Using C to Perform I/O on Simulated Hardware

Part 1 due the week of November 9 at 11:59pm on your lab day
Parts 2 & 3 due the week of November 16 at 11:59pm on your lab day

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). **This lab is designed for use on the csce.unl.edu Linux server.**

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.

The file archive *lab6.tar* includes six 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`.

demo.c A program that demonstrates using the SSBC library. Run the command `make demo` to build the program, and then run it with the command `./demo`. This file is described line-by-line in *ssbc.pdf*

warmup.c Starter code for part 1 of this lab. Run the command `make warmup` to build the program, and then run it with the command `./warmup`.

poll-calculator.c Starter code for part 2 of this lab. Run the command `make poll-calculator` to build the program, and then run it with the command `./poll-calculator`.

interrupt-calculator.c Starter code for part 3 of this 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.**

1 Familiarize Yourself with the Simulated Single Board Computer

20 points

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

- When the left-most toggle switch is in the "on" position, the left three seven-segment displays show 231; when the left-most toggle switch is in the "off" position, the left three seven-segment displays are blank.
- When the right-most toggle switch is in the "on" position, the right-most seven-segment display displays the number corresponding to the last number button that was pressed. When the right-most toggle switch is in the "off" position, the right-most seven-segment display is blank. If a number button has not yet been pressed since launching the program, the right-most seven-segment display will be blank, even if the toggle switch is in the "on" position.
- Toggling the middle two toggle switches has no effect.

Hint: You may want to create a ten-element array of 8-bit bit vectors to map a decimal digit to the bit vector needed to display that digit on a single seven-segment display.

Rubric:

_____ **+4** 231 displays on the three left displays when the left-most toggle switch is in the "on" position.

_____ **+2** The left three displays are blank when the left-most toggle switch is in the "off" position.

_____ **+2** The right-most display is blank if a number button has not yet been pressed, regardless of the position of the right-most toggle switch.

_____ **+4** If a number button has been pressed, then the right-most display shows the number of the most-recent number button pressed when the right-most toggle switch is in the "on" position.

_____ **+2** The right-most display is blank when the right-most toggle switch is in the "off" position.

_____ **+1** The behaviors for the left-most toggle switch and for the right-most toggle switch can happen at the same time (*e.g.*, 231 can be made to appear and disappear while also updating the number entered from the keypad).

_____ **+1** Nothing happens as a result of toggling either of the middle two toggle switches.

_____ **+2** The SSBC screen does not get corrupted under casual use of the program.

_____ **+2** The SSBC remains responsive to inputs throughout using the program.

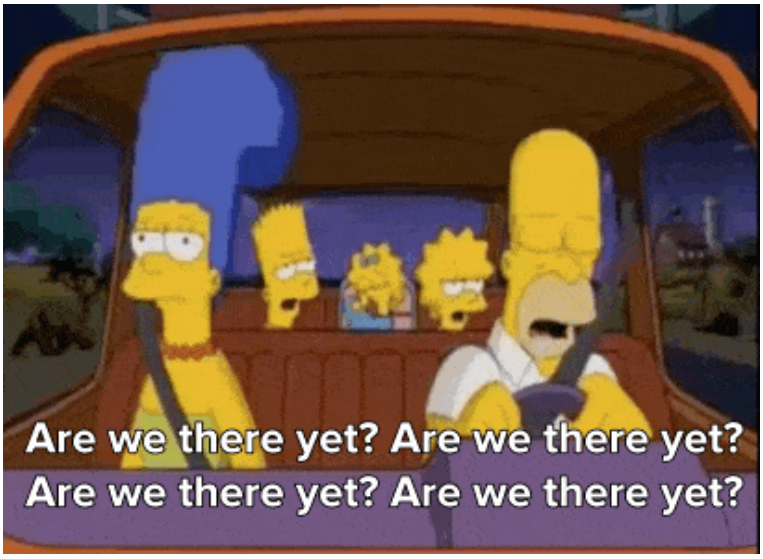

Figure 1: Polling. Image by 20th Century Fox Television

2 Use Polling to Detect Inputs

35 points

It may not have been obvious while you were writing your warmup program, but you can not tell when the user has pressed the same number button twice in a row by simply reading the BCD value from the number pad's register. This is obvious if you run the demonstration program and pay attention to the debug window. In Part 2, you will overcome this problem by *polling* the number pad register's *dirty bit*.

"Dirty bit" is a bit used to indicate that something has changed. Here, the dirty bit indicates that the user has pressed a number button. "Polling" is a technique to periodically read a value to determine if it has changed.

Edit *poll-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.
  - 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 Part 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.

• Implementation

- Poll the number pad register's dirty bit to determine when a number button has been pressed. The SSBC library will place a 1 in the dirty bit whenver a number button has been pressed.
- Afer reading the number pad register's BCD value, clear the dirty bit by setting it to 0.
- You may use any built-in C operations.
- *Hint:* You may want to write a function that will convert a integer into the bit vector for the seven-segment displays' register. The array to generate bit vectors for a single digit from the warmup exercise will also be useful.

Rubric:

_____ **+1** The displays are blank when all toggle switches are in the "off" position

_____ **+1** *Operand*1 with with one digit can be built.

_____ **+2** *Operand*1 with non-repeating digits can be built, *e.g.*, 231.

_____ **+3** *Operand*1 with repeating digits can be built, *e.g.*, 2331.

_____ **+1** When building *operand*1, the "ones" place is always displayed

_____ **+2** When building *operand*1, there are no leading 0s in the "tens," "hundreds," or "thousands" places.

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

_____ **+9** It is possible to build *operand*2 in accordance with specification.

_____ **+1** When toggle switch 2 is moved to the "off" position, *operand*2 takes on the value built.

_____ **+5** When toggle switch 1 is moved to the "on" position, the sum of *operand*1 + *operand*2 is displayed without leading 0s.

_____ **+5** When toggle switch 0 is moved to the "on" position, the difference of *operand*1 − *operand*2 is displayed without leading 0s.

_____ **+3** Number button presses are detected by polling the dirty bit.

_____ **+1** The dirty bit is reset after obtaining a the value of the number button pressed.

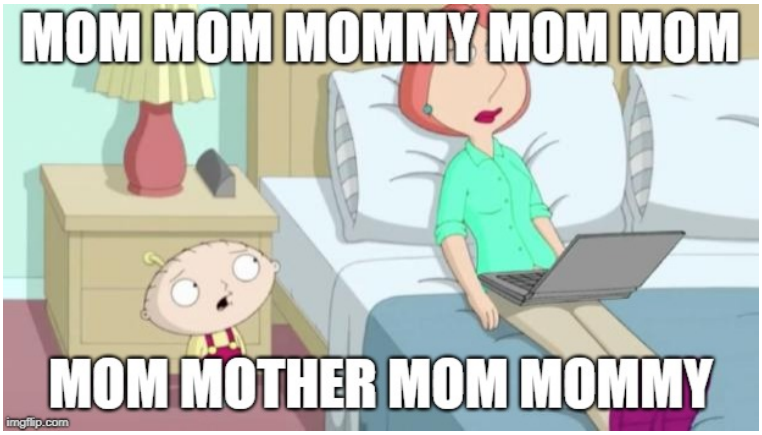

Figure 2: Interrupts. Image by 20th Century Fox Television

3 Use Interrupts to Detect Inputs

20 points

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 Part 3, 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 specification as the calulator from Part 2.
- 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 Part 2 can be reused, but the overall structure will differ.

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.

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: to receive bonus credit, you only need to implement the edge cases in one of the calculator implementations; you do not need to implement the edge cases in both calculator implementations. *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.***

- **Handle an attempt to build a too-great operand (+5)**
  - 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.

- **Handle a too-great sum (+3)** 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.
- **Handle negative differences (+5)** If, when toggle switch 0 is moved to the "off" 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.
- **Handle a too-low difference (+2)** 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.

5 Deliverables

1. *warmup.c*, due the week of November 9 at 11:59pm on the day of your scheduled lab section
2. *poll-calculator.c*, due the week of November 16 at 11:59pm on the day of your scheduled lab section
3. *interrupt-calculator.c*, due the week of November 16 at 11:59pm on the day of your scheduled lab section