Lab 9b

Polling to Detect Inputs on Simulated Hardware

Due: Week of April 19, 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 familiarize yourself with the Simulated Single Board Computer **You should be able to complete this lab assignment during lab time.**

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-polling.zip* or *ssbclab-polling.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`.

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

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

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

2 Use Polling to Detect Inputs

It may not have been obvious while you were writing your warmup program, but you cannot 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. You will now 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 3, 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 3, 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 3, 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.

Turn-in and Grading

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

This assignment is worth 35 points.

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.

3 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 this week's lab or as part of next 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 "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.

_____ **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.