Revision: 10May2019 (JFF)
Richard W. Wall, University of Idaho, rwall@uidaho.edu

1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

Project 6: Handshaking and LCD Control

Purpose

The purpose of this project is to investigate concepts involving parallel communications and handshaking. You will experiment with mechanisms required to pass information between unsynchronized systems using hardware and software techniques. You will build a project that consists of multiple C files. We will also look into using variables in software macros.

Minimum Knowledge and Programming Skills

1. Knowledge of C or C++ programming
2. Working knowledge of MPLAB IDE
3. ASCII character encoding
4. Use of logic analyzer or oscilloscope

Equipment List

1. Cerebot 32MX7cK processor board with USB cable
2. Microchip MPLAB X IDE
3. PmodTPH1 6 pin test point header
4. PmodTPH2 12 pin test point header
5. Oscilloscope / Logic analyzer

Software Resources

1. XC32 C/C++ Compiler Users Guide
2. PIC32 Peripheral Libraries for MBLAB C32 Compiler
3. Cerebot MX7cK Board Reference Manual
4. PIC32 Family Reference Manual Section 21: UART
5. MPLAB ® X Integrated Development Environment
6. C Programming Reference

Programming Concepts

Handshaking – What and Why

Handshaking refers to the mechanism 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 if the receiver is in a condition where it is able to receive the information and when the receiver has successfully received the information. 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 may be classified as either explicit or implicit. Explicit handshaking uses dedicated hardware control signals to indicate the impending action by the sender and readiness of the receiving device as shown in Figure 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 the data is no longer available.

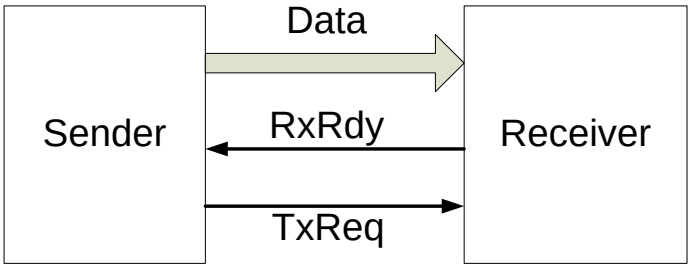

Figure 1. Interconnection for 4-phase handshaking

The timing diagram shown in Figure 2 illustrates the sequence of events 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 is permitted change from a 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 presents the data signals 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 time 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 the 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 fast the sender is able to send data, the data maximum exchange rate must conform to the specifications of the receiving device.

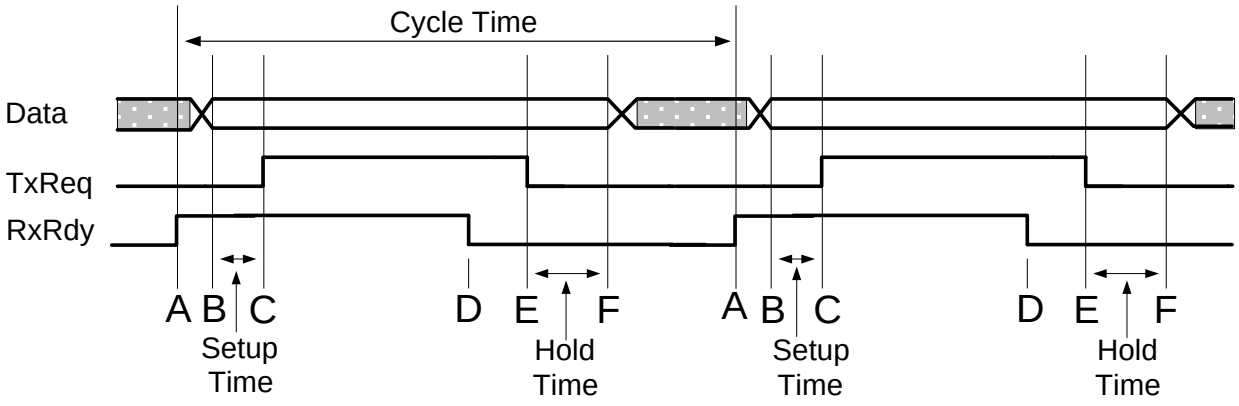

Figure 2. Timing diagram of four phase handshaking showing two complete cycles

Data sent in only one direction, as illustrated in Figure 1, is called simplex communications. For bidirectional data exchanges, the handshaking signal must be mirrored as illustrated in Figure 3. If the system does not support simultaneous bidirectional data exchanges, then it is called half duplex communication. The communications between the PIC32 and the character LCD is an example of half duplex communications. Full duplex communication refers to simultaneous bidirectional data exchange and requires separate data paths.

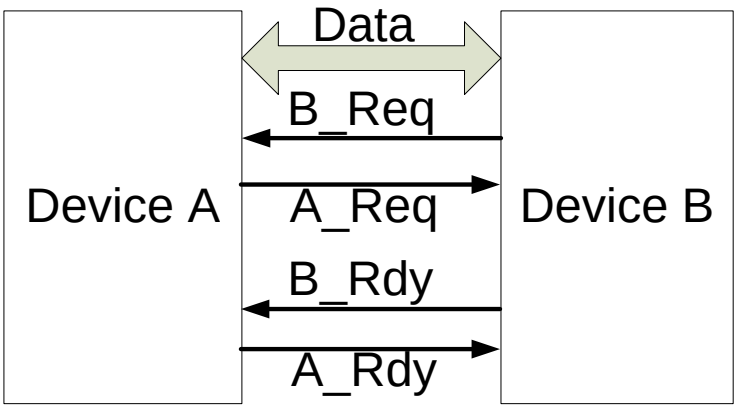

Figure 3. Interconnection bi-directional for 4-phase handshaking

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 an implicitly that the receiver is ready provided the timing constraints are met. Figure 4 and Figure 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. Figure 6 and Figure 7 show the timing for the four-phase handshaking with implied receiver ready and acknowledge. In both cases, the interval between B and C is called the data set up time and the interval between D and F is called the data hold time. In Figure 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.

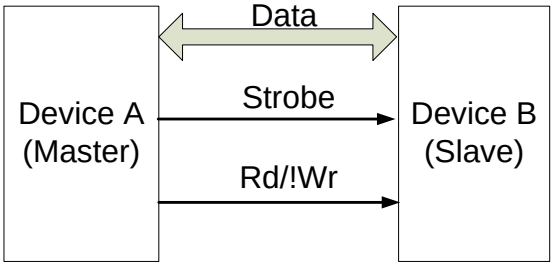

Figure 4. Half duplex using an ENABLE strobe and read / write control

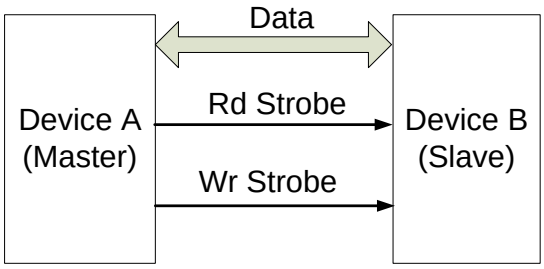

Figure 5. Half duplex using separate read and write strobes

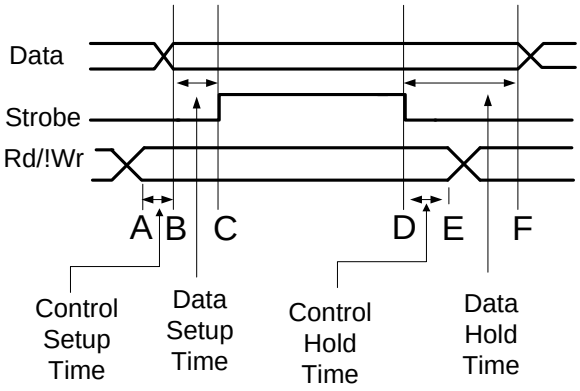

Figure 6. Timing diagram of four-phase handshaking with implied receiver ready corresponding to Figure 4.

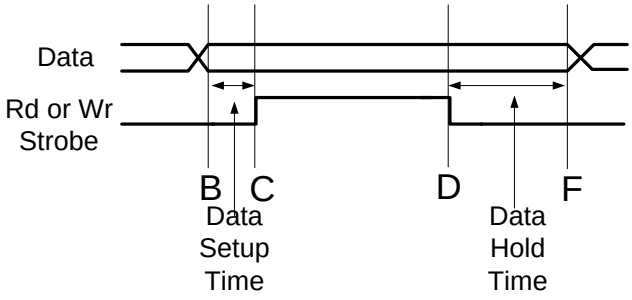

Figure 7. Timing diagram of four-phase handshaking with implied receiver ready corresponding to Figure 5.

There is also two-phase handshaking scheme. 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.

LCD Hardware Interface

The general hardware configuration for this project is shown in Figure 21 of Appendix A. The shaded components are for reference only and are used in Projects 7 and 8. Figure 8 shows the PIC32 Port and pin assignments for the interface to the Digilent PmodCLP character LCD. It uses an LCD controller with operating characteristics common to many character LCD devices. The PmodCLP Reference Manual references the data sheet for the Samsung KS0066. Although LCD units produced by various manufactures have different number of characters per

line and different number of display lines, they have an interface that has become a de facto standard. This interface will be discussed in detail in following paragraphs. The display configuration of the PmodCLP unit has two display lines with each line capable of displaying 16 characters.

There are three signals that the microprocessor uses to control the LCD module. The Enable signal is a read or write strobe depending on the whether the Read / !Write signal is high or low. (The "!" character indicates that the signal is active when the signal is asserted low.) The action of read or write is from the perspective of the microprocessor that serves as the master controller. The register select (RS) signal determines whether data is exchanged with the Instruction Register (IR) or Data Register (DR) inside the LCD Controller. In regards to the handshaking discussion above, this interface represents a half duplex communications scheme. Obviously the LCD handshaking conforms to what was described for Figure 4 and Figure 6. There are two versions of the PmodCLP. As noted on the data sheet, the Rev A module requires 5V power while other revisions use 3.3V power. The hardware configuration for this project is shown in Figure 21 in Appendix A. It shows that test point headers are placed in series with the LCD control lines. These test points will allow us to instrument the handshaking pins and measure the timing parameters.

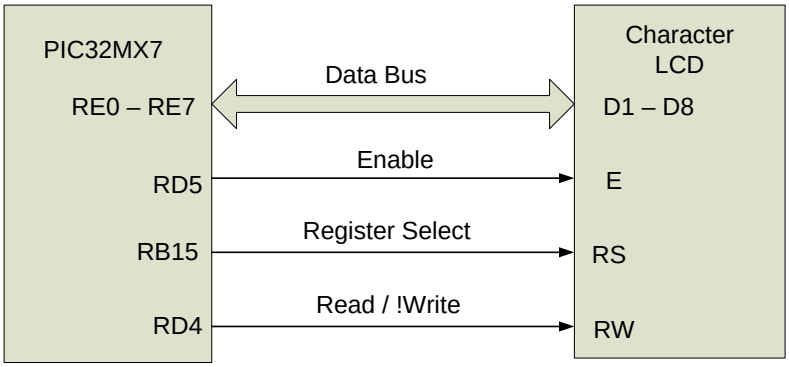

Figure 8. Diagram of the PIC32 - LCD interface

The read cycle timing provided in Figure 9 shows that the RS and the R/W signals must be asserted for a period equal to or longer than the time specified by the RW/RW setup time parameter, *tsu,* prior to the E signal being asserted high. Also, prior to asserting the E signal high, the processor IO pins used for the data lines must be set as inputs. The processor must assert the E signal high for a minimum time as specified the period, *tw*. After the read data output delay interval (*tD*), 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 plus the specified read data hold time (*tDH*).

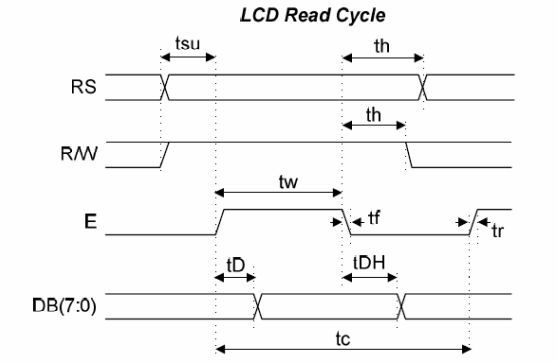

Figure 9. LCD Read cycle timing

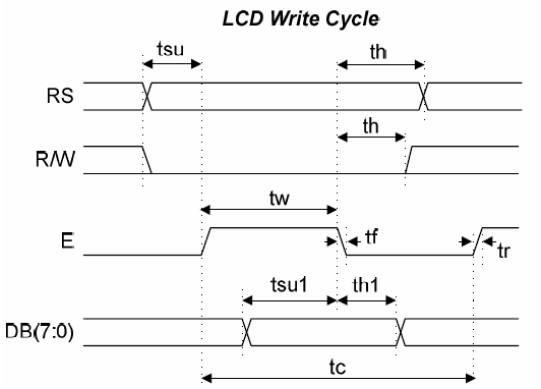

Figure 10. LCD write cycle timing

Figure 10 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/RW setup time, *tsu* parameter prior to the E signal being asserted high. The diagram shows that the processor is permitted to set the pins for output and write to data to the IO port any time so long as happens at a time equal to or greater than the write data setup time, *tsu1*. The processor is expected to maintain the data on the output lines constant for a period equal to or longer than specified by the write data hold time, *th1*.

The minimum time that the read or write operation can be repeated is set by the enable cycle time, *tc*. Table 1 provides the values for the timing parameters for both read and write cycles.

Table 1. LCD Signal Timing

Parameter	Symbol	Min	Max	Unit	Test Pin
Enable cycle time	tc	500		ns	E
Enable High pulse width	tw	220		ns	E
Enable rise/fall time	tr, tf		25	ns	E
RS, R/W setup time	tsu	40		ns	RS, R/W
RS, R/W hold time	th	10		ns	RS, R/W
Read data output delay	tD	60	120	ns	DB0-DB7
Read data hold time	tDH	20		ns	DB0-DB7
Write data setup time	tsu1	40		ns	DB0-DB7
Write data hold time	th1	10		ns	DB0-DB7

LCD Software Interface

The PmodCLP LCD Reference Manual provides essential information for programming and control of the character LCD. Like the PIC32 microprocessor, the character LCD has an initialization or configuration phase and an operational phase. One approach to software

development for this project is to partition the software into six functions or modules as shown in Figure 11. Detailed control flow diagrams for the LCD Read and the LCD Write functions are provided by Figure 12 and Figure 13. These LCD read and write functions implement the hardware interface and must be specifically written to match the hardware connections from the microprocessor to the LCD unit. The remaining four LCD functional blocks depicted in Figure 11 are not dependent on hardware connections.

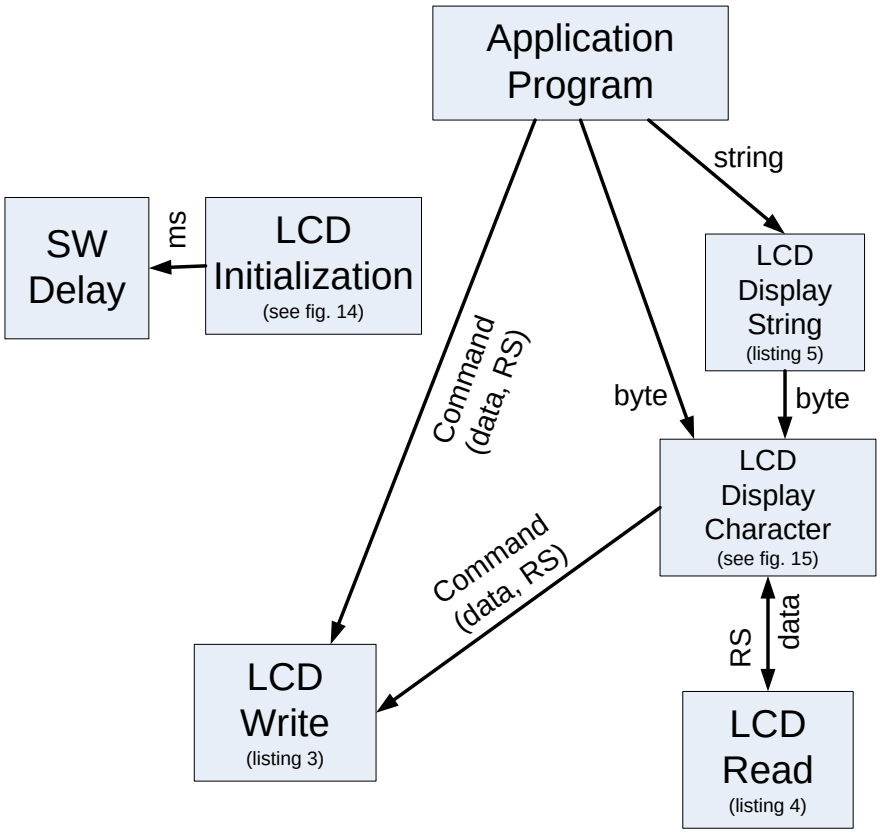

Figure 11. Data flow diagram for the LCD control

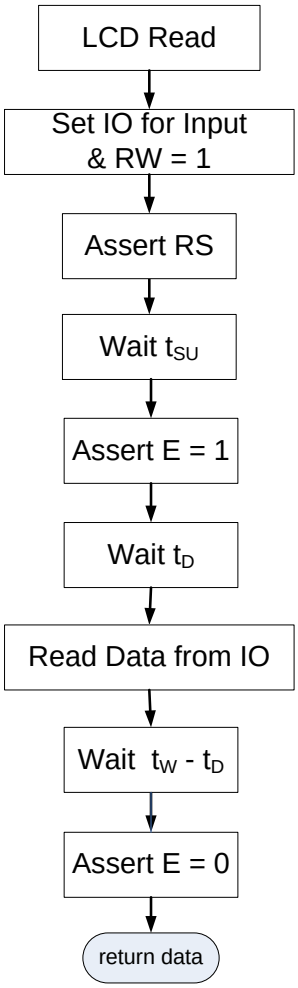

Figure 12. Control flow diagram for LCD read operation

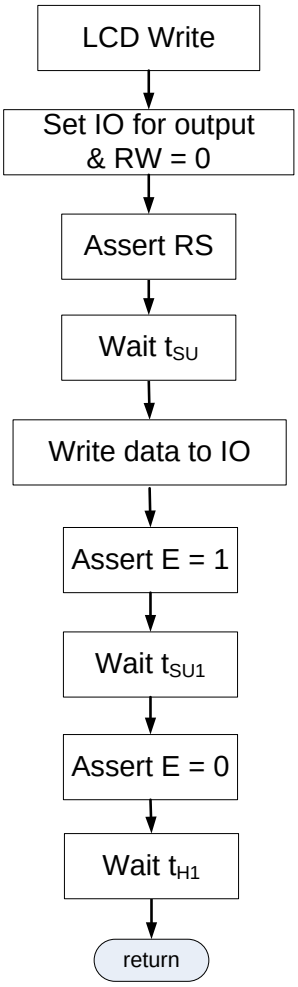

Figure 13. Control flow diagram for LCD write operation

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 register inside the LCD controller is accessed for 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 fixed text characters that are defined in the PmodCLP reference manual. By default, all data is written to and read from the DDRAM when the RS signal is high.

The Character Generator RAM (CGRAM) that is used to generate special symbols. (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.)

Figure 11 shows that a common LCD write function is called from multiple tasks. The LCD Write and Read are the two functions that provide the interface to the LCD hardware. Passing the value that RS is to be set to for the LCD read and LCD write functions allows access to either the display RAM or configuration register, limiting the hardware specific code to these two functions. This minimizes the amount of code resulting in reduced development time, improving portability of user code to different applications, and better memory resource utilization.

The configuration sequence is a series of commands written to the LCD after power up for CPU reset. Page 3 of the PmodCLP LCD reference manual lists the commands and the timing delays between writing the succession of initializing commands to the LCD. These delays can be implemented using one of the delay functions that you developed in Project 2. The values that are written to the LCD shown in Figure 14 configure the LCD for an 8 data line interface with a blinking cursor. The cursor address is reset to zero and 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.

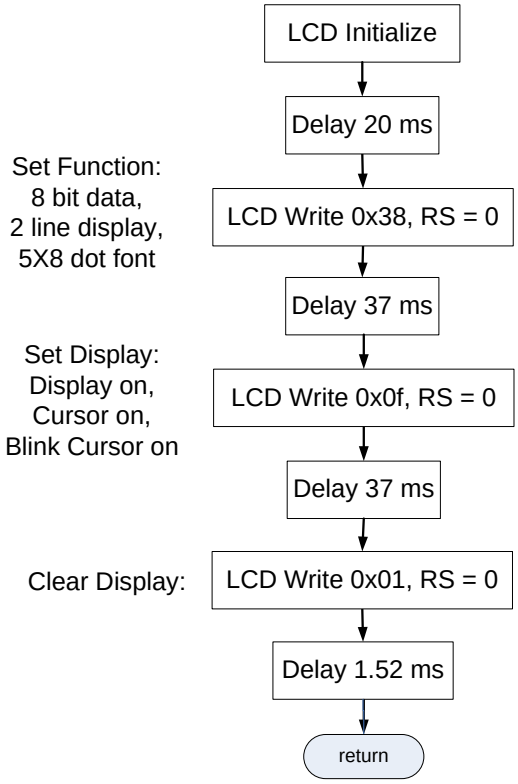

Figure 14. Control flow diagram for the LCD initialization[1]

After initializing, display characters can be written to the LCD. The PmodCLP module has two 16-character lines. Each LCD character position is assigned a DDRAM address. The 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 0x39 are not used. If a character is written to DDRAM address 0x0F, the LCD will properly display the character in the right-most position of the first line and increment the cursor address to the value 0x10. If additional characters are to be displayed, the cursor address must be first reset to 0x40. The process of repositioning the cursor position is described below.

[1] It is recommended that you use generous delay values, e.g., 50 ms and 5 ms.

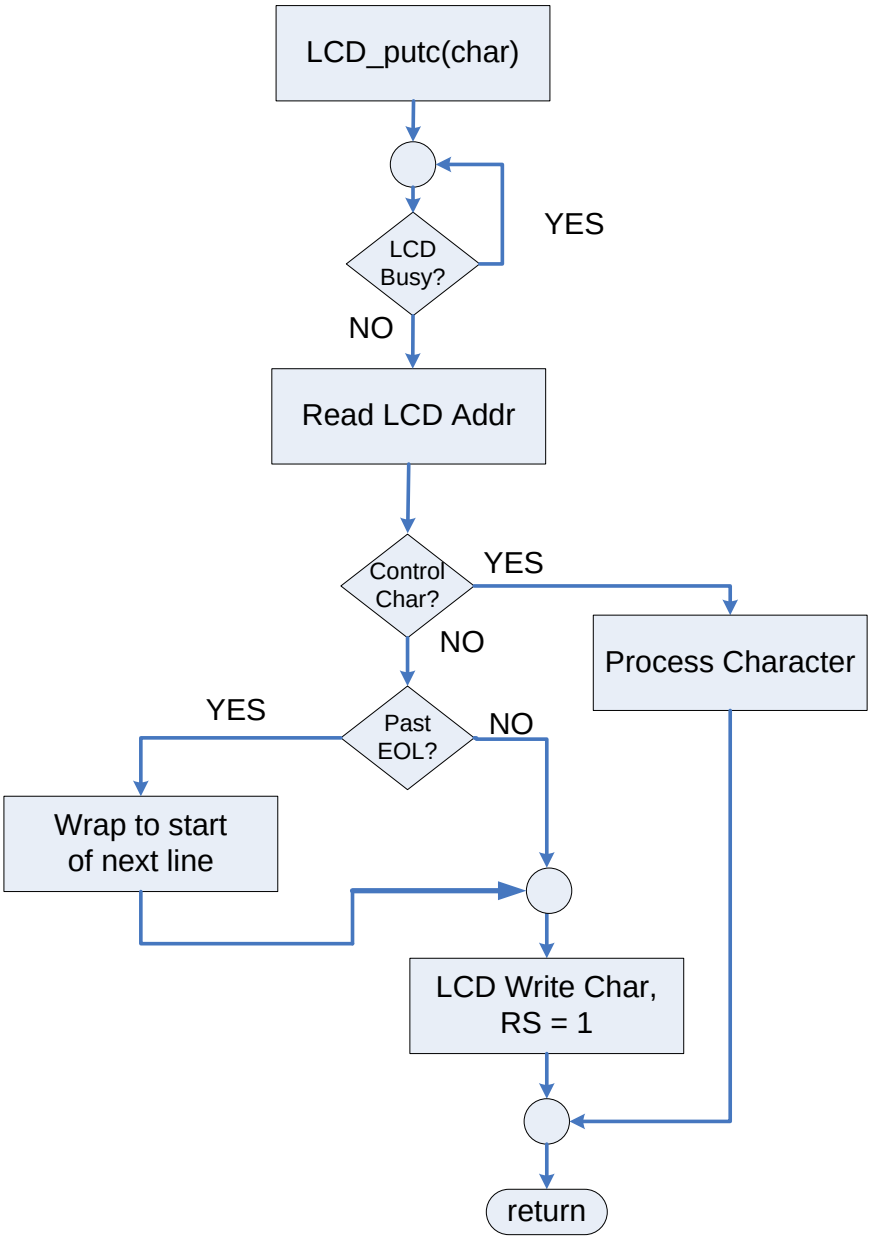

Figure 15. LCD display character control flow diagram – `LCD_putc()`

Figure 15 shows the control flow diagram for displaying a single character on the LCD using `LCD_putc()`. Initially, the status of the busy flag and the current cursor address[2] are read by reading from the LCD with the RS signal set low and the RW signal set high. 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.

[2] The cursor address is not valid until the LCD controller busy flag is a zero.

The next step of the CFD shown in Figure 15 requires testing the character passed to this function to determine if it is an ASCII control character. If the character is a carriage return (`'\r'`) then the function will use the writeLCD function to reset the LCD address to the start of the current line. If the character is a new line (`'\n'`) then the address is set to the start of the "next" line – switching from either the first line to the second or vice versa. Use of a switch-case statement allows for future support of additional ASCII control characters. Changing the LCD address (i.e., cursor location) is described in the following paragraph.

The final step requires determining if the cursor has incremented beyond the visible portion of the display. If the cursor value is between 0x10 and 0x3F, the cursor must be repositioned to the start of the second line or cursor address 0x40. 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. The writeLCD() routine will wait until the busy flag is set to zero before writing the 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 as well as setting the PIC32 IO pins for output when writing the ASCII value of the character to be displayed to the LCD. Similarly, if the cursor address is greater than 0x4F then it can be reset to the start of the first line (address 0x00) by writing 0x80 with the RS line low.

Most LCD modules require multiple milliseconds 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 43us to complete. The READ BUSY FLAG and ADDRESS operation requires zero time to complete and hence the LCD BUSY FLAG can be polled at the maximum rate provided by the minimum Enable cycle time shown in Table 1. The time to complete any write operation varies from one LCD manufacturer to another. The Enable cycle time parameter, t_C, diagrams clearly shows that there is a maximum rate that the LCD can be accessed. For the PmodCLP, the maximum rate is 2MHz.

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 statements such as LATxSET and LATxCLR instructions to control the state of the RS, RW, and E signals. The interface shown in Figure 8 provides the PIC32 port and pin specifications to allow the LCD control using this method. The code developer need only to write the software code to set the control pins to the required values in the sequence described by the control flow diagrams shown in Figure 12 and Figure 13. When using a high speed processor, such as the PIC32 running at 80MHz, additional delays may be needed to meet the minimum timing requirements. For example, Listing 1 is the C code to implement the LCD_read function shown in Figure 12

using the bit banging technique. The code in Listing 1 assumes that the setup and hold requirements are met.

Listing 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
      mPORTDSetBits(LCD_EN);          //Enable set high
      status = mPORTERead();          //Read BF and ADDR from LCD
      mPORTDClearBits(LCD_EN);        //Enable set low
      return (status);
}
```

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 Cerebot MX7cK processor board was designed to take advantage of this resource when interfacing to the PmodCLP 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 eight or sixteen bit bidirectional parallel IO 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 Figure 16. One can readily see the similarities among Figure 16, Figure 4, and Figure 8. Figure 17 illustrates the connection between the PIC32 PMP signals and the LCD. Note that Figure 17 conforms to the same pin assignments as shown in Figure 8.

The address bus is configured as a one-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. Refer to Appendix C of the Cerebot MX7cK Reference Manual to determine which pins are used by the PMP.

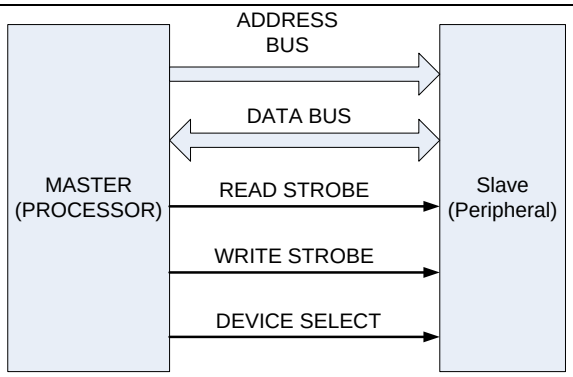

Figure 16. Architecture of a parallel data bus

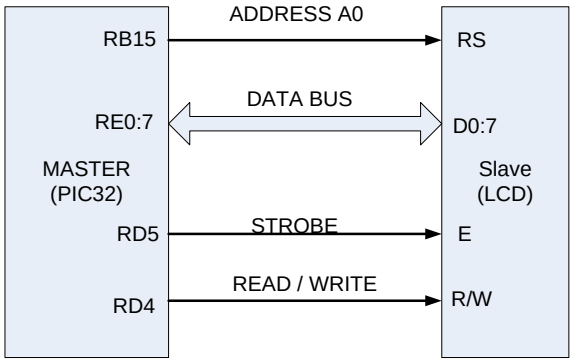

Figure 17. PIC32 PMP interface with LCD

The first step in using the PIC32 PMP interface is to configure the PMP for the LCD interface. Using the functions provided by the PIC32 Peripheral library (Section 17), the PMP is configured with the four parameters in the statements shown in Listing 2.

Listing 2. PMP initialization code for LCD interface [3]

```
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
mPMPOpen(cfg1, cfg2, cfg3, cfg4);
```

The first bit parameter in the variable *cfg1* enables the PMP peripheral. The second bit parameter enables both PMP read and write operations. The PMP_READ_POL_HI bit with the PMP_WRITE_POL_HI bit results in setting the least two significant bits of the PMCON register and 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 implements the handshaking operates as illustrated in Figure 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.

[3] It is recommended that you use generous PMP settings of 4, 15, and 4.

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 dash line in Figure 18 represents on peripheral bus clock period marked T_{PB}. The number of T_{PB} periods for the "B" period is specified by the PMP_WAIT_BEG_b ($1 \le b \le 4$) specified in the PMMODE configuration setting in Listing 2. Likewise the M indicates the number of T_{PB} periods that the 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 E indicates the number of T_{PB} periods that the RD/WR signal will retains its value PMP_WAIT_END_e ($1 \le e \le 4$). For the case shown in Figure 18, the three parameters would be: PMP_WAIT_BEG_4, PMP_WAIT_MID_6, and PMP_WAIT_END_2.

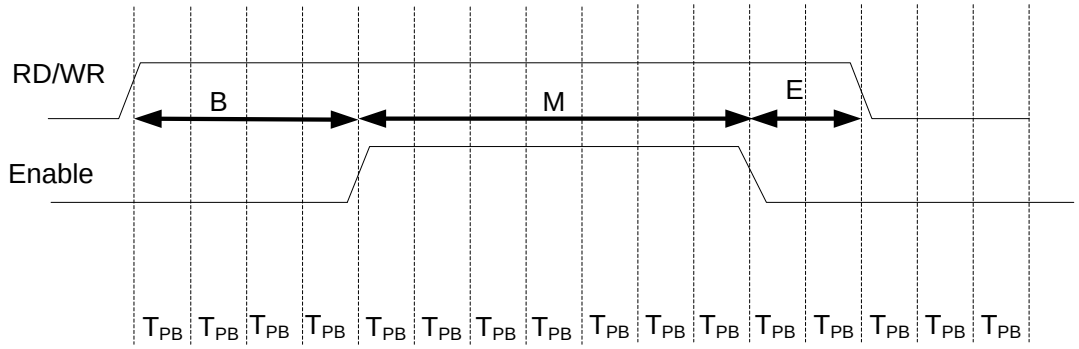

Figure 18. The PMP timing diagram for the beginning, middle and end wait periods.

The minimum settings for the three PMP WAIT intervals can be determined by dividing the value for t_{SU} on the LCD data sheet by T_{PB}. For example, assume that the peripheral bus clock is configured for 10MHz. The t_{SU} specified in the PmodCLP reference manual is 40 ns resulting in interval B equal to 0.4. 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. Hence 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} represent 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 500 ns. For the configuration described above, (B+[M+1]+E) * TPB = (5) * 100ns = 500ns thus 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 hence 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 tell 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 is shown in Listing 3.

Listing 3. LCD write function using the PMP[4]

```
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 4.

Listing 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.

Project Tasks

The first task in this project is to write the six functions depicted in Figure 11 and use them in the application program to alternate display of the two strings defined below using LCD_puts(), separated by a 5 second delay. Partition your code so that all LCD-specific functions and settings are contained in two files: LCDlib.h and LCDlib.c. The function, LCD_puts, was not previously discussed but is shown in Listing 5. The flow diagram for the LCD_putc function is provided in Figure 15.

```
char string1[] = "Does Dr J prefer PIC32 or FPGA??";
char string2[] = "Answer: \116\145\151\164\150\145\162\041";
```

Listing 5. Function to write a text string to the LCD

```
void LCD_puts(char *char_string)
{
```

[4] `busyLCD()` reads the LCD Instruction Register using `readLCD()` and returns the LCD busy flag.

```
   while(*char_string)  // Look for end of string NULL character
   {
      LCD_putc(*char_string);       // Write character to LCD
      char_string++;                // Increment string pointer
   }
} //End of LCD_puts
```

The second task is to verify through measurement that the handshake timing requirements specified in the PmodCLP Reference Manual are satisfied. This is done by modifying your program to continuously write a single character to the LCD using writeLCD(). Justify the measured parameters with the setting specified in the PMP configuration using *mPMPOpen*.

Project Testing

The first task is verified by observing the LCD screen and visually checking that the proper text is displayed on the proper lines of the LCD.

The second task requires measuring the timing of the LCD control and handshaking signals to determine that the LCD minimum timing specifications are satisfied. Figure 19 shows the timing of the RS, R/W and, E signals. Initially RS is low and RW is high when the E signal is pulsed high indicating that the LCD busy flag and cursor address is being read twice in a row. (Why?) The last enable pulse occurs when RS is set high and R/W is set low indicating a write operation.

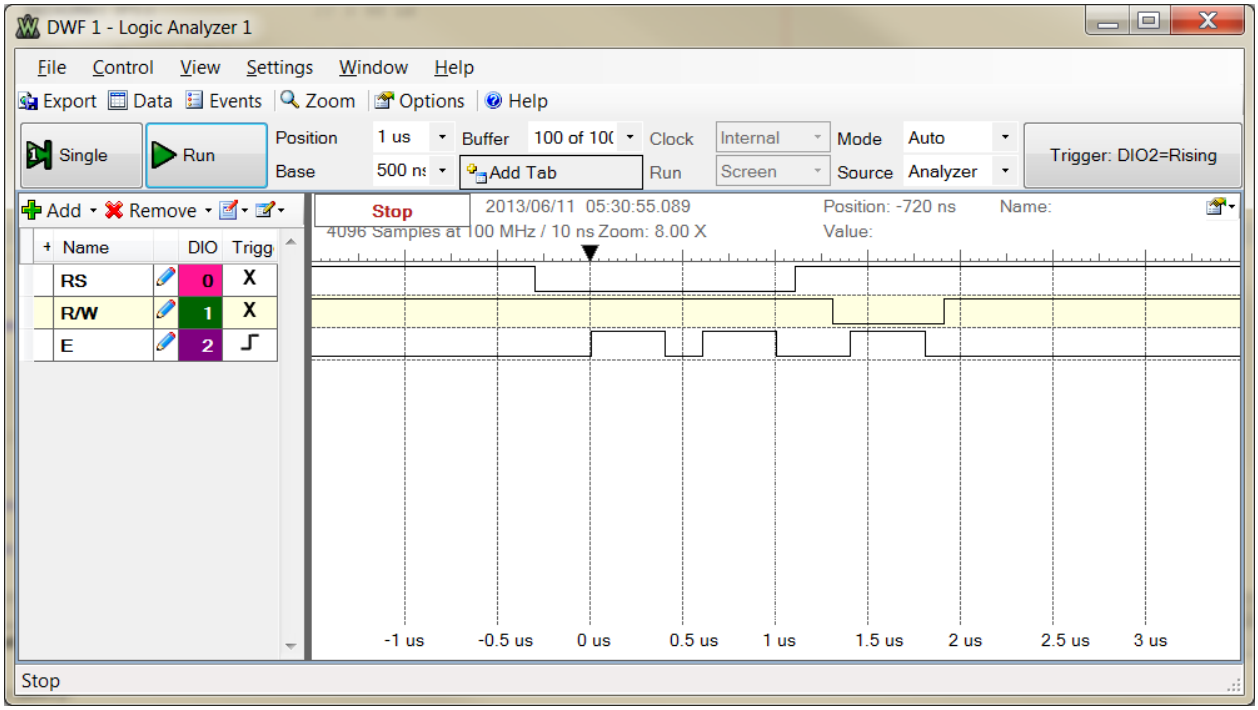

Figure 19. Screen capture of the LCD RS, RW and E signals for a single character write operation.

Figure 20 zooms in on the write operation to allow better measurement of the actual t_{su}, t_W, and t_H times. The falling edge of the R/W signal at marker for 1.1us and the rising edge of the E signal at marker for 1.2us clearly shows that PMP_WAIT_BEG is set to one times T_{PB}. The E signal is high for 0.3us (marker 1.2us to 1.5us) showing that the PMP_WAIT_MID was set to 2. (Remember that there is always at least one T_{PB} period for the strobe signal. The number of M wait period is added to the one T_{PB}.) Record the measurements in Table 2 to show that the minimum LCD timing specifications are satisfied.

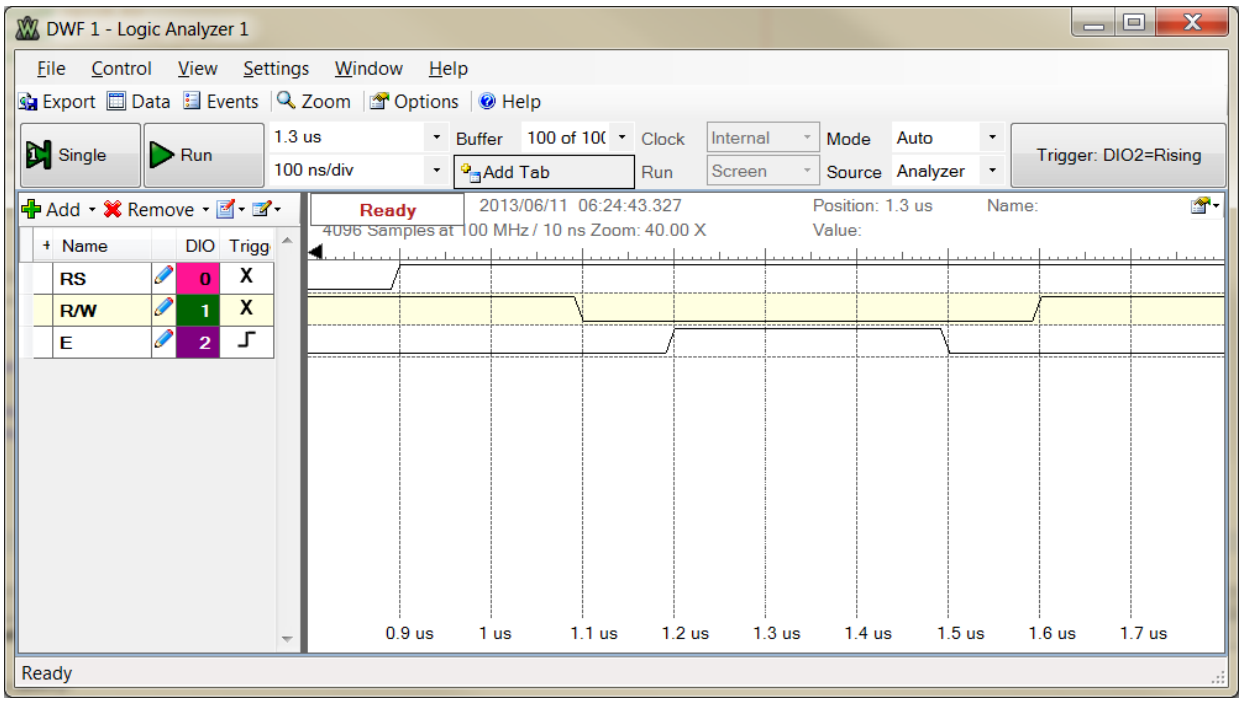

Figure 20. Screen capture of the LCD write operation.

Table 2. LCD Bus Timing Results. (Measure using repeated writes, similar to Fig. 19)

Parameter	Symbol	Min	Max	Units	Test Pin	Measured
Enable Cycle Time	t_C	500		ns	Enable	
Enable High Pulse width	t_W	220		ns	Enable	
RS, RW set up time	t_{SU}	40		ns	RS / RW	
RS, RW hold time	T_H	10		ns	RS / RW	

Appendix A: Project 6 Parts Configuration

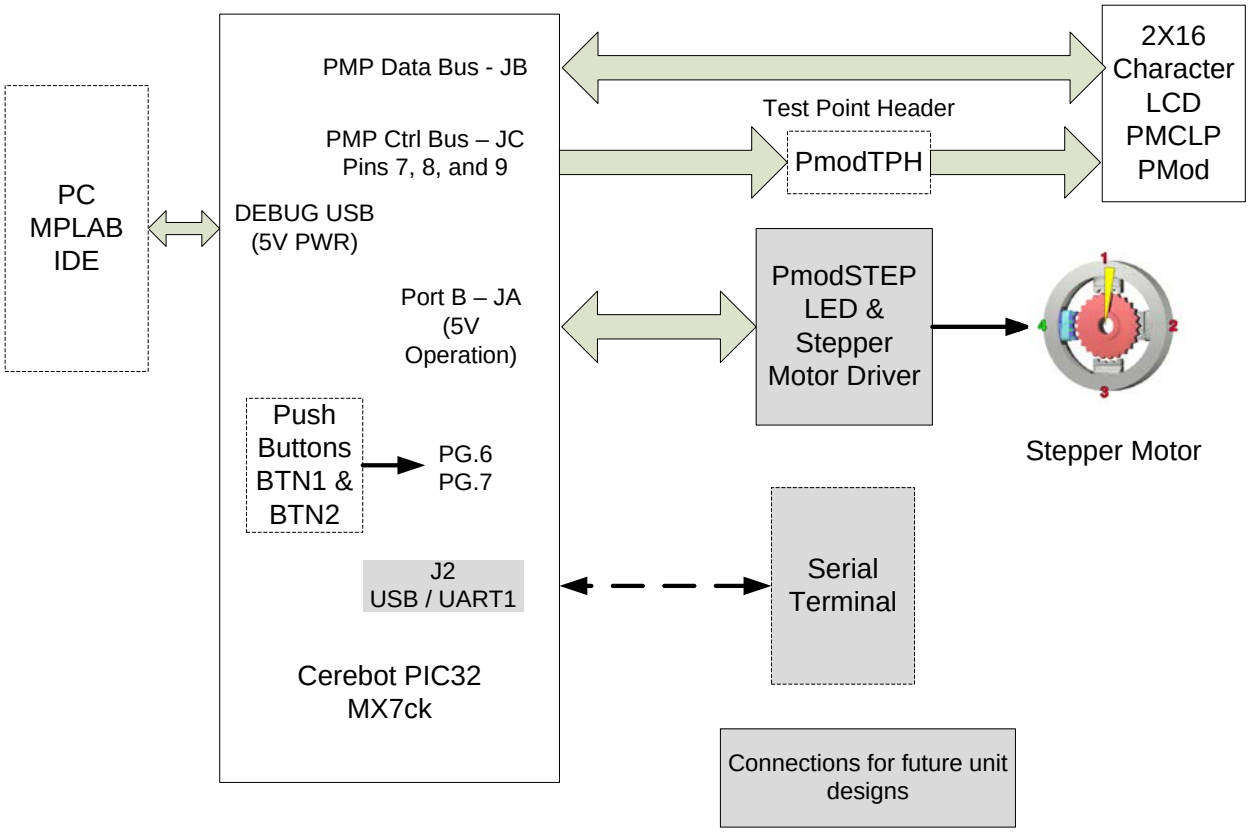

Figure 21. Block diagram of the equipment used in Project 6.

Richard W. Wall, University of Idaho, rwall@uidaho.edu
1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

Appendix B: PmodCLP

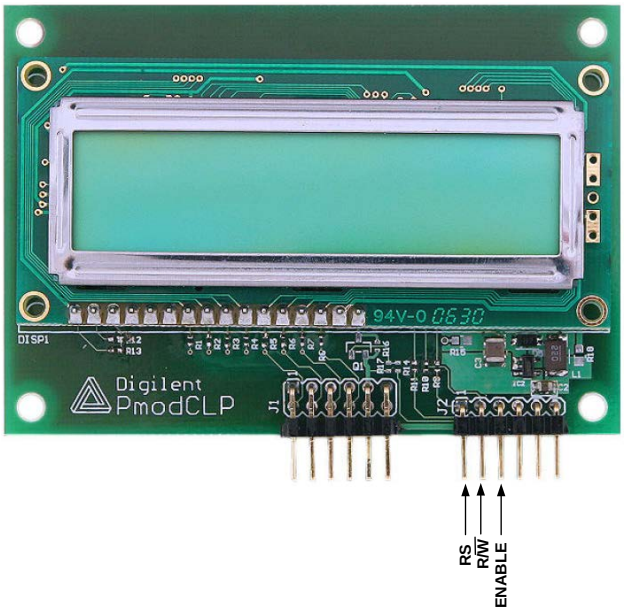

Figure 22. PmodCLP Character LCD pin identification

Richard W. Wall, University of Idaho, rwall@uidaho.edu

DIGILENT® www.digilentinc.com
1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

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 maps corresponding to the alphabet, numbers, punctuation, 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? Easy!

There's a 64-byte block of RAM (random-access memory) that the LCD controller uses in the same way as it does for the ROM based character-generator. 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 allow for eight programmable characters. The character patterns must be initially programmed into the CGRAM of the LCD. First, the CGRAM must be select 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 Figure 23.

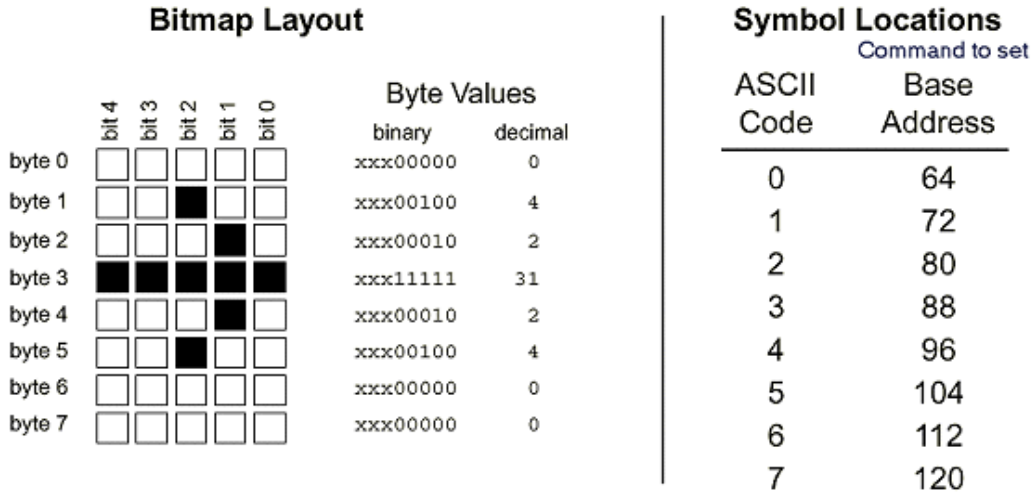

Figure 23. Example bit map for a programmable graphical character.

A WEB based character calculator can be use to help determine the list of bytes that need to be written to the CGRAM.[5] [6] 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 6.

Listing 6. 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 CG RAM).
- Switch to DATA MODE by changing setting the RS pin.
- Send bytes with the bit patterns for your symbol(s). The LCD controller automatically increments CG RAM 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.

[5] http://www.geocities.com/dinceraydin/lcd/charcalc.htm

[6] http://www.quinapalus.com/hd44780udg.html