write characters to the LCD screen. This is done by utilizing several user written functions and the Peripheral Master Port or PMP for short. To control the LCD controller using the PMP library functions like PMPSetAddress and PMPMasterWrite will be used inside of user defined functions like LCDPutChar. The PMP is intended for devices including LCDs, other microcontrollers, external memory devices, other generic communications peripherals.

The functions written for this lab include LCDInit, LCDPutChar, LCDWriteStr, LCDDelay, LCDBusy, LCDWriteChar, and LCDRead. The functions LCDInit, LCDWriteStr, and LCDDelay will be the only functions called in the while(1) loop but will contain the other functions mentioned to complete the necessary tasks to put characters on the LCD screen. For example, when a character array is passed to the LCDWriteStr function the LCDWriteStr functions will call LCDPutChar for every character, besides the NULL character, to write it to the LCD screen. The reason that the PIC32 is interfacing with the LCD control module is because it is much easer than sending direct signals to the LCD. This is because the main purpose of the LCD controller chip is to control the LCD.

All handshaking done in this lab will be primarily handled by the PMP and does not require “bit-banging” to send instructions to the LCD control module. The PMP is set with predefined times needed to successfully handshake with the LCD controller within the PMP initialization code of the lab. This is seen in the way that the PMP is initialized. The command PMP\_WAIT\_BEG\_X, , PMP\_WAIT\_MID\_X, and PMP\_WAIT\_END\_X is used to control timing of the read-write and enable signals in the PMP-LCD controller handshake. It is recommended to use the generous PMP wait settings of 4, 15, and 4 and not the 1, 2, and 1 wait settings from the lab handout. Using the PMP negates the need to bit-bang data because the PMP handles the delays necessary for a successful handshake using predefined timing variables in the PMPInit function.

 **Data Flow Diagram**

**Control Flow Diagram**

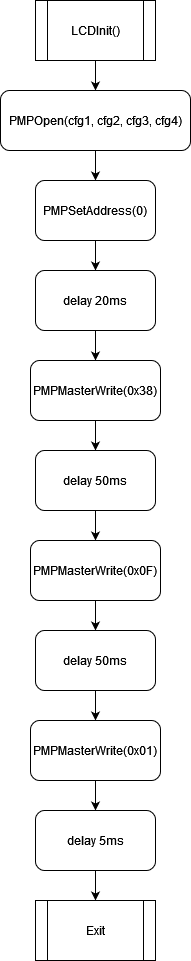
# Implementation

The implementation of lab 6 did not go as planned. There were several issues with properly including only the header files needed. Another issue was that when the busy function was orginaly written it called itself and caused many timing errors. These were solved with time and with help from the TAs and Dr. J. All of the functions used to interact with the LCD were written in the LCDlib.c file and the prototypes are in the LCDlib.h file. The main function that was needed to be written was the LCD\_putc() function. This is the function that handles control characters and the actual business of writing the character to the LCD screen using writeLCD.

The first function written was LCDInit and is what sets up the LCD for use within the program. The CFD can be seen in *CFD 1: LCDInit*. After enshureing that LCDInit was correct the next function written was readLCD. This function reads the data from the selected register and returns it to the program. The function writeLCD was the next function created and took two araguments, RS and char. The RS (register select) would select what register the PMP was writing to and char was the value that the PMP writes. The function busyLCD polls the LCD controller for the busy flag to be deserted. LCD\_putc takes an argument of a character, sees if the character is a control character and processes it if so. If not the function checks to see if the carriage is outside the visible window and sets it to the next line if necessary and writes the character input to the LCD. LCD\_puts takes a string and used a while loop to write the string to the LCD using LCD\_putc.

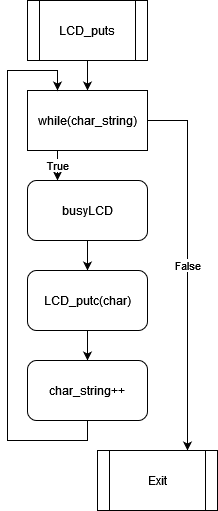
The CFDs of the functions in LCDlib.c are listed below besides the PMP initialization and LCDDelay. The PMP initialization in included in the LCDInit code listing and the LCDDelay is the exact same code as the hardware delay but with a different name and therefore not shown or listed.

CFD 1: LCDInit CDF 2: readLCD CFD 3: writeLCD

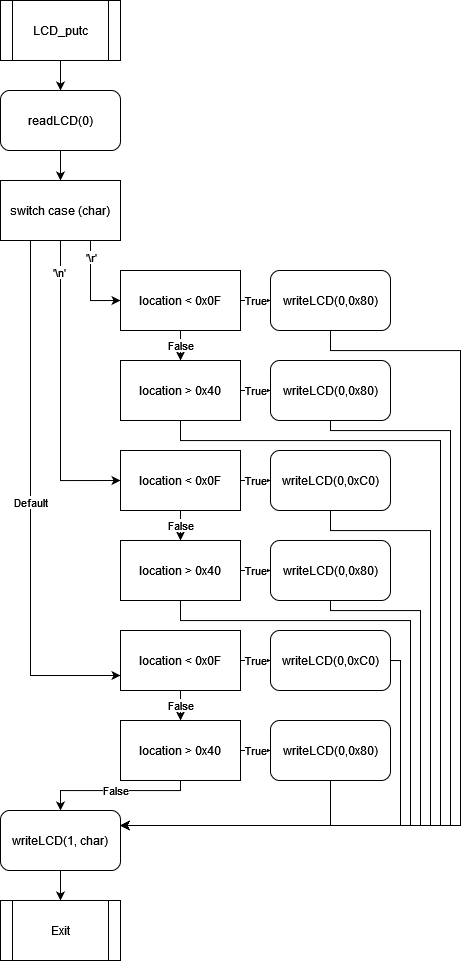
Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text

Description automatically generatedCFD 4: busyLCD CFD 5: LCD\_puts

CFD 6: LCD\_putc



# Testing and Verification

Two forms of verification were used to test and verify that the code written for this lab was working correctly. The first is visual inspection. This consisted of programing the PIC32 and then visually examining the LCD screen to see if the correct sequence of characters showed up as well as being in the correct places. The LCD should be cleared in the setup of the PIC32 and then the question “Does Dr J prefer PIC32 or FPGA??” should display. After five seconds, the answer to the question should display. The answer should read “Answer: \116\145\151\164\150\145\162\041" (The answer is Neither!).

The second form of verification is by using a logic analyzer to examine the waveforms of the communications between the PMP and the LCD control module. This would allow for the timings of the different signals to ensure that the program was not violating any setup times. This is important because the LCD controller would not be able to function properly if signals were being sent to it at a higher rate than it could read. The timed waveforms can be found *Table 1: Measured Serial Communication Timings.* Another interesting thing to note is that in *Figure 5: DDRAM Address Read Timing* two enable signals can be seen. This is the double read sequence used to ensure that the ‘pipelined’ information coming from the LCD controller is correct. This is shown from the fact that the register select line is set to a 0 and the read line is set to a 1 before the enable rises twice.

## Table 1: Measured Serial Communication Timings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Enable Cycle Time (t\_c) | Enable High Pulse Width (t\_w) | RS, RW Set Up Time (t\_su) | RS, RW Hold Time (T\_h) |
| Minimum (ns) | 500 | 220 | 40 | 10 |
| Maximum | NA | NA | NA | NA |
| Measured (ns) | 2700 | 1600 | 600 | 2400 |
| Refrence | Figure 1 | Figure 2 | Figure 3 | Figure 4 |

## Figure 1

Graphical user interface

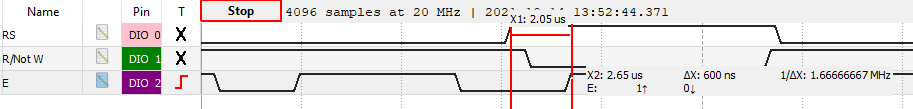
Description automatically generated with medium confidence

## Figure 2

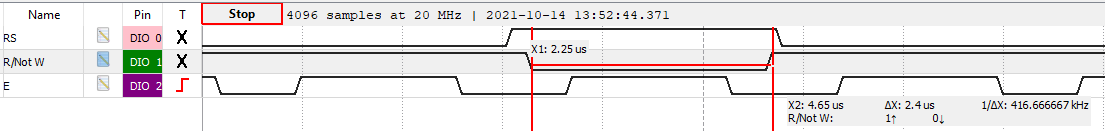
Diagram

Description automatically generated

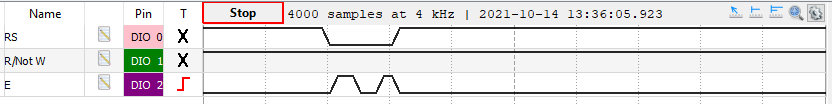
## Figure 3



## Figure 4



## Figure 5: DDRAM Address Read Timing



# Conclusion

When calculating the rate at which the LCD control module can receive characters it is important to note that the longest delay is 500 ns for the enable cycle time. This limits the LCD control module to one character every 500 ns or about 2\*106 symbols per second rate. Using this it can also be calculated how long it would take the LCD control module to completely erase the LCD screen and re write to the visible display area. The operations needed to erase the LCD screen is one. Using the command writeLCD(0,0x01) the LCD controller will erase all characters from the LCD screen and return the carriage to the top left of the screen. Because this is only one character this will take 500 ns. Rewriting the visible display area takes 33 write operations. 16 write operations for the 1st line, a write operation to move the carriage to the beginning of the next line and then 16 more write operations to fill the bottom row of the visible display.

Using the PMP to communicate to the LCD control module is only slightly less efficient once you have the code necessary running. The PMP is clocked using the peripheral bus clock and is slower than the speed the CPU is clocked at. This means that the PMP has a higher latency to respond to input than the CPU. When considering the larger picture, the PMP more efficient for whoever needs to program the PIC32. There is less code needed (and therefore less to go wrong) when the PMP is in the development stage. This is because the initialization of the PMP presets the timings the PMP will use to communicate, and every command will not need a specified delay command every time the LCD controller needs one to ensure the CPU does not send signals at a faster rate can be received.

# Listings

## Listing 1: LCD\_putc()

void LCD\_putc(char c) //Makes decisions about control char

{

int location = readLCD(0);

location = location & 0x7F; //Read the location of DDRAM

switch(c){

case '\r':

if(location < 0x0F)

{

writeLCD(0,0x80);

}

else if(location > 0x40)

{

writeLCD(0, 0xC0);

}

break;

case '\n':

//Do control thing

if(location < 0x0F)

{

writeLCD(0,0xC0);

}

else if(location > 0x40)

{

writeLCD(0, 0x80);

}

break;

default:

if(location > 0x4F)

{

writeLCD(0, 0x80);

}

if((0x0F < location)&&(location < 0x40))

{

writeLCD(0, 0xC0);

//writeLCD(1, 'X');

}

writeLCD(1, c);

}

}

## Listing 2: LCDInit

//PMP init

void LCDInit(void){

int cfg1 = PMP\_ON|PMP\_READ\_WRITE\_EN|PMP\_READ\_POL\_HI|PMP\_WRITE\_POL\_HI;

int cfg2 = PMP\_DATA\_BUS\_8 | PMP\_MODE\_MASTER1 | PMP\_WAIT\_BEG\_4 | PMP\_WAIT\_MID\_15 | PMP\_WAIT\_END\_4;

int cfg3 = PMP\_PEN\_0; // only PMA0 enabled

int cfg4 = PMP\_INT\_OFF; // no interrupts used

mPMPOpen(cfg1, cfg2, cfg3, cfg4);

//END PMP init!

PMPSetAddress(0); // Set LCD RS control

LCDDelay(20);

PMPMasterWrite(0x38); //8 bit data, 2 lines

LCDDelay(50);

PMPMasterWrite(0x0f); //display on, cursor on, blink cursor on

LCDDelay(50);

PMPMasterWrite(0x01); //Clear display

LCDDelay(5);

}