Seth Cram

ECE341

Lab6 Prelab

10/12/2021

**Prelab 6: PMP and LCD Communication**

Goal:

To investigate concepts involving information passing between unsynchronized systems with handshaking using the parallel master port (PMP), liquid crystal display (LCD) controller, and LCD display.

Background Information:

This lab will be our first communication protocol, with parallel and asynchronous data transfer. Parallel meaning that we can pass multiple bits in a single transfer. Asynchronous meaning that there’s no clock involved so data can arrive at any time. Since we’ll pass information between two systems operating at different frequencies and asynchronously, we’ll need to utilize a handshake. Without a handshake, we run the risk of a setup or hold time violation and going metastable as a result.

We’re using an LCD module since its screen can convey much more instrumentation information than a simple LED. This is our first off-chip peripheral since it’s external to the PIC32. It’s connected to the PIC32 through wires. The external LCD module runs at around 270K Hz, with the PMP operating at 10M Hz. So, a handshake will be used to synchronize the different clock domains.

The PMP is for data transfers involving external peripherals. It operates as a slave to the PIC32 processor. For this lab, it also acts as the master of the LCD controller. The order of operations will be: our software running on the processor interacts with the PMP, then the PMP interacts with the LCD controller, and the LCD controller displays on the LCD screen. The signals that form the interface between the PIC32 and the LCD Controller are the bi-directional 8-bit data line, the 2-bit read/write and enable from PMP to LCD Controller, and the 1-bit address line, called the Register Select (RS), also from the PMP to the LCD Controller. The PMP also does the handshaking, every single data transfer, between the PIC32 and LCD module for us. The PMP can generate 16 address lines but we’ll only use a single one for this lab. Although not used in this lab, the PMP can also generate an interrupt.

The LCD screen is 16 characters on the top, locations 00-0F, and 16 on the next line, locations 40-4F. It has more display locations, but we won’t be using them for this lab. For this lab, we’ll configure it for ‘auto-incrementing’, so every time we write a character the address counter will advance to the next position. This means we’ll have to read the address counter to see if we’re at the end of a line yet, every character write.

Important components of the LCD controller include the Data Register (DR), Instruction Register (IR), address counter, display data RAM (DDR), and the busy flag. From the PMP, an RS value of zero corresponds to using IR, an RS of one corresponds to using the DR. The busy flag represents if the LCD controller is currently busy and can’t support a new operation. We can read the busy flag by polling the IR, but we also get the address counter, so we’ll have to use bit-masks to only retrieve the msb, which is the busy flag. In order to display on the LCD screen, we have to write to the address counter to move the cursor, then write data to the DDR. We’ll have to verify we’re displaying within the current range, since we won’t use scrolling to look at all the memory locations of the LCD screen.

This lab is also our first introduction to C strings. In C, a string is just an array of characters terminated by the null-termination character. They can be referred to as an array of characters or a pointer to characters.

Plan:

First, I’ll put the PMP initialization code from the lab 5 handout, listing 2, at the bottom of my system\_init(). Next, we’ll implement ‘writeLCD()’ from listing 3 and ‘readLCD()’ from listing 4, both from the lab handout. But, we won’t call these functions yet. To use ‘writeLCD()’, we need to write the ‘busyLCD()’ function. This function will ‘readLCD()’ for the IR, then use a bitmask to get the busy flag, which is the msb, and return it.

Finally, I’ll write the LCD initialization function and call it right below initializing the PMP. Deriving this from figure 14 in the lab handout, we’ll delay for 50ms, write 0x38 to the IR, delay for 50ms, write 0x0f to the IR, delay for 50ms, write 0x01 to the IR, and finally delay for 5ms. We can’t poll the busy flag during our initialization due to design constraints put in place, so instead we delay for enough time that the LCD controller should be ready for another write operation. Next, I’ll get Listing 5 from the lab handout and implement ‘LCD\_puts()’ for testing purposes. But, in order to use the function, we’ll first need to write ‘LCD\_putc()’ to write a character to the LCD screen. First, we loop until the busy flag isn’t set, read from the LCD IR, bitmask away the busy flag msb, then make sure the cursor is within the screen. If it isn’t, write to the LCD to set it to the start of a line, then check for the control characters using a switch statement. If there are none, write the passed in character to the LCD screen.



