1300 Henley Court
Pullman, WA 99163
509.334.6306
www.store.digilentinc.com

Unit 3: Parallel I/O and Handshaking for LCD Control

Revised March 10, 2017
This manual applies to Unit 3

1 Introduction

Throughout Unit 3 and Labs 3a and 3b, you will learn how to interface a PIC32 processor to a character liquid crystal display (LCD) using 8-bit parallel data lines and handshaking using bit-banging techniques. You will learn to translate timing diagrams provided by target equipment data sheets into timed sequences of program instructions. You will also learn how to use ASCII encoded data and control to generate a stable and easily readable display of real-time data.

You will experiment with the mechanisms required to synchronize asynchronous systems using hardware and software handshaking, as well as explicit and implicit handshaking. You will build a project that consists of multiple C files. We will also look into using variables in software macros.

2 Objectives

1. Identify the PIC32 processor pins used for the LCD interface.
2. Write a library of functions, providing the general capability to display strings of text, characters, and numbers at specified positions on the LCD display. All of these functions have strict timing requirements.
    a. Initialize the LCD using the recommended sequence of instructions to write specific codes to the LCD at specified intervals.
    b. Write function(s) that write ASCII control data to the Display Data Ram (DDRAM).
        i. Translate the ASCII carriage return (CR) and line feed (LF) control characters into a sequence of instructions for LCD control.
        ii. Position the cursor where the next character is to be displayed.
    c. Write function(s) that write ASCII character data to the Display Data Ram (DDRAM).
        i. Write a function that will write a single ASCII encoded character to the display.
        ii. Write a function that will write a string (array) of ASCII encoded characters to the display.
3. Display static and dynamic text on the LCD.
4. Verify that the timing of the handshaking meets the specified minimums.
5. Understand handshaking requirements for interfacing two systems operating at differing clock speeds.

3 Basic Knowledge

1. How to interpret control flow and data flow diagrams.
2. How to interpret a schematic diagram and electric circuits.

3. How to write a computer program using the C language.
4. How to launch a Microchip MPLAB X project.

4 Equipment List

4.1 Hardware

1. Basys MX3 trainer board
    a. The 2X16 LCD included on the Basys MX3 processor board can display 16 characters on each of its two lines. The display can be more easily viewed if the LCD backlight is switched on using the slide switch SW9.
2. Micro USB cable
3. Workstation computer running Windows 10 or higher, MAC OS, or Linux

In addition, we suggest the following instruments:

4. Analog Discovery 2 Logic Analyzer
    a. Note: Fig. B.1 through Fig. B.4 in Appendix B shows an Analog discovery 2 connected directly to the LCD data pins and the three handshaking and control pins: RS, R/W, and EN. This custom modification was made to enable testing during software development. It is not required to make this modification in order to complete Lab 3a and Lab 3b. This is further explained in Appendix B.

4.2 Software

The following programs must be installed on your development work station:

1. Microchip MPLAB X® v3.35 or higher
2. PLIB Peripheral Library
3. XC32 Cross Compiler
4. WaveForms 2015 (if using the Analog Discovery 2)

5 Project Takeaways

1. How to use control flow diagrams to generate program code.
2. How to use software delay to write control commands at specified intervals.
3. How to use software code to generate minimum setup and hold times.
4. How to implement one form of handshaking.
5. How to read and interpret timing software modeling diagrams.

6 Fundamental Concepts

6.1 Communications

Digital communication is an electronic transfer of information between two electronic devices. Parallel communication is a method of conveying multiple binary data bits simultaneously. It contrasts with serial communication, which is covered in Unit 4, and conveys only a single bit at a time. Both parallel and serial communications may have additional signals, such as a clock signal to pace the flow of data, a signal to control the direction of data flow, and handshaking signals.

7 Background Information

7.1 Handshaking: What and Why

Handshaking refers to the mechanism used to facilitate the transfer of data between two systems that may be operating at different clock frequencies. In simple terms, it involves a set of messages or signals exchanged between two devices that control when data is transferred. The sender is the device that is the source of the information, and the receiver is the device intended to be the destination of the information. For effective exchange of information, the sender needs to know first if the receiver is in a condition in which it is able to receive information, and second, when the information has been received. The receiver needs to know when the information is ready to be sent and when the sender determines that the receiver has acquired the information.

The exact implementation of the handshaking process comes in many forms, but it may be classified as either explicit or implicit. Explicit handshaking uses dedicated hardware control signals to indicate the impending action from the sender and readiness of the receiving device, as shown in Fig. 7.1. Each of the two handshaking signals has two possible states. The basic concepts are that the receiving device must indicate when it is ready to receive data and when it is done. The sending device must tell the receiving device when new data is ready and when the data is no longer available.

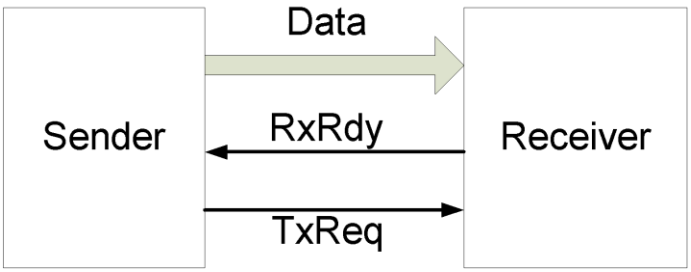

Figure 7.1. Interconnection for 4-phase handshaking.

The timing diagram shown in Fig. 7.2 illustrates the sequence of events required for the sender to transfer data to the receiver using a four-phase handshaking protocol. The shaded area on the data signal indicates intervals when the data level is permitted to change from high to low or low to high. At time “A,” the receiver asserts the RxRdy signal high to indicate to the sender that it is ready to accept data. When the sender has information to send to the receiver, it first asserts the data signals high or low (indicated at time “B”) and then asserts the TxReq signal high at time “C” to indicate the data is now available to be read. The time interval between “B” and “C” is called the data setup time and is specified by the receiving device. At some point in the interval between times “C” and “D”, the receiving device reads the data bus. The receiving device then asserts the RxRdy signal low to indicate to the sender that it has acquired the data. The sending device then asserts its TxReq signal low at time “E” to indicate to

the receiver that the data is no longer available on the data bus. The data must not actually change until time "F" to meet the data hold time specified by the receiving device. The minimum time between successive time points marked "A" is called the cycle time. Regardless of how quickly the sender is able to send data, the data maximum exchange rate must conform to the specifications of the receiving device.

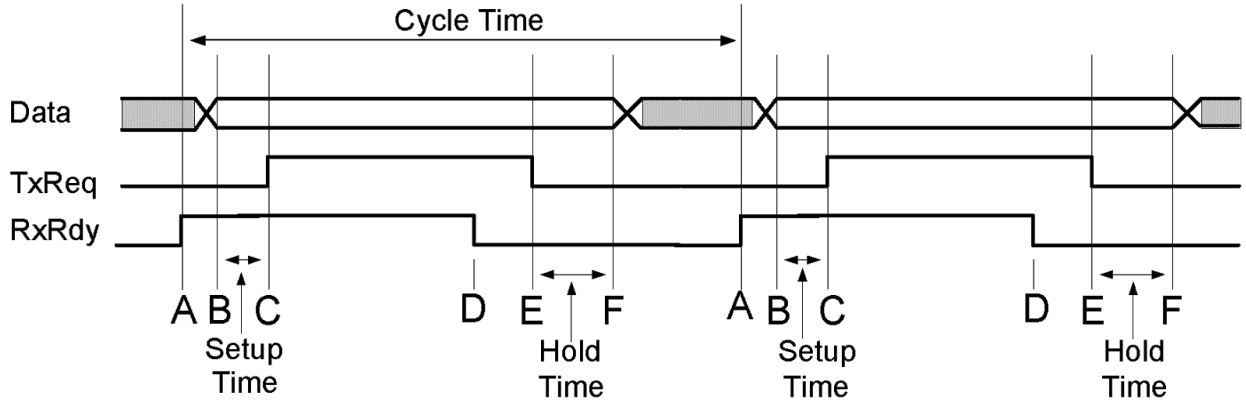

Figure 7.2. Timing diagram of four-phase handshaking showing two complete cycles.

For bidirectional data exchanges, the handshaking signal must be mirrored, as illustrated in Fig. 7.3. Data sent in only one direction (as illustrated in Fig. 7.1) is called simplex communications. When communication is restricted to one direction at any given time, it is referred to as half-duplex operation. This is usually the case for devices that share a common data signal line, as illustrated in Fig. 7.3. Full-duplex operation refers to simultaneous bidirectional data exchange and requires separate data signal lines.

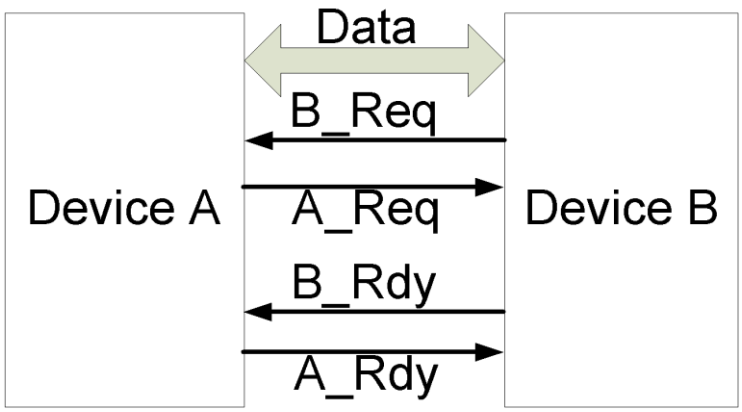

Figure 7.3. Interconnection bi-directional for four-phase handshaking.

A four-phase handshaking scheme can use an explicit strobe signal from the sender to the receiver and implicit handshaking from the receiver to the sender. The sender assumes implicitly that the receiver is ready, provided the timing constraints for the receiving device are met. Figures 7.4 and 7.5 show two common four-phase handshaking interfaces. Since Device "A" controls both the timing and the direction of the data flow, the configuration is commonly referred to as master-slave operation. Figures 7.6 and 7.7 show the timing for the four-phase handshaking with the receiver, with the implication that the device is always ready to receive new data. Additionally, the sending device assumes that the receiving device has acknowledged the data transfer after a specified period of time. In both cases, the interval between B and C is called the data setup time and the interval between D and F is called the data hold time. In Fig. 7.6, the interval between A and B is called the control setup time and the time between D and E is called the control hold time.

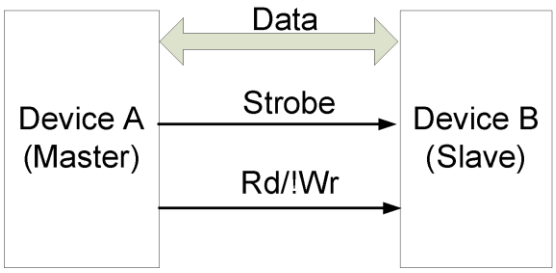

Figure 7.4. Half-duplex using an ENABLE strobe and read / write control.

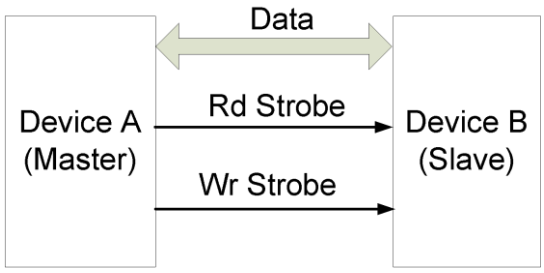

Figure 7.5. Half-duplex using separate read and write strobes.

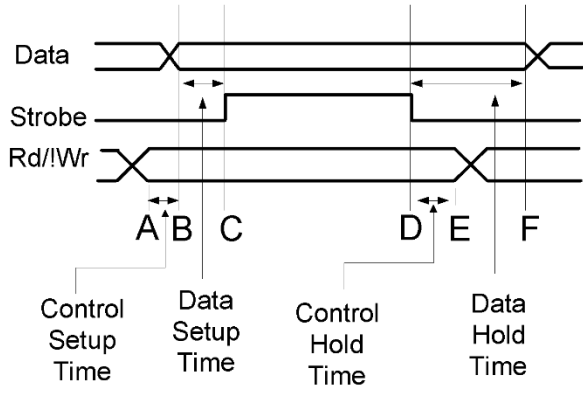

Figure 7.6. Timing diagram of four-phase handshaking with implied receiver ready corresponding to Fig. 7.4.

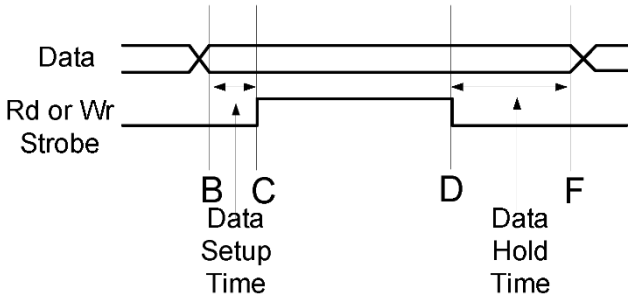

Figure 7.7. Timing diagram of four-phase handshaking with implied receiver ready corresponding to Fig. 7.5.

There is also a two-phase handshaking scheme that is beyond the scope of this discussion. The interested reader should refer to the web link at Velocity Reviews to become informed on the differences between four-phase and two-phase handshaking.

7.2 LCD Hardware Interface

We are using an LCD with a parallel interface to demonstrate the concept of handshaking needed for electronic communications. The LCD will be used in many of the future projects to serve as a local display of varying data. The general hardware configuration for this project is shown in Fig. A.1 of Appendix A. The Basys MX3 processor board shows that it is populated with an SD1602GSLB LCD (for which the datasheet is incomplete). It includes the correct pinout, which we have listed here in Table 7.1; however, it does not include the necessary timing requirements. Because the SC1602B model uses the same Samsung KS0066 controller as the SD1602GSLB, we will assume it is sufficiently similar and use the timing information from the SC1602B datasheet.

Table 7.1. PIC32 Pin Assignment for the LCD Interface.

Port #	Processor Function	LCD Pin
RE0	PMD0/RE0	DB0
RE1	PMD1/RE1	DB1
RE2	AN20/PMD2/RE2	DB2
RE3	RPE3/PMD3/RE3	DB3
RE4	AN21/PMD4/RE4	DB4
RE5	AN22/RPE5/PMD5/RE5	DB5

RE6	AN23/PM6/RE6	DB6
RE7	AN27/PMD7/RE7	DB7
RD4	RPD4/PMWR/RD4	DISP_EN
RD5	RPD5/PMRD/RD5	DISP_R/W
RB15	AN15/RPB15/PMA0/CTED6/RB15	DISP_RS

Although LCD units produced by various manufacturers have different numbers of characters per line and different numbers of display lines, they have an interface that has become a de facto standard. This interface will be discussed in detail in the following paragraphs.

There are three signals that the microprocessor uses to control the data transfer with the LCD module. The Enable (E) signal is a read or write strobe, depending on whether the Read/!Write signal is high or low. (The "!" character indicates that the signal is active when the signal is asserted low, commonly referred to as "active low.") The action of the read and write operations is from the perspective of the microprocessor that serves as the master controller. The LCD controller IC has multiple memory banks or data storage areas: the configuration registers, the data display RAM (DDRAM), and the character generation RAM (CGRAM). The register select (RS) signal determines whether data is exchanged with the LCD DDRAM/CGRAM or the configuration registers. In regards to the handshaking discussion above, the interface we will be using represents a half-duplex communications scheme. Therefore, the LCD handshaking conforms to what was described for Fig. 7.4 and Fig. 7.6.

Figure 7.8 shows the timing diagram for the LCD write cycle. The RS and the R/W signals must be asserted for a period equal to or longer than the time specified by the RS and R/W setup time parameter, t_{su1}, prior to the E signal being asserted high. The diagram shows that the processor is permitted to set the pins used for data as outputs and write the data to the I/O port any time prior to the E signal, making a high to low transition, provided that this happens at a time equal to or greater than the write data setup time, t_{su2}. In theory, the R/W signal should be set low before the data pins are set as outputs.

The processor must assert the E signal high for a minimum time as specified by the period, t_w. The processor is expected to maintain the logic level for RS and R/W for a period greater than t_{h1} and the data must remain at constant logic levels on the output pins for a period equal to or longer than that specified by the write data hold time, t_{h2}.

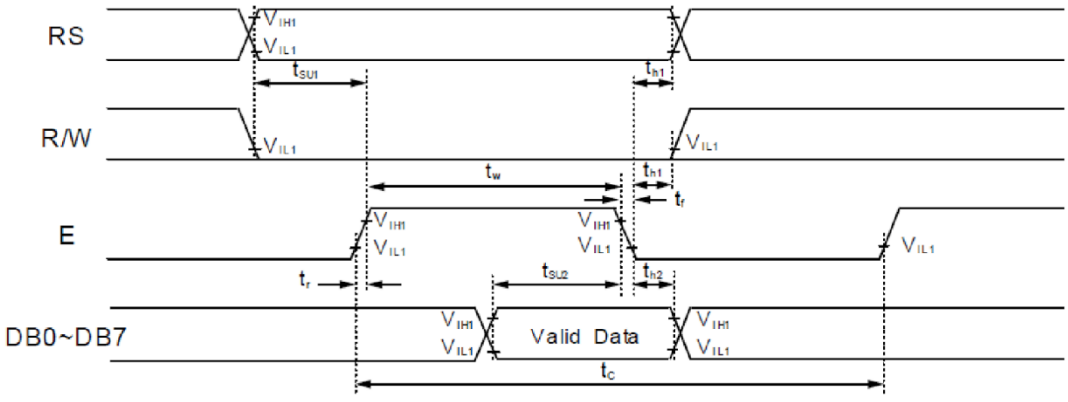

Figure 7.8. LCD character write cycle timing.

The LCD read cycle timing provided in Fig. 7.9 shows that both the RS and R/W signals must be asserted for a period equal to or longer than the time specified by the R/W setup time parameter, t_{su}, prior to the E signal being asserted high. Also, prior to asserting the E signal high, the processor I/O pins used for the data lines must be set as

inputs to avoid both the LCD and the PIC32 I/O pins contending to control the level of the data lines. The processor must assert the E signal high for a minimum time, as specified by the period, t_w. After the read data output delay interval (t_D) and prior to the processor asserting the EN signal low, the processor will be able to read the data outputs from the LCD. The LCD maintains the data output until the E signal has been asserted low, and through the specified read data hold time (t_{DH}).

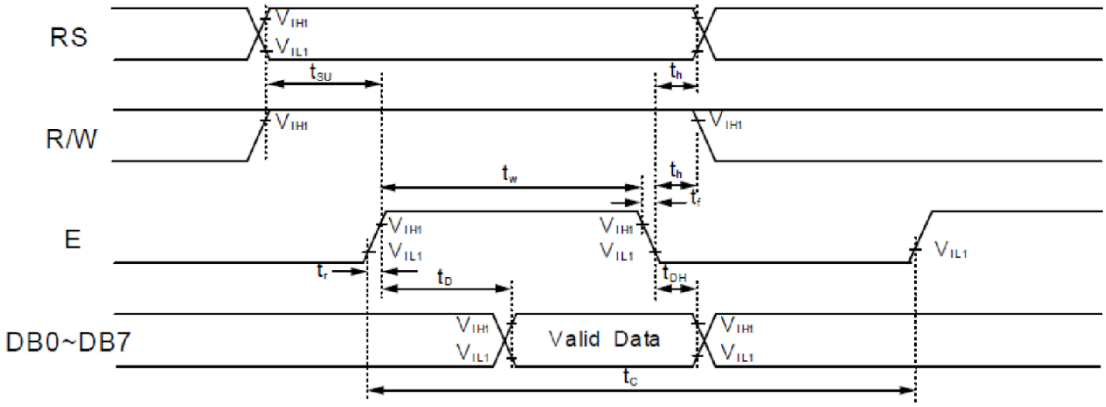

Figure 7.9. LCD Read cycle timing.

The minimum time before the read or write operation can be repeated is set by the enable cycle time, t_c. Table 7.2 provides the values of the timing parameters for both read cycles and write cycles. The times shown in Table 7.2 are worst case times for a supply voltage in the range of 2.7 V to 4.5 V and operating temperature in the range of -30 °C to +85 °C.

Table 7.2. LCD signal timing.

Mode	Characteristic	Symbol	Min	Typical	Max	Units
Write Mode Refer to Figure 7.8	E Cycle Time	t_c	1000	-		ns
	E Rise/Fall Time	t_r, t_f	-	-	25	
	E Pulse Width (High, Low)	t_w	450	-	-	
	R/W and RS Setup Time	t_{su1}	60	-	-	
	R/W and RS Hold Time	t_{H1}	20	-	-	
	Data Setup Time	t_{su2}	195	-	-	
	Data Hold Time	t_{H2}	10	-		
Read Mode Refer to Figure 7.9	E Cycle Time	t_c	1000	-		ns
	E Rise/Fall Time	t_r, t_f	-	-	25	
	E Pulse Width (High, Low)	t_w	450	-	-	
	R/W and RS Setup Time	t_{su1}	60	-	-	
	R/W and RS Hold Time	t_{H1}	20	-	-	
	Data output Delay Time	t_d	-	-	360	
	Data Hold Time	t_{H2}	5	-		

7.3 LCD Software Interface

The character LCD has an initialization or configuration phase before the operational phase. One approach to software development for this project is to partition the software into six functions or modules, as shown in Fig. 7.10. Detailed control flow diagrams for the LCD Write and the LCD Read functions are provided by Fig. 7.12 and

Fig. 7.13. The control flow diagrams only apply for code that uses bit-banging techniques for implementing the LCD interface, as discussed in detail below. These LCD read and write functions control the hardware interface and must be specifically written to match the I/O pin connections from the microprocessor to the LCD unit. The remaining four LCD functional blocks depicted in Fig. 7.10 are not dependent on hardware connections.

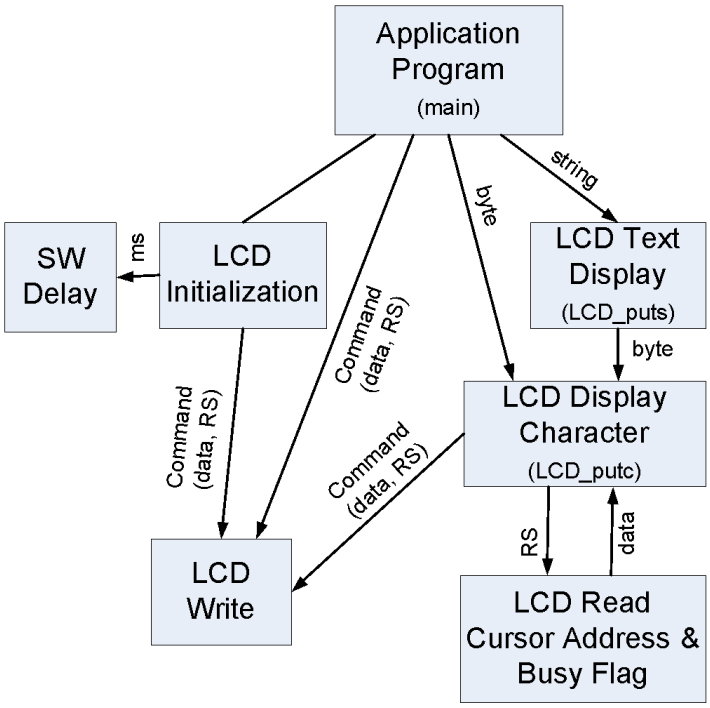

Figure 7.10. Data flow diagram for the LCD control.

The LCD configuration sequence is a series of commands written to the LCD after a power-up or CPU reset. The values that are written to the LCD, shown in Fig. 7.11, set up the LCD for an 8-bit data interface with a blinking cursor. The cursor address is reset to zero, placing the cursor at the start of the first line. The cursor address automatically increments as new DDRAM data is written to the LCD. The cursor address is the memory address of the DDRAM where the cursor is seen and the location where the next character written to the DDRAM will be displayed. Note that the Busy Flag is not polled during the initialization because it may not be valid until the sequence is completed.

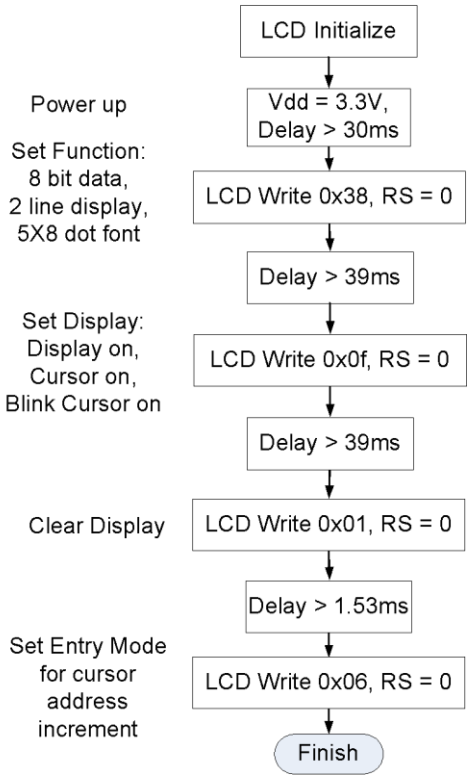

Figure 7.11. Control flow diagram for the LCD initialization.

Due to the high execution speed of the PIC32 processor, the "Wait" tasks shown in Fig. 7.12 and Fig. 7.13 are required to be implemented as software delays. The LCD Write and LCD Read software flow diagrams represent the transfer of data from the PIC32 to one of two internal LCD registers. The internal operations of the LCD operate at a much slower speed, as shown in Table 7.3. It should be noted that the busy flag will be read immediately following the writing of an instruction. The next instruction, whether it is a control command or display data, can be written to the LCD as soon as the busy flag is read as zero. The execution times shown in Table 7.3 are worst case times and are affected by operating temperature and supply voltage.

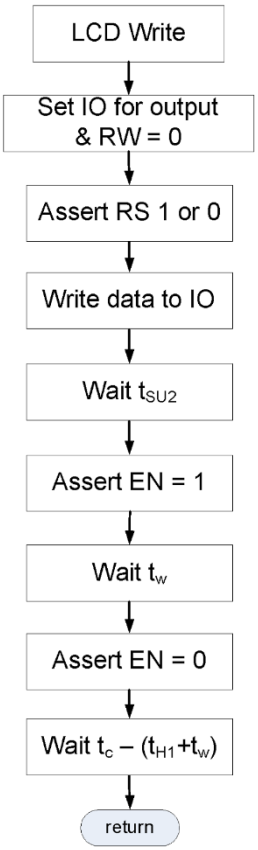

Figure 7.12. Control flow diagram for LCD Write operation using bit-banging.

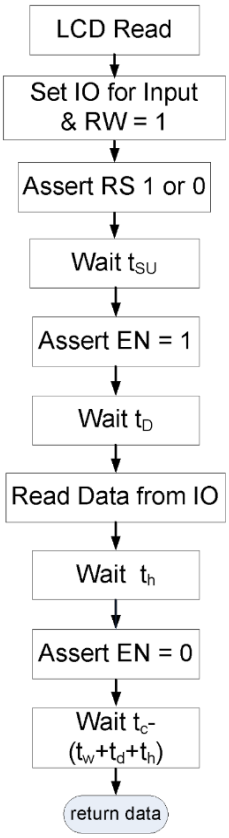

Figure 7.13. Control flow diagram for LCD Read operation using bit-banging.

The LCD has an additional set of buffer registers labeled the IR and DR. The PIC32 can write to these two registers following the timing requirements shown in Table 7.2, or following the control flow diagrams shown in Fig. 7.12 and Fig. 7.13; however, the LCD has a much slower clock speed and requires more time to implement the instructions written to the buffer. The actual LCD minimum implementation times are shown in Table 7.3. One method of incorporating the various implementation times is to add time delay after the instruction has been written. Doing so requires that the additional delay be over specified to ensure proper LCD operation.

Table 7.3. LCD control instruction tables.

Instruction	Instruction Code										Description	Execution time (fosc = 270 kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write “20H” to DDRAM and set DDRAM address to “00H” from AC.	1.53 ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.53 ms

Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction and enable the shift of entire display.	39 µs
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Set display (D), cursor (C), and blinking of cursor (B) on/off control bit.	39 µs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display shift control bit, and the direction, without changing of DDRAM data.	39 µs
Function Set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-bit/4-bit), numbers of display line (N: 2-line/1-line), and display font type (F: 5x11 dots/5x8 dots).	39 µs
Set CGRAM Address	0	0	0	1	AC 5	AC 4	AC 3	AC 2	AC 1	AC0	Set CGRAM address in address counter.	39 µs
Set DDRAM Address	0	0	1	AC 6	AC 5	AC 4	AC 3	AC 2	AC 1	AC 0	Set DDRAM address in address counter.	39 µs
Read Busy Flag and Address	0	1	BF	AC 6	AC 5	AC 4	AC 3	AC 2	AC 1	AC 0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0 µs
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43 µs
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43 µs

As described above, the LCD controller has two sets of memory registers: one for configuring the LCD operations and one for holding the data that generates the LCD display characters. The register select (RS) signal controls which set of memory registers are accessed for both read and write operations. The display memory is further

divided into memory for the display of a fixed set of symbols and characters and memory that can be used to generate user defined characters. The display data Ram (DDRAM) is used for displaying the predefined text characters. By default, all data is written to and read from the DDRAM when the RS signal is high.

The character generator RAM (CGRAM) is used to generate special symbols that can be programmed by the controlling processor. Accessing the CGRAM and generating special LCD symbols and characters is beyond the scope of this project. See Appendix C for information regarding configuring the LCD for this mode of operation.

Other than the LCD initialization function, the LCD Write and Read functions depicted in Fig. 7.12 and Fig. 7.13 are the only functions that directly interface to the LCD hardware. The same LCD byte write function is used whenever sending a byte to either the data RAM or to the control registers. This is possible because the value that RS is to be set to is passed to the LCD read and LCD write functions. The advantage of implementing the LCD control in this manner is that it limits the hardware specific code to these two functions. This maximizes the portability of user code to different applications, and results in better memory resource utilization.

After initializing, display characters can be written to the LCD. Each LCD character position is assigned a DDRAM address. DDRAM addresses of the first line of the LCD display are from 0 to 0x0F. The addresses of the second display line are from 0x40 to 0x4F. The DDRAM addresses from 0x10 to 0x3F are not used. If a character is written to DDRAM address 0x0F, the LCD will properly display the character in the rightmost position of the first line and increment the cursor address to the value 0x10. If additional characters are to be displayed beginning on the second line, the cursor address must be first reset to 0x40. The process of repositioning the cursor position is described below.

Figure 7.14 shows the control flow diagram (CFD) for displaying a single character on the LCD. Initially, the status of the Busy Flag is read until the LCD reports that it has been set low. This is accomplished by repeatedly reading the LCD with the RS signal set low and the RW signal set high, as is demonstrated in Fig. B.2 and B.3 in Appendix B. The most significant bit (D7) of the data byte read from the LCD represents the busy flag - if this bit is set to a one, the LCD is busy. The seven least significant bits in the byte read from the LCD contain the DDRAM address of the cursor position: 0x00 through 0x0F for the first line and 0X40 through 0X4F for the second line.

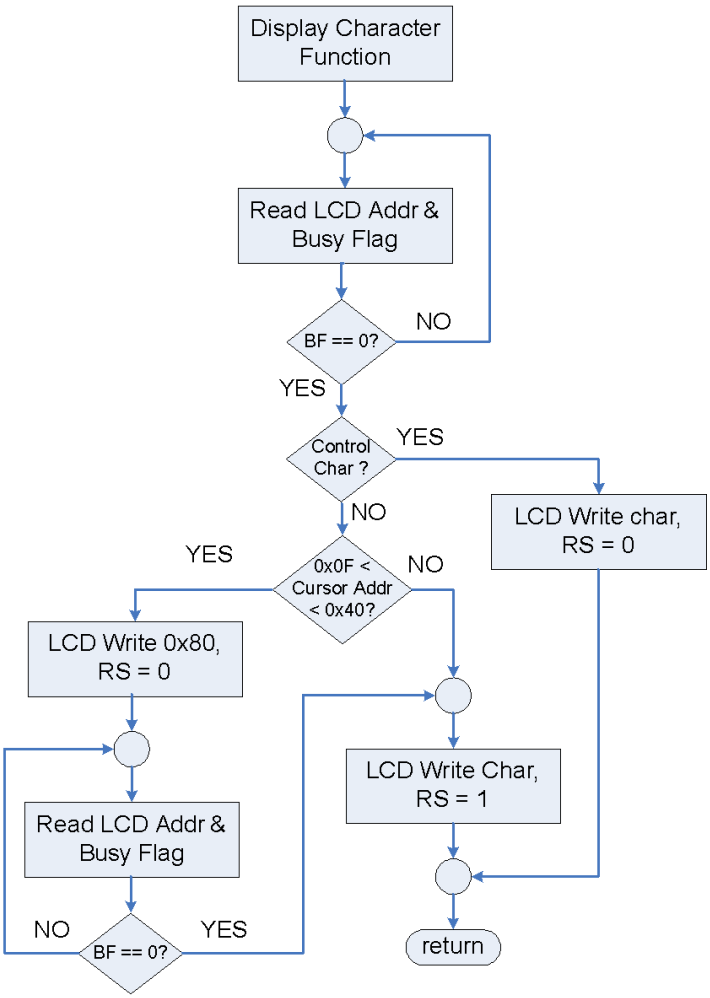

Figure 7.14. LCD display character control flow diagram.

After reading the cursor address and busy flag, the next step shown in Fig. 7.14 requires testing the binary value of the character passed to this function to determine if it is an ASCII control character. For instance, when either the carriage return (CR) or the new line (NL) ASCII control characters are sent to this function, the code must send the ***Return Home*** or the ***Clear Display*** instruction codes to the LCD by passing one of the values 0x02 or 0x01, respectively, with both the RS and RW signals set low.

If writing a string of ASCII text to the LCD, the cursor address is automatically incremented to point to the next character position. No characters will be displayed if the cursor address is between 0x10 and 0x3F, hence the cursor does not automatically increment from the end of the first line to the beginning of the second line. It requires additional code to reposition the cursor from the end of the first line to beginning of the second. This is accomplished by writing the new cursor address, 0x40, added to the write DDRAM control bit, 0x80, resulting in a value of 0xC0 that is written to the LCD with the RS and RW signals set low. At this point, the program must wait until the busy flag is set to zero before continuing on with the process of writing the next character to the LCD. The busy flag represents a blocking semaphore, or signal, because the processor is held up waiting for the busy flag to clear. Once the LCD busy flag is read as zero, the RS signal must be set high and the RW signal set low. This is in addition to setting the PIC32 PORT E I/O pins zero through seven as outputs so the ASCII value of the character to be displayed can be written to the LCD DDRAM. The control flow diagram shown in Fig. 7.14 can be modified to implement additional ASCII control characters.

Most LCD modules require multiple microseconds to complete the CLEAR DISPLAY and RETURN HOME commands. According to the Samsung KS006U data sheet, operations other than READ BUSY FLAG and ADDRESS, CLEAR DISPLAY, and RETURN HOME operations can require up to 43 μs to complete. The CLEAR DISPLAY and RETURN HOME operations can require up to 1.53 ms to complete. The READ BUSY FLAG and ADDRESS operation can be completed as quickly as the LCD timing permits. This allows LCD BUSY FLAG to be polled at the maximum rate provided by the minimum Enable cycle time shown in Fig. 7.8 and Fig. 7.9. The time to complete any write operation varies from one LCD manufacturer to another, so the particular device data sheet must be examined closely. The Enable cycle time parameter, t_c, shown in Fig. 7.8 and Fig. 7.9, in conjunction with Table 7.2, clearly shows that there is a maximum rate of 1 MHz at which the LCD Busy Flag can be read.

7.3 Bit-banging LCD Interface

Any microprocessor can interface to a character LCD using the technique called "bit-banging." Bit-banging refers to the process of explicitly setting specific output pins either high or low as required by the handshaking protocol. This is accomplished using software instructions, such as LATxSET and LATxCLR, to control the state of the RS, RW, and E signals. The interface shown in Fig. A.1 provides the PIC32 port and pin specifications to allow the LCD control using this method. The code developer needs only to write the software code to set the control direction of the pins used for data, and the values of the pins used for control, in the sequence described by the control flow diagrams shown in Fig. 7.12 and Fig. 7.13. For example, Listing 7.1 is the C code you can use to implement the LCD_read function shown in Fig. 7.13 using the bit-banging technique. The code in Listing 7.1 is written under the assumption that the setup and hold requirements are met and no time delays are required to meet setup and hold times. When using a high-speed processor, such as the PIC32 running at 80 MHz, additional delays may be needed to meet the minimum timing requirements.

Listing 7.1. Example Bit-banging Code for the LCD_read Function

```
#include <plib.h>          // Defines BIT constants
#define LCD_DATA (BIT_0|BIT_1|BIT_2|BIT_3|BIT_4|BIT_5|BIT_6| BIT_7) // Port E
#define LCD_EN      BIT_5         // Port D
#define LCD_RS      BIT_15        // Port B
#define LCD_RW      BIT_4         // Port D

int LCD_read(int RS)
{
       int status;
       PORTSetPinsDigitalIn(IOPORT_E, LCD_DATA); //Set as input to read
       mPORTDSetBits(LCD_RW);            //LCD_RW set high to read LCD
       if(RS)
              mPORTBsetBits(LCD_RS);     //LCD_RS set high
       else
              mPORTBClearBits(LCD_RS); // LCD_RS low
/* If the RS and RW setup time must be extended - a delay must be added here */
       mPORTDSetBits(LCD_EN);            //Enable set high
/* If the enable time must be extended - a delay must be added here */
       status = mPORTERead();            //Read BF and ADDR from LCD
       mPORTDClearBits(LCD_EN);   //Enable set low
       return (status);
}
```

7.5 Parallel Master Port LCD Interface

The PIC32 family of processors is enabled with an addressable parallel port called the Parallel Master Port, or PMP. The Basys MX3 trainer board was designed to take advantage of this resource when interfacing with the onboard parallel interface LCD. Using the PMP interface to the LCD is an alternative method to the bit-banging approach

described above. It is beyond the scope of this project to explain all of the various ways the PMP can be configured. In general, the PMP consists of an 8- or 16-bit bidirectional parallel I/O data bus, a dedicated or multiplexed address bus, and control signals to allow read, write, and address latch enable operations.

The purpose of the PMP is to facilitate a single parallel data bus interface with multiple devices. The general architecture of a parallel data bus is shown in Fig. 7.15. Figure 7.16 illustrates the connection between the PIC32 PMP signals and the LCD. Note that Fig. 7.16 conforms to the same general I/O pin configuration that is shown in Fig. 7.4.

The address bus is configured as a 1-bit signal representing address signal A0 connected to the LCD RS input. The PIC32 strobe output pin is connected to the LCD E input and the PIC32 read/write output control pin to the LCD R/W input. The PIC32 pins assigned to the PMP functions are fixed in hardware. The Basys MX3 interface with the LCD uses the PMP pin assignment shown in Fig. 7.16.

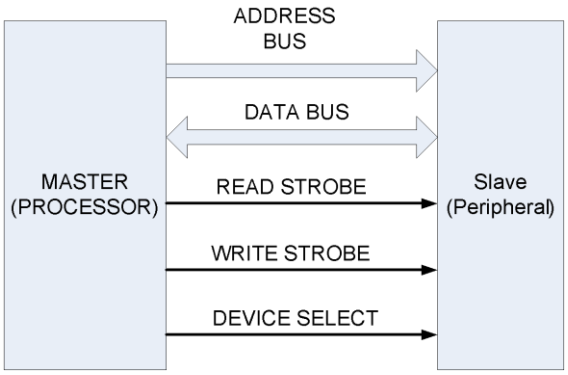

Figure 7.15. Architecture of a parallel data bus.

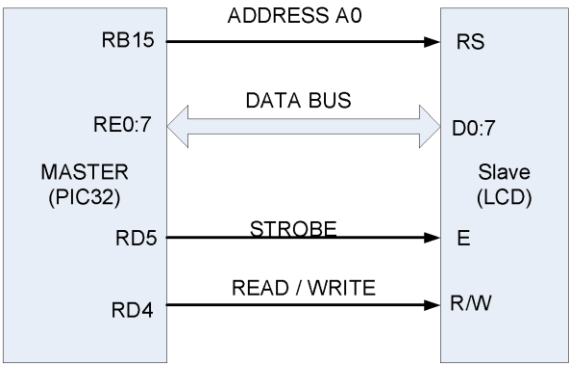

Figure 7.16. PIC32 PMP interface with LCD.

Since the some of the pins used for the LCD interface with the PIC32MX370 processor are either analog inputs or digital I/O, it is necessary to first declare all LCD interface pins as digital by either using the PLIB function *PORTSetPinsDigitalIn(…);* or *ANSELxxx = 0* as shown in Listing 7.2. Using the functions provided by the PIC32 Peripheral library (Section 17), the PMP is configured with the four parameters in the four configuring declarations and the PMPOpen statements, shown below in Listing 7.2.

Listing 7.2. PMP Initialization Code for LCD Interface

```
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_1 | PMP_WAIT_MID_2 | PMP_WAIT_END_1;
int cfg3 = PMP_PEN_0;         // only PMA0 enabled
int cfg4 = PMP_INT_OFF;       // no interrupts used

PORTSetPinsDigitalIn(IOPORT_E, LCD_DATAbits);  // RE0:7
PORTSetPinsDigitalOut(IOPORT_D, ENpin);        // RD4
PORTSetPinsDigitalOut(IOPORT_D, RWpin);        // RD5
PORTSetPinsDigitalOut(IOPORT_B, RSpin);        // RB15

mPMPOpen(cfg1, cfg2, cfg3, cfg4);

// End of Listing
```

The first bit parameter in the variable *cfg1* enables the PMP peripheral. The second bit parameter enables both of the PMP read and write operations. The PMP_READ_POL_HI bit with the PMP_WRITE_POL_HI bit results in setting the last two significant bits of the PMCON register, and it indicates that the PMP read/write control is to be high when reading and low when writing. The interpretation of these two bits is conditional on the

PMP_MODE_MASTER1 control bits of the PMMODE register and implementation is done via the handshaking operation illustrated back in Fig. 7.4. The variable *cfg2* is written to the PMMODE register. The PMP_DATA_BUS_ 8-bit control specifies that the data bus is 8 bits. The three PMP_WAIT controls are discussed in detail below.

The *cfg3* variable is set for PMP_PEN_0, indicating that only PMP address bit 0 is enabled and is written to the parallel port address register, PMADDR.

The last parameter disables PMP interrupts. If the PMP interrupt is enabled, with the PMP configured for the master mode, an interrupt will be generated on every completed read and write operation. With the PMP interrupt disabled, the PMFLAG will need to be polled to determine when the PMP read and write cycles are completed.

The WAIT controls in the PMP mode register control the timing of the assertion of the RD/WR select signal in relation to the LCD Enable strobe. Each vertical dashed line in Fig. 7.17 represents intervals of the peripheral bus clock period, marked T_{PB}, where the period of T_{PB} is the inverse of the peripheral bus clock frequency. The number of T_{PB} periods for the "B" period is specified by PMP_WAIT_BEG_b ($1 \le b \le 4$), specified in the PMMODE configuration setting in Listing 7.2. Likewise, the M indicates the number of T_{PB} periods that the LCD E signal will be active, as specified by PMP_WAIT_MID_m ($0 \le m \le 15$). The minimum strobe period is one T_{PB}. Hence, the total strobe period is PMP_WAIT_MID_x plus one. The ending interval marked "E" in Fig. 7.17 indicates the number of T_{PB} periods that the RD/WR signal will retain its value, PMP_WAIT_END_e ($1 \le e \le 4$). For the case shown in Fig. 7.17, the three parameters would be: PMP_WAIT_BEG_4, PMP_WAIT_MID_6, and PMP_WAIT_END_2.

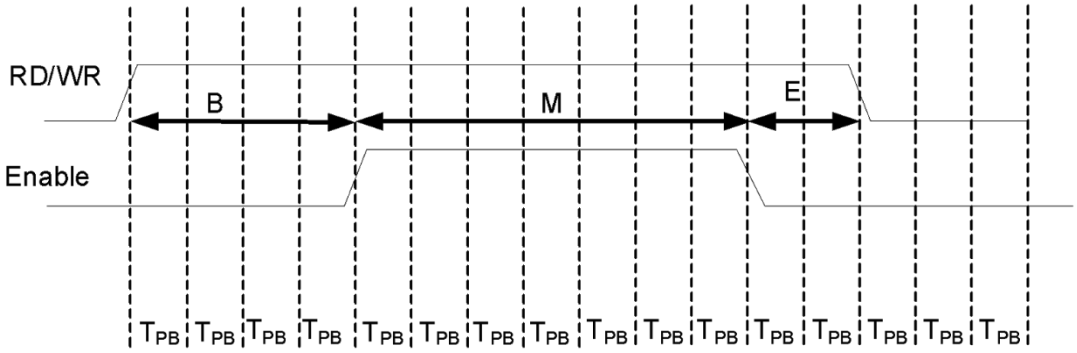

Figure 7.17. The PMP timing diagram for the beginning, middle, and end wait periods.

The minimum settings for the three PMP WAIT intervals can be determined by dividing the value for t_{SU} on the LCD datasheet by T_{PB}. For example, assume that the peripheral bus clock is configured for 10 MHz. The t_{SU} specified in the KU0066U data sheet is 60 ns, resulting in interval B being equal to 0.6 T_{PB} periods. Of course, this should be rounded up, so a proper setting is PMP_WAIT_BEG_1. The value for the interval M is computed using the value specified for t_w. In this case, M+1 is equal to 2.5, and after rounding up and subtracting one, M equals 2. Thus, PMP_WAIT_MID_2 can be used. The value for interval E is 0.1, resulting in PMP_WAIT_END_1. Since [(B+[M+1]+E) * T_{PB}] represents the minimum time to complete an LCD read or write operation, this minimum time must be greater than the LCD enable cycle time, t_c, which is specified as 450 ns. For the configuration described above, [(B+[M+1]+E) * TPB] = [(5) * 100 ns] = 500 ns, thereby satisfying the LCD timing specifications. If the cycle time is too short, wait periods can be added to any segment within the allowable ranges.

Since the PMP is clocked from the peripheral bus clock, it is likely that the PMP cycle is slower than the core CPU, which means that the PMP also has a busy flag that must be polled before writing or reading the next byte. With the PMP interrupt disabled, the only way to determine if the PMP is ready for the next read or write cycle is to poll the PMP flag. The *PMPMasterWrite* function provided in the PIC32 Peripheral Library polls the PMP flag to ensure it has been cleared prior to writing. The code for writing to the LCD using the PMP is shown in Listing 7.3.

Listing 7.3. LCD Write Function Using the PMP

```
void writeLCD( int addr, char c)
{
   while( busyLCD());      // Wait for LCD to be ready
   PMPSetAddress( addr);  // Set LCD RS control
   PMPMasterWrite( c);    // initiate write sequence
} // End of writeLCD
```

Reading the PMP requires two successive read operations, as shown in Listing 7.4. Figure 7.17 shows that the PIC32 reads the Busy Flag 14 times before the Busy Flag resets. Since the *readLCD* function calls *mPMPMasterReadByte();* twice in a row before returning the value, the *readLCD* function is only called seven times before the Busy Flag is returned as zero. Regardless of how many times the Busy Flag is read, the LCD doesn't return a Busy Flag value of zero until 33.6 µs after the LCD write operation. A capture of the timing for the PMP is shown in Appendix B.

Listing 7.4. LCD Read Function Using the PMP

```
char readLCD( int addr )
{
   PMPSetAddress( addr);        // Set LCD RS control
   mPMPMasterReadByte();        // initiate dummy read sequence
   return mPMPMasterReadByte();// read actual data
} // End of readLCD
```

For additional details concerning the PIC32 PMP, refer to Section 13 of the PIC32 Family Reference Manual.

8 References

1. Basys MX3 Data sheet
2. PIC32MX330/350/370/430/450/470 Family Data Sheet
3. 2 Line 16 Character LCD Data Sheet, http://www.lcd-modules.com.tw/upload/web/SC/SC1602B.pdf
4. LCD Data sheet, https://www.sparkfun.com/datasheets/LCD/ADM1602K-NSW-FBS-3.3v.pdf
5. ASCII Table, http://www.asciitable.com/

Appendix A: Unit 3 Parts Configuration

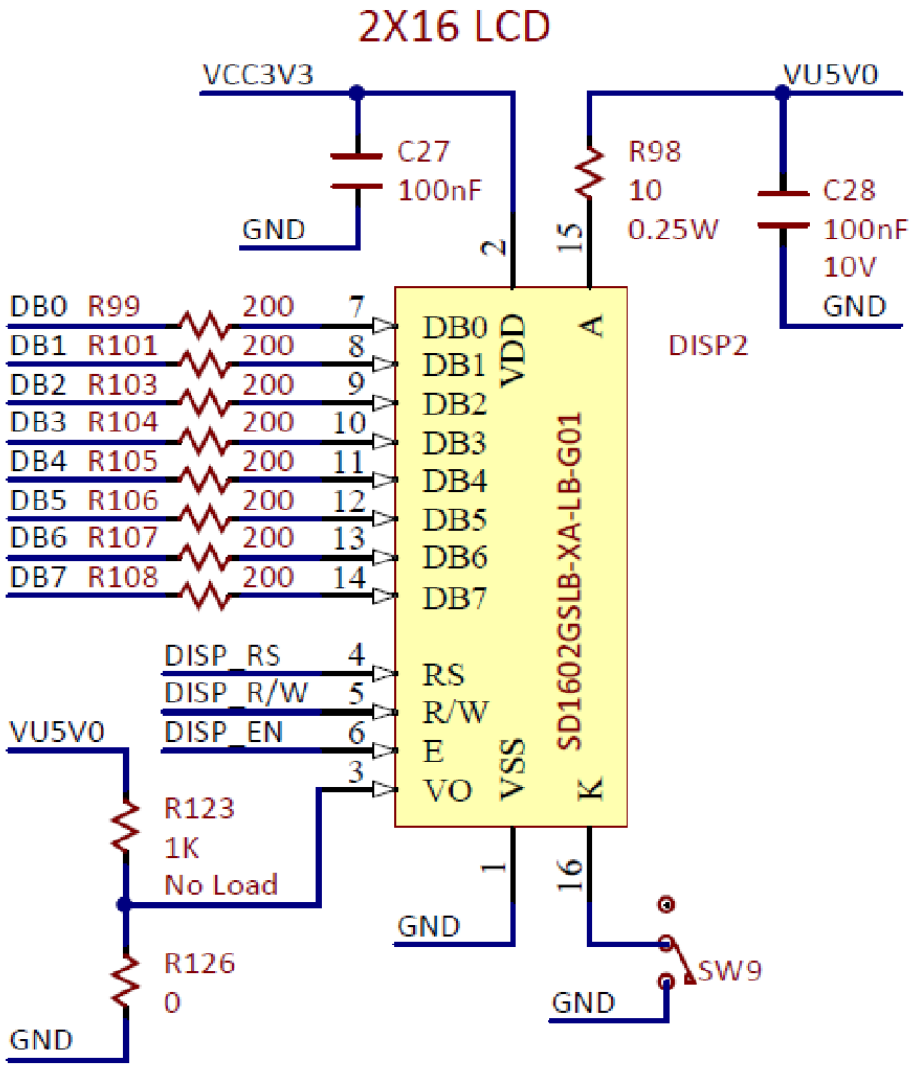

Figure A.1. LCD connection schematic diagram for Labs 3a and 3b.

Appendix B: LCD Custom Instrumentation

As noted above in section 4.2, Figures B.2 through Figure B.4 show data from an Analog Discovery 2 connected directly to the LCD data pins and the three handshaking and control pins, RS, R/W, and EN. Figure B.1 shows a prototype of the Basys MX3 that was altered to include longer header pins on the LCD. This custom modification was made to enable testing during software development. It is not required to make this modification in order to complete Lab 3a and Lab 3b.

Note the Analog Discovery 2 is not connected directly to the on-board 30-pin Analog Discovery 2 connector, J4, in this case because the LCD data pins are not broken out to the Analog Discovery Interface connector.

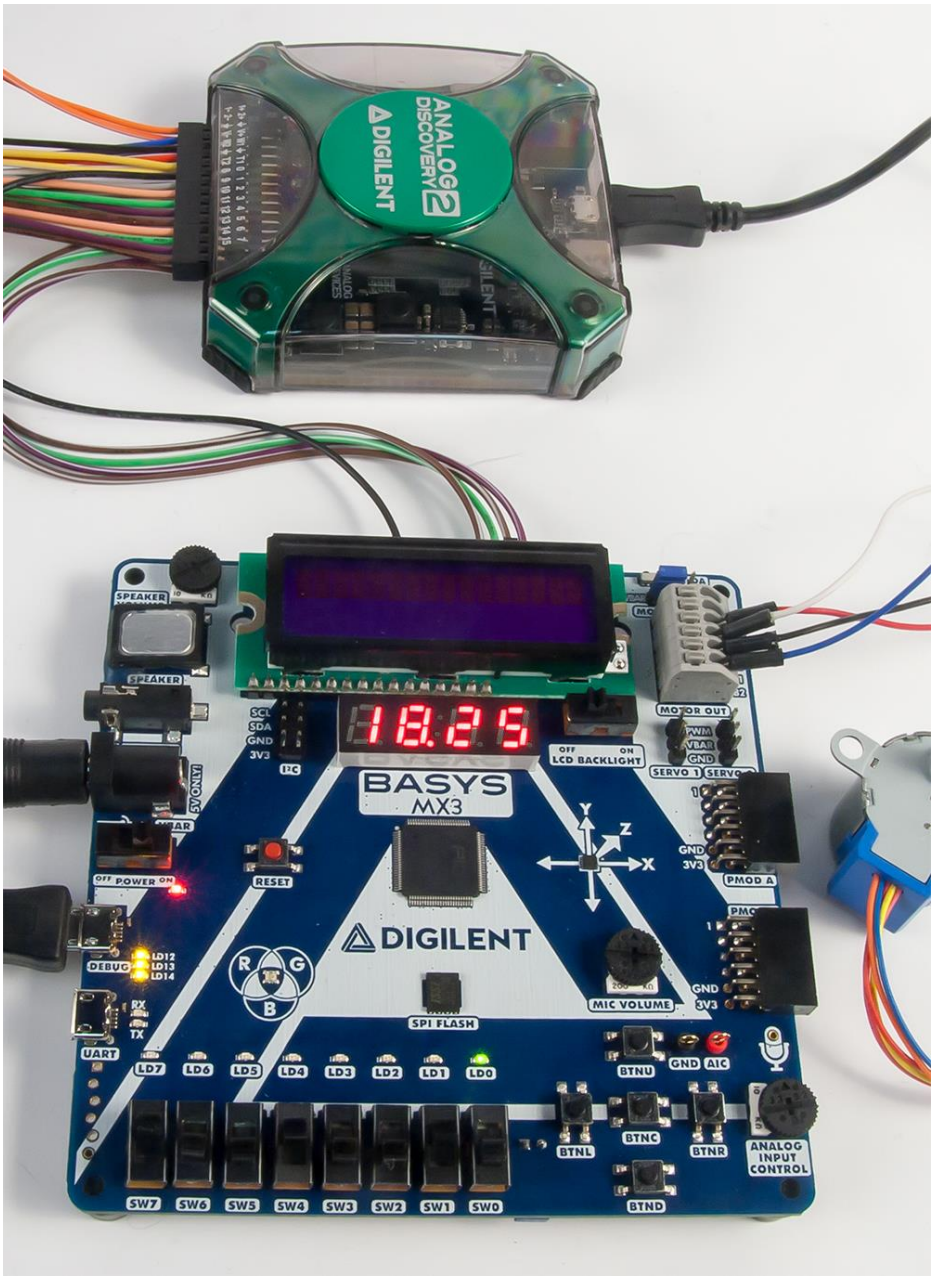

Figure B.1. Equipment configuration used in Lab 3a and 3b for Unit 3.

The alternate method to implementing fixed delays, as was discussed in the main body of this text, when writing control or display data bytes is to continuously read the LCD Busy Flag and wait for the Busy Flag to be reset before

completing any read or write operation. Therefore, there is no waiting to read the Busy Flag. Figure B.2 is a screen capture of this operation. The Busy Flag and DDRAM address is initially read as 0x4c, meaning that the cursor is located at the 12th character position on the second line and the Busy Flag is off. Following that, a new cursor position, 0x48, is written to the LCD. The value shown in Fig. B.2 is 0xC8 because the 8th bit is set to instruct the LCD to move the cursor to a new position using the "Set DDRAM Address" instruction. The Busy Flag and DDRAM address is read six consecutive times. The busy flag is set high the first five times. The sixth time the busy flag is set low (not busy) and the address shows 0x48, meaning the cursor position is now located for the eighth character position of the second row. The period the Busy Flag reads high for is 33.6 µs. This is consistent with the 39 µs listed in Table 7.3. Again, the execution times shown in Table 7.3 are worst case and dependent on operating temperature and supply voltage.

Figure B.2. LCD timing diagram for a Set Cursor Position instruction using 1 µs setup and EN extension delays.

Figure B.3 shows the timing for the Set Cursor Position instruction where the read cycle speed is increased by a factor of 10. Note that the period waiting for the instruction execution to complete is still 33.6 µs, thus showing the LCD processing time is independent of the processor to LCD interface speed.

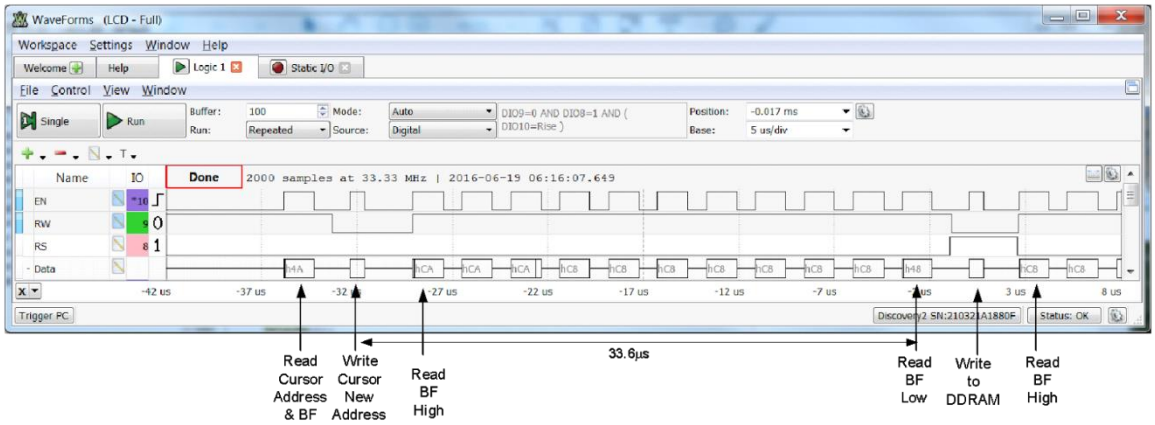

Figure B.3. LCD timing diagram for a Set Cursor Position instruction using 0.1 µs setup and EN extension delays.

Figure B.4 shows the similar timing of the Set Cursor Position when the PMP interface to the LCD is used in place of the Bit-Banging type of parallel bus control. As one can readily see, the LCD response in Fig. B.3 is very similar to Fig. B.4; however, as discussed in Section 7.5, the PMP interface requires far fewer lines of instruction code.

Figure B.4. LCD timing diagram for a Set Cursor Position instruction using the PIC32 PMP interface.

Appendix C: LCD Custom Characters

When you send the ASCII code for a character like "A" to an LCD module, the LCD controller looks up the appropriate 5x8-pixel pattern in ROM (read-only memory) and displays that pattern on the LCD. That character-generator ROM contains 192 bit-mapped characters corresponding to the alphabet, numbers, punctuation, and a limited set of Japanese Kanji characters and Greek symbols.

The ROM is part of the main LCD controller (e.g., HD44780, KS0066, etc.), is mask-programmed, and cannot be changed by the user. The manufacturers do offer alternative symbol sets in ROM for European and Asian languages, but most U.S. distributors stock only the standard character set shown in the LCD Serial Backpack manual.

Alphanumeric LCD controllers do not allow you to turn individual pixels on or off – they just let you pick a particular pattern (corresponding to an ASCII code) and display it on the screen. If you can't change the ROM and you can't control pixels, how do you create graphics on these LCDs? It's easy!

There is a 64-byte block of RAM (random-access memory) that the LCD controller uses in the same way as it does the ROM-based character-generator. This is the CGRAM. When the controller receives an ASCII code in the range that's mapped to the CGRAM, it uses the bit patterns stored there to display a pattern on the LCD. The main difference is that you can write to CGRAM, thereby defining your own graphic symbols.

A character LCD that uses the HD44780 or the KS0068 controller allows for eight programmable characters. The character patterns must initially be programmed into the CGRAM of the LCD. First, the CGRAM must be selected by setting the CGRAM address for the character that is to be programmed. Each eight consecutive addresses, starting at CGRAM address zero, are assigned to one programmable graphical character. Each bit of the data at each address sets the pixel on (1) or off (0). The pixels on a particular row at the right of the character display have the lowest binary value. The rows of pixels start at the top and move down the character as the addresses increase in value, as shown in Fig. C.1.

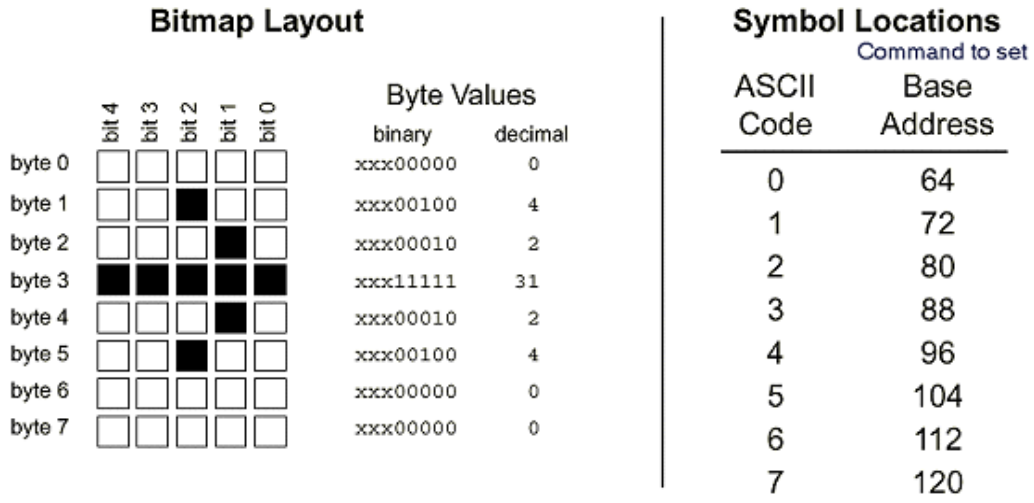

Figure C.1. Example bitmap for a programmable graphical character.

A web-based character calculator can be used to help determine the list of bytes that need to be written to the CGRAM. Be advised that the calculator in the second reference only generates seven bytes of data and does not specify data for the bottom row of pixels. Writing to CGRAM is a lot like moving the cursor to a particular position on the display and displaying characters at that new location. The steps are shown in Listing C.1.

Listing C.1. Steps to Program Graphical Characters

- Reset RS and R/W pins of the LCD to prepare the LCD to accept instructions.
- Set the CG RAM address by sending an instruction byte from 64 to 127 (locations 0-63 in CGRAM).
- Switch to DATA MODE by changing the RS pin to one.
- Send bytes with the bit patterns for your symbol(s). The LCD controller automatically increments CGRAM addresses, just as it does cursor positions on the display.
- To leave CGRAM, switch to COMMAND MODE to set the address counter to a valid display address (e.g., 128, 1st character of 1st line); the clear-screen instruction (byte 1); or the home instruction (byte 2). Now bytes are once again being written to the visible portion of the display.
- To see the custom character(s) you have defined, print ASCII codes 0 through 7.