Lab 4
Interrupts

Code and Demo Due: End of lab May 8 or May 9
Report Due: 11:59 PM May 11

Introduction

In this lab you will accomplish several goals:

- Implement interrupts on both a master and slave device
- Write an interrupt service routine
- Document your semester project through a report

All lab assignments must be done **individually**.

Requirements

This lab has you add interrupts to your existing system from Lab 3. You will expand your system by introducing a Verilog module that does nothing except generate an IRQ signal that causes an interrupt on your ARM board.

The expanded memory map for your CPLD system is now:

Address Range	Peripheral
0x74	Switches
0x6C	Bar LEDs
0x50–0x5F	RAM 2
0x2F	CPLD Board LEDs
0x13	Interrupt module
0x00–0x0F	RAM 1

CPLD Board

You must add an interrupt module to your CPLD system that connects to the shared buses. The interrupt module must accept read and write operations, that operate as follows:

- Reads return the state of the interrupt line, 0x00 if the interrupt module is not currently causing an interrupt and 0x01 if the interrupt module is currently causing an interrupt.
- Writes cause or clear an interrupt depending on the value written to the interrupt module. Writing a value of 0x01 to the interrupt module causes it to generate an interrupt. The interrupt module continues to cause an interrupt (hold IRQ active) until the value 0x00 is written, at which point the interrupt module deactivates IRQ.

An example module definition for your interrupt module is:

```
module interrupt_ctl(input ce_n, read_n, write_n, reset_n, inout [7:0] data_bus, output irq);
```

ARM Board

The program for your ARM Board must now be expanded to use and handle interrupts. For this lab you will be triggering an interrupt on an external pin, called an External Interrupt (EXTI) in the system documentation. Take a look at § 9.2 (and all of § 9) of the Reference Manual for details on EXTI. **Note that your microprocessor only has access to EXTI0 to EXTI4.**

Your interrupt handler may be written in C or assembly. You may make function calls in your interrupt handler, but these should be limited. Your interrupt handler has the following tasks:

1. Disable interrupts
2. Clear the pending bit for the interrupt
3. Notify the interrupt source the interrupt is being handled
4. Increment a global counter
5. Enable interrupts

You do not need to save system state, as the microprocessor does this for you already.

Your interrupt handler must have a specific name, similar to `EXTI?_IRQHandler`, where `?` is replaced by an appropriate number. Define (you don't need to declare) your ISR as `void EXTI?_IRQHandler(void)`. If done correctly, the compiler and assembler will use your function as the ISR for the external interrupt.

To setup interrupts on your ARM board, follow these general directions

1. Enable clocks for the interrupt pin and SYSCFG
2. Initialize the interrupt pin (`GPIO_Init`)
3. Configure the interrupt pin as an interrupt source (`SYSCFG_EXTILineConfig` and `EXTI_Init`)
4. Configure the NVIC and interrupt priority (`NVIC_Init`)

Don't forget to enable interrupts.

Augment your LCD display to include the interrupt count. Some example displays include:

- `3DR000`: The value `0x00` was read from address `0x3D` and there have been no interrupts.
- `02W224`: The value `0x22` was written to the address `0x02` and there have been four interrupts.

Schematic Diagram

To help document your system, create a schematic diagram of your overall system. Your schematic should indicate all the major electrical components and the pins selected. The documentation created for Lab 2 will help you when creating the schematic diagram. This should be an electrical schematic, so include the resistors, LEDs, switches, and memory, but do not diagram the internals of the CPLD design. You may treat your ARM board and CPLD board each as a single block. Your schematic may be done with schematic capture software or *neatly* by hand.

Demonstrations

You must demonstrate your entire system on the last week of classes during your lab period, May 8 or May 9.

Lab Report

The final documentation for your lab consists of a Lab Report, which indicates how you designed your system and what you accomplished this semester. A suggested outline, with section contents, is:

1. Introduction: The overall goal of your system. Include any information a reader may need to understand your report (e.g., memory map, CPLD logical diagram).
2. Schematic Diagram
3. Methodology: Describe how you went about building your system and your design process.
4. Implementation: Describe your design choices. Provide enough information to allow another person to duplicate your project, without resorting to listing all your code. Short code sequences are appropriate if it improves clarity. Questions you might answer include:
   - What are your SPI settings?
   - What is the data format for SPI transactions?
   - How do you handle reads of CPLD modules, which require multiple SPI transactions?
   - Does your decoder perform partial or full address decoding?
   - How is the information on your LCD displayed?
5. Conclusion: Describe the status of your project and indicate what you learned through the lab exercises.

General guidelines for the Lab Report:

- Be descriptive, but concise. Prose quality (spelling, grammar, etc.) counts.
- Use section titles and figure and table captions. Reference and describe all figures and tables.
- Recommended formatting: single spaced, 1" margins, 11 point font, page numbers, title page.

Results

Your work for this lab, and the semester, will be evaluated based on:

- Your Lab Report
- Source code (C, assembly, and Verilog) that you submit through Learn
- Demonstration of your working system to your lab TA on the due date

Grading

Your lab will be graded based on:

- 40 points: Lab Report
- 40 points: Demonstration
- 20 points: Code quality and neatness

Hints

- Design and test your system in steps.
- Use the laboratory equipment to verify what your system is doing. For example, if your interrupts are not working, use the equipment to find out if your CPLD is not driving the IRQ line or if your ARM board is ignoring it.