ECE 354 – Lab 2: PIC–PLD Bus Interconnect

In this lab, we will connect the PIC to a PLD that will act as coprocessor to the PIC. You will develop an external bus interface that allows for reliable communication between the devices. This is the first step to building a complete PIC system that includes external memory (Lab 3). In addition to the bus interface, we will use a timer interrupts to control the periodic task of toggling of an LED. This will test the robustness of the bus implementation as processing can get interrupted at any time. In addition to developing PIC assembler code, you will need to implement significant amounts of functionality in the PLD. You may use VHDL, Altera schematics, or any other development tool of your choice for this purpose.

Tasks

For this lab you must complete the following tasks.

1. Your microcontroller should be programmed to generate a timer interrupt from timer 0 (TMR0) every millisecond. You should configure and use timer 0 to get as close to 1 millisecond as possible. This interrupt should control a COUNTER register that is incremented every time an interrupt occurs. When the COUNTER overflows (256th increment) an LED that is connected to bit 2 of port C should be toggled. As a result, the LED should turn on after about a quarter of a second and then turn after another quarter second (roughly 2 Hz). This should happen independently from any other code executed on the PIC. The code for controlling the LED should be included in the interrupt service routine (see lecture). In the lab report, explain how you determined the prescaler value for timer 0 and any other necessary settings.
2. Design a bus interface between the PIC and PLD. Be careful considering the fact that PIC code can be interrupted by the timer at any time. The design of the bus interface and the state machines is an important component of this lab. Do spend some thought on it before starting the implementation.
3. Ports A, B, C, and D of your PIC should be interfaced to a 7032 PLD. Ports A, B, and D are used to transfer address and data values between the PIC and PLD devices and individual bits on Port C can be used for control. Please be careful to configure data direction registers accordingly. Avoid using bits 2, 6, 7 on Port C for your PIC/PLD interface since they will be used by the toggle LED (bit 2) and UART (bits 6, 7) respectively.
4. After physically wiring the PIC to the PLD, write READ and WRITE macros or subroutines in assembler that allow for reliable data transfer between the two parts.
5. The PLD should implement the following functions[1]:
    1. Increment: write value to address 0x1 and read incremented value from 0x2.
    2. Bit count: write value to address 0x3 and read the number of 1-bits in byte from 0x4.
    3. Maximum: write value to address 0x5 and the read maximum written to 0x5 since power-up from 0x6.

[1] The capabilities of our PLD are very limited. Depending on how you implement the state machine, etc., there might not be enough room for all the functions listed. For this lab you must at least implement the bit count function. The other functions are optional, but the more you can implement the more interesting the lab will be (and it is only a bit more effort).

You can assume that writes to result addresses (0x2, 0x4, 0x6) are ignored (i.e., return 0x0). Reads from addresses that contain values that were previously written should yield these values. Reads and writes to other addresses should not occur and can be handled arbitrarily.

4. The main program of the PIC should do the following:
   - For the addresses 0x1, 0x3, 0x5 (in that order), read the COUNTER value and write it to the respective address. Please read the counter value every time before writing to the PLD.
   - For each implemented function (increment, bit count, maximum) read the appropriate value and print one line on the terminal with the values of the argument and the result. For example: "`Bit count of 0x4 is 0x1`". For the printing, all values should be read from the PIC. You can choose the way you print the result as long as it's clearly what it means.
   - Repeat printing these three lines indefinitely.

Demo

During the demo, your PIC should continuously communicate with the PLD, exchange data and results, and print these on the terminal. The main focus of this lab is on the bus. It's more important that your system has a reliable bus interface than a fancy co-processor function. You should be able to explain the results that are printed on the screen.

You must provide a logic analyzer demo of the bus transactions. You may set up your lab to show the bus transactions on the logic analyzer live or show previously generated printouts.

Things to keep in mind

Often it may appear that hardware is not working the way that is expected. An initial reaction would be to assume that the hardware is broken and needs to be replaced. Before going to find the TA keep the following questions in mind:

- Are all the inputs on the digital components driven by some value? Often, floating signals will cause chips, such as the 16F877, to operate incorrectly.
- Is there a simple test case I can try to verify functionality? Simply coding the entire lab at once or wiring the entire kit at once will likely lead to frustration. Instead, plan for simple tests that verify the functionality of specific components and software segments.
- Is the chip plugged into the board correctly? Please be careful plugging the MPLAB-ICD, the 16F877, and other components into the breadboard. Aligning the parts incorrectly will lead to broken pins and hours of debugging. Carefully plug and unplug each component from the breadboard.

Lab Report

Same lab report guidelines as for Lab 1. Please include an annotated logic analyzer printout.