Lab 9c

Using Interrupts to Detect Inputs on Simulated Hardware

Due: Week of April 26, before the start of your lab section

This is an individual-effort project. You may discuss concepts and syntax with other students, but you may discuss solutions only with the professor and the TAs. Sharing code with or copying code from another student or the internet is prohibited.

Due to the circumstances of this semester, we will not be using Altera single board computers for labs. Instead, we will use a simulated single board computer (SSBC).

In this assignment, you will use interrupts to implement a simple calulator on the Simulated Single Board Computer

The instructions are written assuming you will edit and run the code on your account on the *csce.unl.edu* Linux server. If you wish, you may edit the code in a different environment; however, you will not be able to link an executable except on your account on the *csce.unl.edu* Linux server.

1 Getting Started

The accompanying document, *ssbc.pdf* describes how to use the SSBC library: how to interact with the simulated SBC and how to use its inputs and outputs in a program.

Download *ssbclab-interrupts.zip* or *ssbclab-interrupts.tar* from Canvas or ~cse231 on *csce.unl.edu* and copy it to your account on the *csce.unl.edu* Linux server. Once copied, unpackage the file. You will find four files:

ssbc.h The header file for the SSBC library.

Makefile The Makefile to build the programs for this lab. You can build the programs individually, or you can build all of them using the command `make all`.

interrupt-calculator.c Starter code forthe lab. Run the command `make interrupt-calculator` to build the program, and then run it with the command `./interrupt-calculator`.

Note: **Do NOT use `printf` to print debugging information. Use `ssbc_print` instead.**

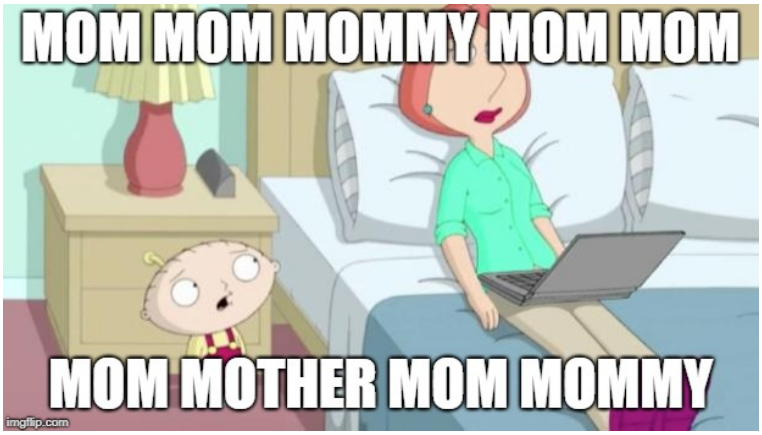

Figure 1: Interrupts. Image by 20th Century Fox Television

2 Use Interrupts to Detect Inputs

In a more complex program, polling inputs can be unwieldy. An alternative is interrupt-driven programming, in which your program responds to hardware interrupts rather than repeatedly checking whether an update is available. A mix of polling and interrupts is common when only some inputs generate an interrupt. In this lab, your program will detect number button presses by reacting to a simulated hardware interrupt.

The simulated SBC simulates hardware interrupts with operating system signals. Specifically, if your program sets the "IRQ enable" bit, then the SSBC libary will raise `SIGUSR1` signal whenever a number button is pressed. The SSBC library initially sets this signal to be ignored; however, if you register a signal handler with `sigset` then your program can react to the signal as though it were a hardware interrupt.

Edit *interrupt-calculator.c* to write a program that uses the SSBC's input and output registers to meet the following specification:

- Functionality: implement a simple 2-function decimal integer calculator using the same functional specification as the polling calulator. For your convenience, this specification is reproduced in Section 3.
- Implementation
  - Write a signal handler to take the appropriate action when a number button is pressed, based on the program's current state.
  - Register that signal handler to execute whenever the `SIGUSR1` signal is raised.
  - Do *not* poll the number keypad's register. (You will still need to poll the toggle switches' register.)
  - You may use any built-in C operations.
  - *Hint:* Much of the code you wrote for the polling calculator can be reused, but the overall structure will differ.

Turn-in and Grading

When you have completed this assignment, upload *interrupt-calculator.c* to Canvas.

This assignment is worth 20 points.

Rubric:

_____ **+0.33** The displays are blank when all toggle switches are in the "off" position

_____ **+4.5** It is possible to build *operand*1 in accordance with specification.

_____ **+0.33** When toggle switch 3 is moved to the "off" position, *operand*1 takes on the value built.

_____ **+4.5** It is possible to build *operand*2 in accordance with specification.

_____ **+0.33** When toggle switch 2 is moved to the "off" position, *operand*2 takes on the value built.

_____ **+2.5** When toggle switch 1 is moved to the "on" position, the sum of *operand*1 + *operand*2 is displayed without leading 0s.

_____ **+2.5** When toggle switch 0 is moved to the "on" position, the difference of *operand*1 − *operand*2 is displayed without leading 0s.

_____ **+5** Number button presses are detected by handling the `SIGUSR1` signal.

3 Functional Specification

- Initially, *operand*1 and *operand*2 will hold the value 0.
- When all toggle switches are in the "off" position, the seven-segment displays will be blank.
- When toggle switch 3 (the switch that is toggled by the 'a' key) is moved to the "on" position, the program starts to build *operand*1. When toggle switch 3 is moved to the "off" position, *operand*1 takes the value that was built.
- When toggle switch 2 (the switch that is toggled by the 's' key) is moved to the "on" position, the program starts to build *operand*2. When toggle switch 2 is moved to the "off" position, *operand*2 takes the value that was built.
- When building an operand:
  - The right-most seven-segment display (the "ones" place) will always display a digit. The remaining seven-segment displays will not display leading 0s.

  - The value being built will initially be 0, and so 0 will be initially displayed.
  - When the first number button is pressed after starting to build an operand, the "ones" place will display that digit. For example, if the user presses '5', then 5 will be displayed.
  - There will never be leading 0s in the "tens," "hundreds," and "thousands" places.
  - When the second, third, and fourth number button is pressed (if a second, third, or fourth number button is pressed) then the existing digits will shift one decimal digit to the left, and the newly-entered number will go in the "ones" place. For example, if the user presses '5', '3', '0', and '9', then the display will show these values in sequence:

    0
    5
    53
    530
    5309

  - There is no way to build a negative operand.
  - Except as noted in Section 4, you may assume that operands are representable with four or fewer decimal digits. Unless you implement the bonus, we are not specifying the correct behavior if the user presses a fifth number button.

- Number button presses will be ignored except when building operands.
- When toggle switch 1 (the switch that is toggled by the 'd' key) is moved to the "on" position, the seven-segment displays will display the sum of $operand1 + operand2$. If the sum is 0 then 0 shall be displayed; otherwise, there shall be no leading 0s. Except as noted in Part 4, you may assume that the sum is representable with four or fewer decimal digits. Unless you implement the bonus, we are not specifying the correct behavior for sums greater than $9,999$.
- When toggle switch 0 (the switch that is toggled by the 'f' key) is moved to the "on" position, the seven-segment displays will display the difference of $operand1 - operand2$. If the difference is 0 then 0 shall be displayed; otherwise, there shall be no leading 0s. Except as noted in Part 4, you may assume that the difference is non-negative; unless you implement the bonus, the correct behavior for negative differences is unspecified.
- You may assume that at most one toggle switch will ever be in the "on" position. We are not specifying the correct behavior if two toggle switches are in the "on" position.

4 Extra Credit

Up to 15 bonus points

The calculator specification left a few edge cases unspecified. For bonus credit, implement some or all of these specifications for the edge cases.

Note: You may receive bonus credit as part of last week's lab or as part of this week's lab, but not both. *If you implement the edge cases, indicate so in the header comments of the calculator implementation that you want evaluated for bonus credit.* ***We will not evaluate a calculator implementation for bonus credit that does not state that it has the edge cases implemented.***

_____ **Bonus +5** Handle an attempt to build a too-great operand

- When building an operand, if the user presses a fifth number button, the operand being built is invalid. The seven-segment displays shall display Err.
- The only way to clear this error is to move the operand's toggle switch to the "off" position. When the user does so, the original operand's value will be preserved. For example:
  - *operand*1 holds the value 53.
  - The user moves toggle switch 3 to the "on" position and presses '4', '5', '2', '3' which causes the display will show these values in sequence:

    0
    4
    45
    452
    4523
  - If the user were to move toggle switch 3 to the "off" position, then *operand*1 would take the value $4,523$. Instead, the user presses '7' which causes the display to show Err.
  - The user now moves toggle switch 3 to the "off" position, and *operand*1 retains the value 53.
- This edge case must work for both operands to receive any credit for this edge case.

_____ **Bonus +3** Handle a too-great sum

If, when toggle switch 1 is moved to the "on" position, the sum is greater than $9,999$ then the seven-segment displays shall display Err.

_____ **Bonus +5** Handle negative differences

If, when toggle switch 0 is moved to the "on" position, the difference is between -1 and -999 (inclusive) then the difference shall be displayed, including the negative sign. As with non-negative values, there shall be no leading 0s. For example, subtracting $53 - 102$ would cause the seven-segment displays to display -49.

_____ **Bonus +2** Handle a too-low difference

If, when toggle switch 0 is moved to the "off" position, the difference is less than -999 then the seven-segment displays shall display Err.

- If you do not implement the 5-point "Handle negative differences" bonus, then you can receive the 2-point "Handle a too-low difference" by displaying Err for any value less than 0.