Simulated Single Board Computer*

SSBC-1

The SSBC-1 is the latest product from Exciting Emulations[1] and is guaranteed to bring you minutes of fun. The finest piece of simulated hardware, the SSBC-1 is second only to actually placing your hands on a real, physical single board computer.

In using the SSBC-1, you will have the opportunity to use inputs such as simulated toggle switches and a simulated numeric keypad, and you will have the opportunity to create output via simulated seven-segment displays.

1 Overview

As shown in Figure 1, the SSBC-1 includes a numeric keypad and four toggle switches for input, and four seven-segment displays for output. Unlike a real single board computer, you will not have your SSBC-1 attached to a separate general-purpose computer for side-loading and debugging, and so the SSBC-1 also includes a small space for you to print information to help you debug your programs. ***Note:*** **your terminal window must be at least** 24×80 **for SSBC-1.** A larger terminal window will also work, but smaller terminal window will give you an unsatisfactory experience.

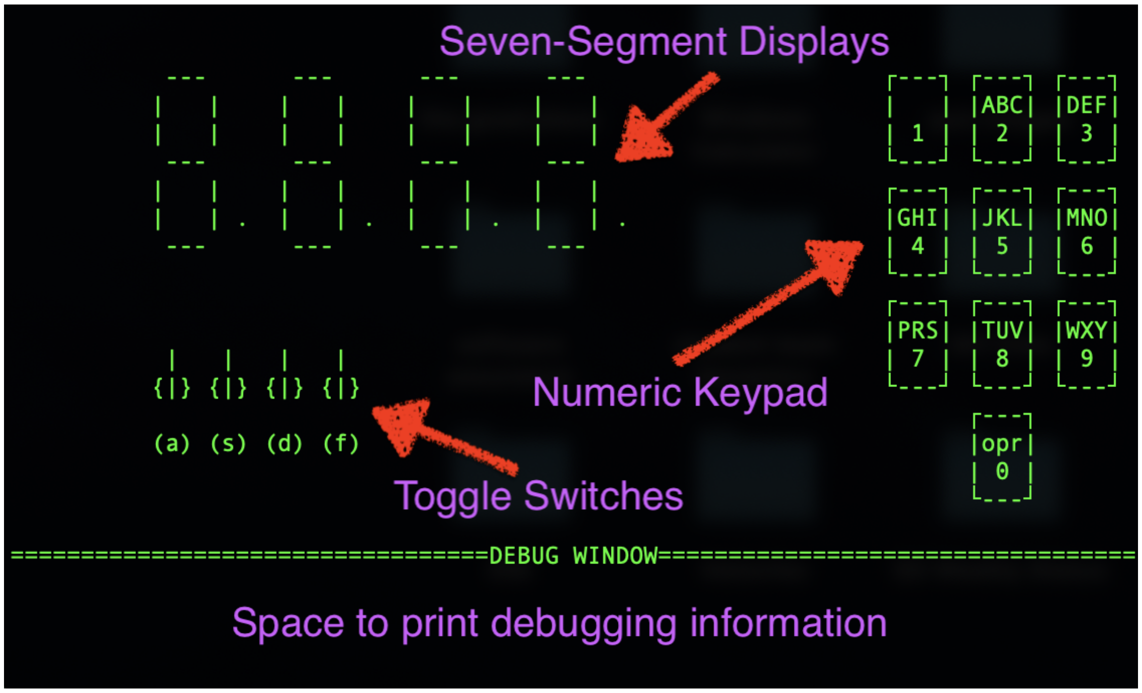
The toggle switches (Figure 2) can be placed in one of two positions, and a toggle switch will hold its position until it is placed in the other position. You can toggle a switch by pressing the letter on your keyboard that corresponds to the switch. For example, the leftmost switch can be placed in the "up" position by pressing `a`, and then it can be returned to the "down" position by pressing `a` again.

The numeric keypad (Figure 3) consists of ten momentary buttons, labeled 0-9. You can depress a number button by pressing the number on your keyboard that corresponds to the number button. For example, by pressing `3`, the upper-left number button will briefly illuminate, and the value 3 will be placed in the input register for the numeric keypad. As described in Section 3.2.2, while depressing the button is momentary, the value will persist in the input register even after the number button is released.

The seven-segment displays (Figure 4) consists of seven segments that can be illuminated or not to create characters, such as the number 3, as well as a "dot" that can be used

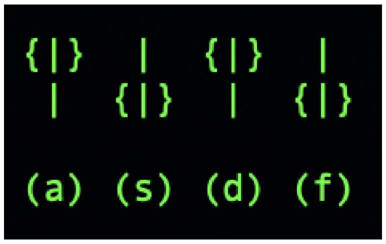
*The SSBC library, starter code, and documentation ©2020, Christopher A. Bohn

[1]Exciting Emulations is a wholly-figmentary subsidiary of Babbage's Analytic Engines, a Limited-Reality Corporation.



```
 ---     ---     ---     ---                 ┌---┐  ┌---┐  ┌---┐
|   |   |   |   |   |   |   |                |   |  |ABC|  |DEF|
|   |   |   |   |   |   |   |                | 1 |  | 2 |  | 3 |
 ---     ---     ---     ---                 └---┘  └---┘  └---┘
|   |   |   |   |   |   |   |                ┌---┐  ┌---┐  ┌---┐
|   | . |   | . |   | . |   | .              |GHI|  |JKL|  |MNO|
 ---     ---     ---     ---                 | 4 |  | 5 |  | 6 |
                                             └---┘  └---┘  └---┘
                                             ┌---┐  ┌---┐  ┌---┐
 |    |    |    |                            |PRS|  |TUV|  |WXY|
{|}  {|}  {|}  {|}                           | 7 |  | 8 |  | 9 |
                                             └---┘  └---┘  └---┘
(a)  (s)  (d)  (f)                                  ┌---┐
                                                    |opr|
                                                    | 0 |
                                                    └---┘
==================================DEBUG WINDOW==================================
```

Figure 1: Overview of SSBC-1.



```
{|}   |   {|}   |
 |   {|}   |   {|}

(a)  (s)  (d)  (f)
```

Figure 2: Toggle switches.

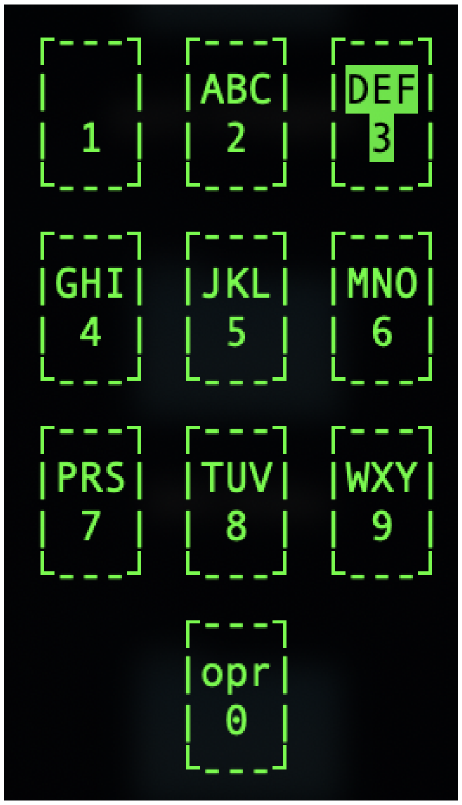


Figure 3: Numeric keypad.

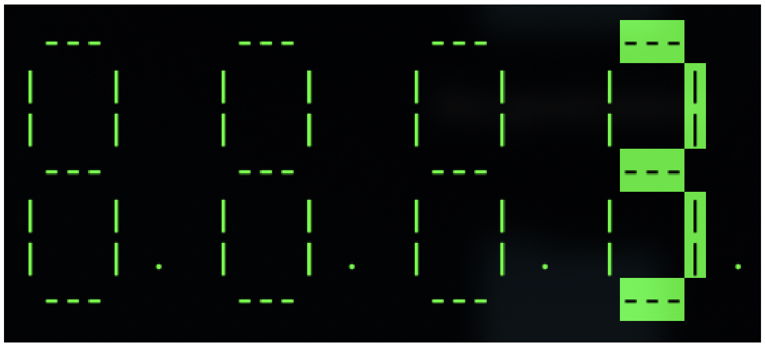
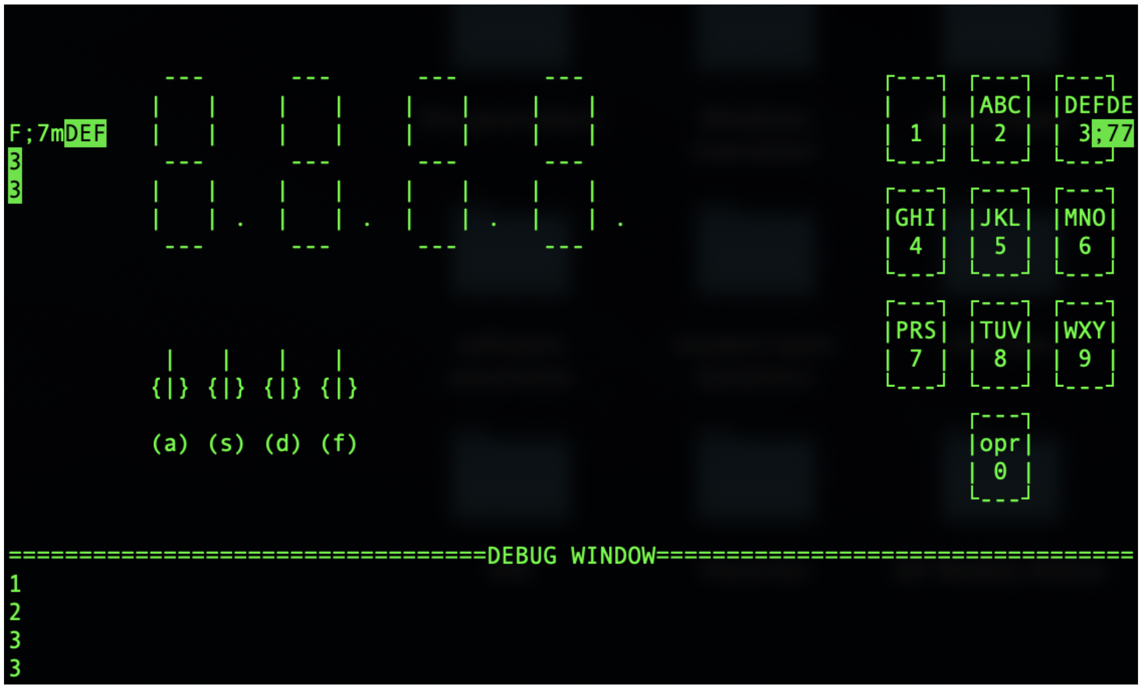


Figure 4: Seven-segment displays.



```
             ---      ---      ---      ---                       ┌---┐  ┌---┐  ┌---┐
            |   |    |   |    |   |    |   |                      |   |  |ABC|  |DEFDE
F;7mDEF     |   |    |   |    |   |    |   |                      | 1 |  | 2 |  | 3;77
3            ---      ---      ---      ---                       └---┘  └---┘  └---┘
3           |   |    |   |    |   |    |   |                      ┌---┐  ┌---┐  ┌---┐
            |   | .  |   | .  |   | .  |   | .                    |GHI|  |JKL|  |MNO|
             ---      ---      ---      ---                       | 4 |  | 5 |  | 6 |
                                                                  └---┘  └---┘  └---┘
                                                                  ┌---┐  ┌---┐  ┌---┐
                                                                  |PRS|  |TUV|  |WXY|
              |    |    |    |                                    | 7 |  | 8 |  | 9 |
             {|}  {|}  {|}  {|}                                   └---┘  └---┘  └---┘
                                                                         ┌---┐
             (a)  (s)  (d)  (f)                                          |opr|
                                                                         | 0 |
                                                                         └---┘
=================================DEBUG WINDOW==================================
1
2
3
3
```

Figure 5: Garbled SSBC-1 screen.

for decimal points. The seven- segment displays will always illuminate their segments in accordance with the contents of the display register.

1.1 Other Controls

In addition to the controls described above, the SSBC-1 will also respond to these actions on your keyboard:

- SSBC-1 is an imperfect simulation of a single board computer. If your screen becomes garbled (Figure 5) then you can press the `space` bar to refresh the screen. In extreme cases, you may need to press the `space` bar multiple times. (Under normal usage, your screen should not become garbled.)
- If you wish to terminate the SSBC-1, then press `Control-C`.
- Another way to terminate the SSBC-1 is to type `quit` with no more than 500ms between key presses.
- Any other keyboard presses will result in an audio and visual alert from SSBC-1 but will otherwise be ignored.

2 Demonstration Program

The `ssbc.h` header file defines seven functions:

void ssbc_launch() This function should be the first line of your program's `main` function. It will initialize SSBC-1 and generate the SSBC-1 screen.

void ssbc_terminate() In addition to the means to terminate SSBC-1from the keyboard, described earlier, you can also let SSBC-1 terminate by having your program end its perpetual loop. If you do this, your program should call `ssbc_terminate` to restore your terminal window's settings.

pthread_mutex_t *ssbc_get_mutex() This function returns a mutual exclusion token to manage race conditions in the SSBC-1's input/ output registers.

int ssbc_print(const char *fmt, ...) This function is used to print text in the debug window. The specification for the arguments is identical to that of `printf`. *Do not use `printf` for printing while using the SSBC-1!* The `ssbc_print` function returns the number of characters printed (which may exceed the available space in the debug window.)

void *ssbc_get_keypad_address() This function returns the address of the numeric keypad's register.

void *ssbc_get_toggle_address() This function returns the address of the toggle switches' register.

void *ssbc_get_7_segment_address() This function returns the address of the seven-segment displays' register.

The `demo.c` program demonstrates the use of these functions. When a toggle switch is moved to the "on" position, one of the seven-segment displays illuminates the letter used to toggle the switch. When a number key is pressed, the number is printed in the debug menu.

```
/*
 * Simulated Single Board Computer, (c) 2020 Christopher A. Bohn
 */

#include "ssbc.h"

pthread_mutex_t *mutex;
uint32_t *display_controls;
uint8_t *toggle_control;
uint16_t *keypad_control;
int running = 1;

```

```c
13 int main() {
14     ssbc_launch();
15     uint8_t letters[] = {0x78, 0x4F, 0x76, 0x6F}; // {f,d,s,a}
16     mutex = ssbc_get_mutex();
17     display_controls = ssbc_get_7_segment_address();
18     toggle_control = ssbc_get_toggle_address();
19     keypad_control = ssbc_get_keypad_address();
20     keypad_control += 1;
21     uint8_t lastNumber = 0x0F;
22     while (running) {
23         pthread_mutex_lock(mutex);
24         uint8_t toggle_positions = *toggle_control;
25         uint8_t number = *keypad_control & 0x0Fu;
26         pthread_mutex_unlock(mutex);
27         for (unsigned int i = 0; i < 4; i++) {
28             if (toggle_positions & 0x1u << i) {
29                 pthread_mutex_lock(mutex);
30                 uint32_t clear_position = ~(0xFFu << (8 * i));
31                 uint32_t character_in_position = (uint32_t)(letters[i]) << (8
32                 *display_controls = (*display_controls & clear_position) | ch
33                 pthread_mutex_unlock(mutex);
34             } else {
35                 pthread_mutex_lock(mutex);
36                 uint32_t clear_position = ~(0xFFu << (8 * i));
37                 *display_controls = *display_controls & clear_position;
38                 pthread_mutex_unlock(mutex);
39             }
40         }
41         if (number != lastNumber) {
42             pthread_mutex_lock(mutex);
43             ssbc_print("%d\n", number);
44             pthread_mutex_unlock(mutex);
45         }
46         lastNumber = number;
47         if (number == 0) {
48             running = 0;
49         }
50     }
51     ssbc_terminate();
52     return 0;
53 }
```

Line 5 imports ‘ssbc.h‘; you shouldn’t need to import any other headers.

Lines 7-10 declare global variables to hold values obtained from some of the function calls described earlier. In this example, we could have had these variables local to `main`, but some programs will be easier to write if these variables are global. Notice that we can use pointers to any size of integer to hold the addresses of SSBC-1's input/output registers.

The `running` variable on line 11 is used as a loop variable on line 22.

Line 14 launches the SSBC-1. Line 15 contains the bit vectors that will be used to display letters on the seven-segment displays. See Section 3.1.1 for a discussion of the bit vectors.

Lines 16-19 populate the variables in lines 7-10. Because this program does not use the first 16 bits of the numeric keypad's register, line 20 uses pointer arithmetic to point to the second 16 bits.

Line 21 is a variable to keep track of the last number pressed. Recall that after a number is pressed, its corresponding value persists in the register. This means that without further information our program cannot determine whether the value in the register is a stale value or the result of a new keypress. In Sections 3.2.2 and 4.1 we will discuss how to detect that a key has been pressed. For now, `demo.c` will ignore a value if it is the same value that was previously read from the register.

Lines 22-50 are the main loop. A common idiom is to write an infinite loop using `while(1)` or `for(;;)`. Here the loop is conditioned on the `running` variable from line 11. This will allow us to write code that will end the loop due to user input.

Lines 23 & 26 (along with other lines later in the loop) lock and unlock the mutual exclusion token. As discussed in Section 3, the SSBC-1 is an imperfect simulation that will require you to use the mutual exclusion token to guarantee that reading and writing registers is atomic.[2]

Lines 24 & 25 read from the input registers, and the loop on lines 27-40 updates the bit vector in the seven-segment displays' register based on the bit vector read from the toggle switches' register.

If a new number button has been pressed, then line 43 prints the number to the debug window. Note that this print statement also requires locking and unlocking the mutual exclusion token.

If that number happened to be 0 then line 48 sets the `running` variable to 0, which will terminate the loop. Once the loop has terminated, the call to `ssbc_terminate` on line 51 restores the terminal window's original settings.

[2]Students: why do you think this is necessary for a simulated SBC and not for a real, physical SBC?

3 Input/Output Register Descriptions

This section describes the registers of the memory-mapped simulated inputs and outputs.

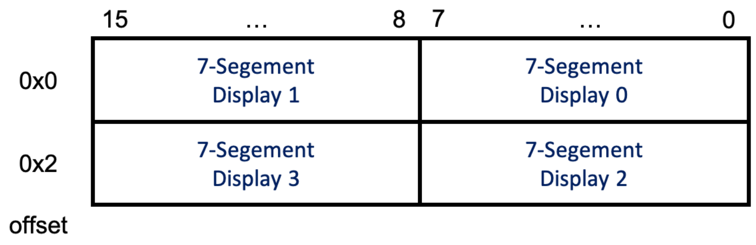
SSBC-1 is an imperfect simulation. Unlike physical hardware, the simulated input hardware is not guaranteed to atomically update the input registers, and the simulated output hardware is not guaranteed to reflect instantaneous changes to the output registers. Interleavings with the program may result in inconsistent updates to and reads from the input/output registers. Moreover, interleavings with the program may result in the terminal window being improperly refreshed, such as is shown in Figure 5. Obtaining a mutual exclusion lock before accessing a simulated I/O register and releasing the lock when finished with reading from or writing to the register will result in atomic reads & writes. Because the `ssbc_print` function also updates the display, be sure to obtain the lock before printing and release the lock when finished. ***Note:* failure to release the lock will render your program and the SSBC-1 non-responsive.**

In the diagrams that follow, the values on the left indicate each row's address offset (in bytes) from the register's base address. The values along the top indicate the bit positions for the fields within the bit vectors.

3.1 Output Register

3.1.1 Seven Segment Displays Register

The seven-segment displays (Figure 4) are numbrered, from right-to-left 0..3. Each has a field within the seven-segment displays' register that is used to illuminate particular segments.



Within each field, the bits that corrsespond to each segment and the dot are shown in Figure 6.

3.2 Input Registers

3.2.1 Toggle Switches Register

The toggle switches (Figure 2) are numbered, from right-to-left, 0..3. Each has a corresponding bit in the toggle switches' register that indicates its on/off orientation.

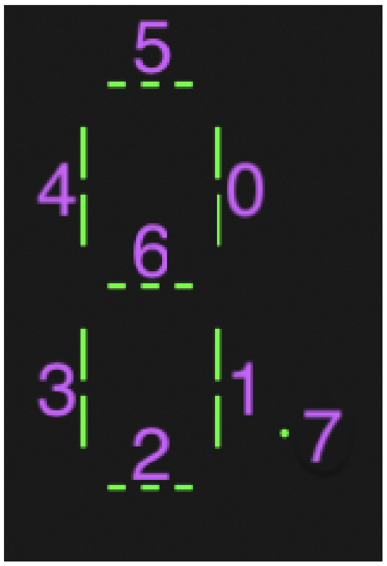
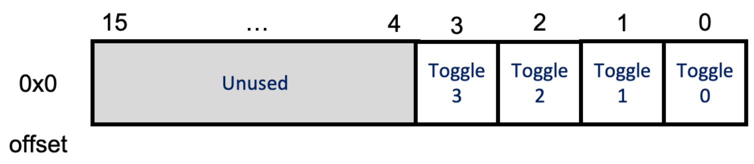


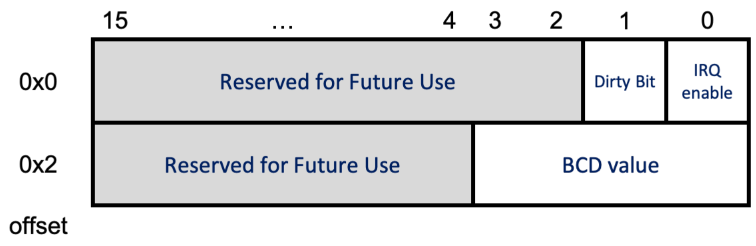
Figure 6: Bits to illuminate each segment for a seven-segment display.



3.2.2 Numeric Keypad Register

When a number button on the numeric keypad is pressed, the binary representation of that number is placed in the `BCD value` field of the numeric keypad register. After the SSBC-1 has been initialized and before any number buttons have been pressed, the `BCD value` field will hold a bit vector that does not correspond to a decimal value. After a number button has been pressed, the `BCD value` field will hold the bit vector corresponding to the last button pressed.

When a number button is pressed, the **`dirty bit`** is set to 1. If your program sets the **`dirty bit`** to 0 when the `BCD value` is read, then the program can poll the **`dirty bit`** to determine whether there has been a fresh key press.



4 Simulated Interrupts

The SSBC-1 uses OS signals to simulate hardware interrupts. If you define a signal handler that returns void and optionally takes an `int` argument, and register that signal handler with `sigset(`*`signal, signal_handler`*`)` then the signal handler will be called whenever that signal has been issued.

Register signals only after calling `ssbc_launch`, as the SSBC-1's initialization code sets the simulated interrupts to be ignored by default.

4.1 Numeric Keypad Interrupt

If the numeric keypad register's `IRQ enable` bit is set, then the SSBC-1 will issue the `SIGUSR1` signal whenever a number key has been pressed. A signal handler registered for `SIGUSR1` will be called whenever a number key is pressed, if the `IRQ enable` bit is set.