LAB 4: INTERRUPT PROCESSING IN C

INTRODUCTION

The previous labs worked with simple input/output (I/O) devices using ***program-controlled I/O***; the programs continuously monitored each "device" to determine when to take action. The primary disadvantage of using this approach to handle I/O devices is that the CPU spends considerable amounts of time scanning devices to see if they require attention, whether or not there is any device activity. The purpose of this lab is to learn how to design C programs for the STM32L1xx microcontroller to handle devices in ***interrupt-driven I/O*** mode. This allows the CPU to perform tasks other than monitoring the devices; these devices will signal the CPU via interrupt request signals when they require attention. Interrupt-driven I/O is used heavily in embedded system applications, with interrupt signals coming from external devices, data acquisition hardware, timers, etc. For this introductory lab, we will simulate the operation of two external devices by using switches to issue interrupt requests. The application program will comprise a main task, which runs continuously, and two interrupt service routines, which are executed when interrupt requests are detected.

STM32L1xx MICROCONTROLLER INTERRUPTS

Review the accompanying presentation on the course web page for an overview of:

- Program interrupts in Cortex-M processors
- Prioritized, vectored interrupts and vector table
- The Cortex-M Nested Vectored Interrupt Controller
- Configuring GPIO pins to allow external devices to request interrupts
- C program setup for interrupt-driven operation

LAB EXPERIMENT

For this lab, the "main program" is to continuously display counting sequences on two 4-bit (4 LED) displays, with the first sequence changing at twice the rate of the second. The first sequence will always be increasing (0 to 9 and repeat). The second sequence will initially be increasing. Two GPIO pins, PA0 and PA1, are to be configured as external interrupt signals EXTI0 and EXTI1, respectively, and the two LEDs on the Discovery board are to help monitor when interrupts have occurred. If an interrupt occurs on pin PA0, the state of one of the on-board LEDs is to be toggled and the sequence is to change from increasing to decreasing. If an interrupt occurs on pin PA1, the state of the other on-board LED is to be toggled and the sequence is to change from decreasing to increasing. The user button on the Discovery board is to control PA0, and one of the bits of the *Waveforms* Static IO window is to be configured as a switch, to control PA1.

PRE-LAB ASSIGNMENT

Reading

Review the Lab 4 presentation slides, which provide information on working with interrupts in STM32L1xx microcontrollers, and your ELEC 2220 notes on interrupts. Chapter 12, section 12.1, of the Cady and Chapter 9 of the Valvano text (from Summer 2014) overviews of interrupt-driven operation. Prof. Nelson's ELEC 2220 course notes are also available on http://www.eng.auburn.edu/~nelsovp/courses/elec2220/.

Hardware Design

The I/O port connections to the "peripheral devices" should be made as listed in Table 2. Parallel input/output port GPIOC is to be used to display the counting patterns on two sets of 4 LEDs (in the *Waveforms* Static I/O window), and to drive the two on-board LEDs. The on-board User Button and a push-button switch in the *Waveforms* Static I/O window are to drive pins PA0 and PA1, respectively. LEDs in the Static I/O window should also be connected to PA0, PC8 and PC9, to monitor these devices

Parallel Port Pins	Connected Devices
PA0	User Button
PA1	Static I/O window "pushbutton"
PC3 – PC0	1st 4-bit LED display
PC7 – PC4	2nd 4-bit LED display
PC8	LED toggled in PA0 ISR
PC9	LED toggled in PA1 ISR

Table 2. Parallel input/output port connections.

To demonstrate the operation of the program, the logic analyzer and oscilloscope are to be used to capture and show counting patterns and the states of the two LEDs as interrupts occur.

In your laboratory notebook, sketch a diagram that corresponds to the connections described in Table 2. Show details of how the microcontroller pins and EEBoard lines (test instruments) are to be connected. ***Do this prior to lab.***

Software Design

Review the earlier labs to recall how to initialize and access GPIO ports, and review the above information and pre-lab reading about interrupt processing. Remember to thoroughly comment your program.

This program is to comprise a main routine and two interrupt service routines.

1. The main program should execute in a continuous loop, displaying separate counting sequences on two sets of 4 LEDs in the *Waveforms* Static IO window.

2. The first count is to be increasing (0 to 9 and repeat) on the four LEDs connected to PC3-PC0. This count should change approximately once per half second, and continue throughout the duration of the program.
3. The second count is to be displayed on the four LEDs connected to PC7-PC4, and is to change approximately once per second. The second count is to initially be increasing.
    a. If the on-board User Button is pressed (drives pin PA0), the state of the LED connected to PC8 should be toggled, and the second count sequence should change from increasing to decreasing (or continue to be decreasing if that is the current sequence.)
    b. If the Static I/O window push button is activated (drives pin PA1), the state of the LED connected to PC9 should be toggled, and the second count sequence should change to increasing (or continue to be increasing if that is the current sequence.)

One possible way to implement the direction control is to have the interrupt service routines set or reset a global variable that indicates the direction of the second counter, with all counting handled by the main program or a separate counting function. The time delay can either be implemented within the main program, or called as a separate function.

In your laboratory notebook, record the following ***prior to lab***.

1. flowcharts for program and the two interrupt service routines
2. draft program and the two interrupt service routines (or directions to content on H:drive)
3. a plan for testing the two counters

Hints on setting up interrupts:

1. Write your interrupt handlers, using function names from the interrupt vector table in the startup code file.
2. Near the beginning your main program, call a function to initialize the GPIO pins, configure PA0 and PA1 as external interrupts, and configure the NVIC module to handle for these two interrupts.
3. Enable CPU interrupts, *after everything has been set up* and the program is ready to begin processing interrupt requests.

LAB PROCEDURE

1. If not already done, ensure that you have a ground connection between the Discovery board and the EEBOARD.

2. Make the connections listed in Table 2. To the EEBOARD Digital I/O block, connect the GPIOC pins for the two 4-LED displays and the two on-board LEDs, and in the *Waveforms* Static I/O window, configure these bits as LEDs.

3. As shown in Table 2, connect an EEBOARD Digital I/O signal to GPIOA pin PA1. In the *Waveforms* Static I/O window, configure that Digital I/O as a pushbutton switch. This switch should be initially in the logic 0 (inactive) state. If you wish, you may also connect PA0 to one of the Digital I/O signals, and configure it as an LED to monitor its state.

4. Double-check all connections before downloading your program to the Discovery board.

5. Enter, compile and download your program. Execute your program, verifying that the correct counts are displayed on the two digits, and that the counting sequence is correctly altered by the two interrupts.

6. Set up the logic analyzer to capture and display the counting sequences on the two 4-LED displays, plus the states of the two interrupt signals and the LEDs toggled by the interrupt service routines. (Creating two 4-bit "buses" in the logic analyzer window would facilitate watching the counting sequences.) *Note that you can use the same Digital I/O connections for both logic analyzer and static I/O windows.* Try to use the interrupt signals to "trigger" the logic analyzer so that you can verify that the LEDs and counting sequence change correctly as each interrupt activates.

7. Set up the four oscilloscope channels to capture the states of the two buttons and the two LEDs toggled by the interrupt service routines. Set up the triggering condition to trigger the display when PA0 is activated, and then verify that the corresponding LED is changed by the service routine. Then repeat, but with the display triggered when PA1 is activated. This illustrates how one might verify that interrupt signals have occurred, and that the corresponding interrupt service routines have been executed.

8. Demonstrate your working programs to the lab instructor.

LAB REPORT INFORMATION

1. Briefly describe your hardware design. Attach a schematic diagram.
2. Attach a printout of your C program, including well thought through comments.
3. Discuss your results. Attach a logic analyzer screen capture showing the count changing direction in response to an interrupt signal, and an oscilloscope screen capture showing an interrupt signal and the corresponding LED being toggled.