CS/EE 5780/6780 Handout #4
Spring 2007
Myers

LAB #5: Keypad Interface Using Interrupts

Lab writeup is due to your TA at the beginning of your next scheduled lab. Don't put this off to the last minute! There is pre-lab work to complete before the start of the next lab. **NO LATE LAB REPORTS WILL BE ACCEPTED.**

1 Objectives

- Redesign the hardware and software interface between a keypad and a microcomputer using interrupts.

2 Reading

- Read Chapter 4.
- Read section 8.1 about keyboard scanning and debouncing.

3 Background

In this lab, you will redesign the keypad interface from the last lab using interrupt synchronization. There are two advantages to interrupts in this application. First, placing keypad input into a background thread, allows the main program to execute other tasks while waiting for key entries. Second, interrupts give the ability to create accurate and bounded interface delays. First, you can use real-time interrupts to implement periodic polling. Second, you can read ahead (Section 6.1) and use input capture. Third, you can read ahead (Section 6.2) and use output compare periodic polling. The choice is yours.

4 Parts

Depending on how you choose to implement the interrupt, you may need some external logic gates which are available in the ECE lab for purchase.

5 Software

Below is a prototype for your keypad device driver:

1. Data structures: global, private (accessed only by device driver, not the user)
   `OpenFlag` - Boolean that is true if the keyboard port is open, initially false, set to true by `KeyOpen`, set to false by `KeyClose`, should be in static storage.
   `Fifo` - FIFO queue with `Clr`, `Put`, and `Get` operations. This should be dynamic storage created by `KeyOpen`.

2. Initialization routines (called by user)
   `KeyOpen` - Initialization of the keyboard port, sets `OpenFlag` to true, initializes the hardware, returns error code if unsuccessful (hardware non-existent, already open, etc.), no input parameters, output parameter is error code.
   `KeyClose` - release of keyboard port, sets `OpenFlag` to false, returns error code if not previously open.

3. Regular I/O calls (called by user to perform I/O)
   `KeyIn` - input an key value from the keyboard port, tries to `Get` a byte from the `Fifo`, returns data if successful, returns error code if unsucessful (device not open, `Fifo` empty, etc.).
   `KeyStatus` - returns the status of the keyboard port, returns true if a call to `KeyIn` would return with a key (i.e., there is data in the `Fifo`), returns false if a call to `KeyIn` would not return right away, but rather it would wait.

4. Support software (private, not directly accessible by the user)
   There is one interrupt service handler: `KeyHan` which should occur either every 20 ms (if periodic polling is used) or whenever a key is detected (if a hardware interrupt is used), scan, debounce, and deal with 1 or 2 key rollover.

6 Tasks

1. Prepare a schematic for your design including all components (note, you will need pull-up resistors on all the keypad inputs).
2. Rewrite your low-level keypad device driver using interrupts.
3. Combine your security code access program with a program that counts on the Project Board LEDs. In other words, your security system should count on LEDs when no keys are being pressed. But, it should accept a security code when there are key presses.
4. Connect your circuit and debug your software.

7 Prelab

You should complete the first 3 tasks before your lab section.

8 Writeup

1. Your hardware schematic.
2. A printout of your C code.