Aaron Hong (ahong02)

Stephen Macris (smacris)

EE469, 5/19/23

**Lab 5 Report**

Procedure:

In this lab, the CPUlator NIOS II interface (<https://cpulator.01xz.net/?sys=nios-de1soc>) was used to write programs that implement functionality for its LEDs and push buttons. Programs written in both C and NIOS II were compared and analyzed.

Results**:**

**Task 1:**

*Part 1:*

1. What does the (volatile long \*) keyword mean?

The (volatile long \*) keyword initializes a pointer to a volatile long (32-bit) data type. The volatile keyword specifies that this value is subject to change.

2. Verify that this program works as intended in the CPUlator. Make sure that the Language set on the CPUlator is in C. Take a screenshot of the LEDs, showing that the right-most LED is illuminated. Make sure to include this screenshot in your report.

As shown in the below screenshot, the given code works properly since the rightmost LED is illuminated.

A picture containing text, software, computer icon, web page

Description automatically generated

3. Modify this example code to illuminate all 10 of the LEDs on the NIOS board. What value did you choose to write to \*LED? Why? Take a screenshot of the LEDs, showing that all of the LEDs are illuminated. Make sure to include this screenshot in your report.

The value 0b1111111111 was written to \*LED. This is a binary value that assigns the last 10 bits at the address of \*LED. Since one LED is turned on by assigning one bit, 10 LEDs can be turned on by assigning 10 bits.

A picture containing text, software, web page, computer icon

Description automatically generated

*Part 2:*

1. Compile this program. In the Disassembly tab, continuously Step Over the generated Assembly code

until you are looping through the main branch.

2. What is the PC address of the main branch? Why is it not at address 0?

The main branch is at PC address 0x230. It isn’t at address 0 because there needs to be space for the startup instructions that initialize the runtime.

3. Why is this while(1){} loop necessary?

The while(1){} loop is necessary because it ensures that the program will keep running and continuously update Swval with the state of the switches.

4. Which temporary register (R0, R1, R2, etc.) holds the value of Swval?

R2 holds the value of Swval.

5. Notice that when we do a Step Over in the main branch, the temporary register R2 toggles between three different hexadecimal values. What’s the significance of these three difference values that you see?

The first value that is loaded is the first half of the address of the switch values in memory. The second value is the full address. The third value is the actual switch values.

**Task 2:**

*Part 1:*

1. What is the register address for the LEDs on the NIOS processor? Taking a look at the example codes in Task 1, how would we define this register address in C?

The register address for the LEDs on the NIOS processor is 0xFF200000. You define the register address in C with the following statement: #define LED ((volatile long \*) 0xFF200000).

2. What is the register address for the Pushbuttons on the NIOS processor? Taking a look at the example codes in Task 1, how would we define this register address in C?

The register address for the Pushbuttons on the NIOS processor is 0xFF200050. You define the register address in C with the following statement: #define PUSHBUTTONS ((volatile long \*) 0xFF200050).

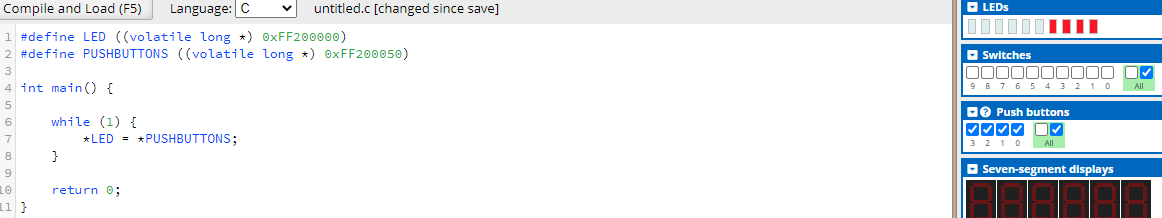
3. How do we obtain (in C) the information on which button was pressed?

Calling \*PUSHBUTTONS accesses the data that the PUSHBUTTONS pointer points to. It was initialized to point to address 0xFF200050, which is where the NIOS processor stores push button status.

4. When you compile your program, what is the size of the ELF executable (you can find this information in the Messages section of the CPUlator)?

The size of the ELF executable is 2656 bytes.

5. Take a screenshot of the LED and Pushbutton peripherals on CPUlator and include them in your report. Also save your .c source code (Ctrl + S, or go to File -> Save) as Lab5\_task2\_part1.c



*Part 2:*

1. Fill in the register address for the PUSHBUTTON and LED. Compile this program on the CPUlator. For this, make sure to set the Language to NIOS II. Verify that the functionality for this assembly code is the same as the C code you created in Task 2 part 1.

2. When you compile this program, what is the size of the ELF executable (you can find this information in the Messages section of the CPUlator)?

The total size of the ELF executable is 28 bytes.

3. Is the size of this ELF executable similar or different from the size produced from the C code? Please explain thoroughly why these are similar or different.

This ELF executable is much smaller than the one produced from the C code. This is because writing the same program in NIOS II compiles to much fewer assembly instructions. Namely, the startup code from the C implementation is not there anymore: the NIOS II program assembly starts at address 0.

4. Save your assembly code (Ctrl + S, or go to File -> Save) as a .s file called Lab5\_task2\_part2.s.

Appendix

*Task 2 Part 1 C code:*

#define LED ((volatile long \*) 0xFF200000)

#define PUSHBUTTONS ((volatile long \*) 0xFF200050)

int main() {

    while (1) {

    \*LED = \*PUSHBUTTONS;

    }

    return 0;

}

*Task 2 Part 2 NIOS II Assembly code:*

.equ PUSHBUTTON, 0xFF200050

.equ LED, 0xFF200000

start:

movia r2,PUSHBUTTON

ldwio r3,(r2) # Read in buttons - active high

movia r2,LED

stwio r3,0(r2) # Write to LEDs

br start