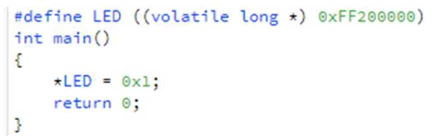
Nattapon Oonlmaom and Kiana Peterson
EE 469
May 26, 2023
Lab 5: Input/Output Interfacing with NIOS II

Procedure

The purpose of this lab is to learn input/output interfacing with NIOS II using CPUlator. The objectives are to be familiar with NIOS II processor, to learn basics of general-purpose input and output (GPIO), to perform simple digital I/O interfacing push button and interfacing LED, to understanding the pointers in C and how to use them to address GPIO registers, and to understanding optimization differences between C and Assembly code.

Task #1:

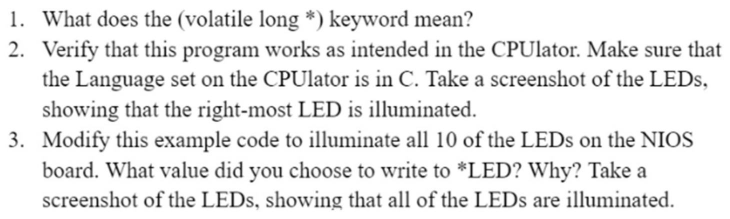
Part 1



```
#define LED ((volatile long *) 0xFF200000)
int main()
{
    *LED = 0x1;
    return 0;
}
```

Figure 1.1 C Program for *Task1 P1*

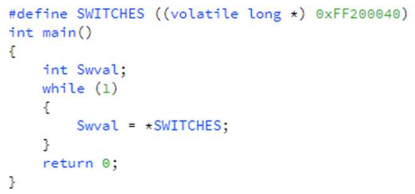
For Task 1 Part 1, a sample C program was given to interface the NIOS II processor with the LEDs. The code given, shown above, illuminates the right-most LED. After putting this code into the CPUlator, we answered the following given questions:



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

To answer these questions, the instructions were followed carefully, the code was modified, and some screenshots were taken as instructed. Then, the search engine was utilized and cited to find the meaning of notations, such as “volatile long *” for answering questions. Then, the code was analyzed and modified to illuminate all ten LEDs. The modified code can be found under Appendix, and the answers to these questions can be found underneath the Results section.

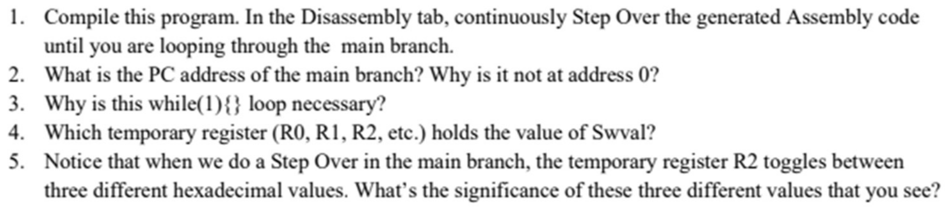
Part 2



```
#define SWITCHES ((volatile long *) 0xFF200040)
int main()
{
    int Swval;
    while (1)
    {
        Swval = *SWITCHES;
    }
    return 0;
}
```

Figure 1.2 C Program for *Task 1 P2*

For Task 1 Part 2, a sample C program was given to interface the NIOS II processor with the switches. After putting this given code into the CPUlator, the following given questions were answered:



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?
3. Why is this while(1){} loop necessary?
4. Which temporary register (R0, R1, R2, etc.) 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 different values that you see?

To answer these questions, the code was analyzed using the given sets of instructions and the search engine was used to find the meaning of notation, such as "while(1){} loop" and "temporary register" to answer the required questions. Then, the register memory of the CPUlator was analyzed to answer these questions underneath the Results section.

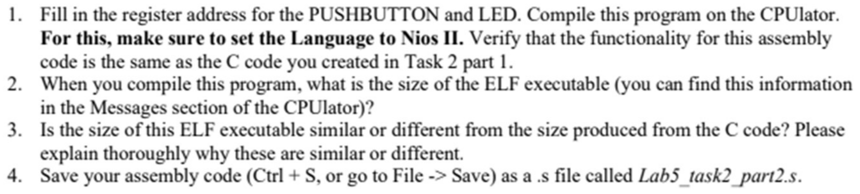
Task #2:

Part 1

Task 2 Part 1 was to write a C code to read the values of the NIOS pushbuttons and display that value onto the LED peripheral. If button 0 is pushed, the right-most LED illuminates. If button 1 is pushed, the second rightmost LED illuminates. If both buttons are pushed, both right-most and second rightmost LED illuminates. To write the code, the guiding questions were answered and can be found under Results. To answer the guiding questions, a search engine was used to find accurate answers and the source is cited under the bibliography. To modified the code correctly, some random values were assigned to *LED to observe patterns. It is also recommended that future students try this method. The code written is attached under Appendix.

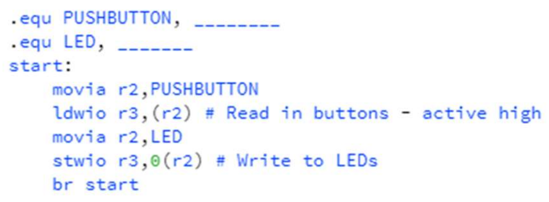
Part 2

Task 2 Part 2 was to fill into the given code (shown below) to fill in the register address for the PUSHBUTTON and LED. Then, follow the given instructions and answer the following questions:



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)?
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.
4. Save your assembly code (Ctrl + S, or go to File -> Save) as a .s file called *Lab5_task2_part2.s*.

For this task, the approach was similar to the rest of the task. The code was analyzed and modified using the given sets of instructions and the search engine was used to find the meaning of notation, such as “ELF executable” to answer the required questions. Then, the code is saved as *Lab5_task2_part2.s*. In addition to the answers of the asked questions under Results, a snapshot of the code can be found under Appendix.



```
.equ PUSHBUTTON, ________
.equ LED, _______
start:
    movia r2,PUSHBUTTON
    ldwio r3,(r2) # Read in buttons - active high
    movia r2,LED
    stwio r3,0(r2) # Write to LEDs
    br start
```

Figure 1.3 C Program for *Task 2 P2*

To summarize the challenges encountered in this lab, it was found that the most important lesson is to critically think and observe patterns. For the future students, it is advised to observe for patterns and the differences of each task. Even though the lab is not too difficult, it could get challenging if some details are overlooked. Additionally, it is found that utilizing search engines to clarify some notations and terms is critical to understanding.

Results

Task #1:

Part 1

1. The (volatile long *) keyword is casting the memory address to a pointer of the C program. The keyword *volatile* tells the compiler that the declared variable value may change unexpectedly. The *long* * indicates the casted pointer type– usually representing a 32-bit signed integer [1]. In this case, it informs the compiler that the memory location "LED," an I/O register, can be modified by external factors beyond the compiler's controls.

2. The given C code was verified in the CPUlator. Below is a screenshot of the right-most LED illuminated.

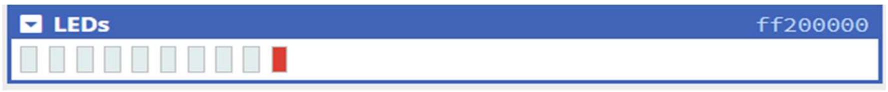


Figure 2.1 Right-Most LED Illuminated

3. The code was modified to illuminate all 10 of the LEDs. The value chosen to write to *LED was *0x3ff.* This is because the NIOS II processor interface uses a binary representation, and *0x3ff* represents a binary number with all 10 bits set to 1 [2].

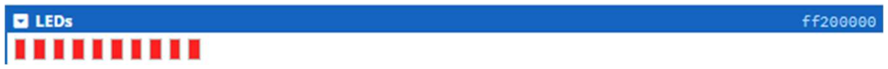


Figure 2.2 All LED Illuminated

Part 2

1. This program was compiled and continuously Step Over the generated Assembly code until looping through the main branch.
2. The PC (program counter) address of the main branch is *00000230*. This is because the initial addresses are occupied by the system initializations (stepped over before reaching the main branch) and therefore the actual execution at the start of the main branch is not at PC 0.
3. This *while(1){}* loop is necessary because it is an infinite loop that will continuously read the values of switches casted and update Swval accordingly until terminated [1]. The program would have been able to read the switches only once without the loop, which is not suitable for interfacing with switches' purpose.
4. The temporary register that holds the value of Swval is R2.
5. The three different hexadecimal values toggling in the temporary register R2 (ff200000, ff200040, and 00000000) should be corresponding to the values being read from the switches. Each value could indicate different conditions (i.e. one could suggest no switches pushed, one could suggest one or more switches pushed, or one could indicate

all switches being pushed). These values could be used to trigger different responses accordingly.

Task #2:

Part 1

1. The register address for the LEDs on the NIOS processor is *0xFF200000*. According to the example code is Task 1, this register address can be defined as *#define LED ((volatile long *) 0xFF200000)* in C.
2. The register address for the Pushbuttons on the NIOS processor is *0xFF200050*. According to the example code is Task 1, this register address can be defined as *#define PUSHBUTTONS ((volatile long *) 0xFF200050)* in C.
3. We can obtain the information on which button was pressed in C by initializing the button values and assigning it to the pushbutton under a while loop. This has to be instantiated under the loop, so the button is continuously read. The code for obtaining the information on which button was pressed in C can be found under Appendix.
4. When compiled, the size of the ELF executable is 2656 bytes.

Part 2

1. The register address for the PUSHBUTTON (0xFF200050) and for the LED (0xFF200000) was filled. Then, the given program was compiled and verified to be the same as the C code in Task 2 part 1 using CPUlator.
2. The size of the ELF executable is 28 bytes.
3. The size of this ELF executable is different from the size produced from the C code. The size of this ELF executable is smaller than that of the C code used in Task 2 part 1. This is because assembly code is low-level and is directly translated into machine code while C is higher-level. C has to go through additional layers of processing and framework before executed, which introduce additional data, resulting in a larger executable size. Assembly language on the other hand is minimal– it directly reads the buttons and writes to the LEDs [3].
4. The assembly code is saved as *Lab5_task2_part2.s.*

Final Product

In the end, the goals/objectives to be familiar with NIOS II processor, to learn basics of general-purpose input and output (GPIO), to perform simple digital I/O interfacing push button and interfacing LED, to understanding the pointers in C and how to use them to address GPIO registers, and to understanding optimization differences between C and Assembly code was accomplished. For Task 1 Part 1, a C program was used to interface the NIOS II processor with the LEDs and as a result, the given code was modified to illuminate all 10 LEDs. For Task 1 Part 2, a C was used to interface the NIOS II processor with the switches, and the sample code was analyzed for an understand for the pointers in C and how to use them. For Task 2 Part 1, a C code to read the values of the NIOS pushbuttons and display them onto the LED was written and the the ELF executable was noted. Lastly, for Task 2 Part 2, the code for Task 2 Part 1 was given in NIOS II Assembly. The code was modified to be filled in with the register address for the pushbutton and the LED, then compiled and verified that it matches with the behavior of Task 2 Part 1. The ELF executable of this code was then analyzed against Task 2 Part 1 to illustrate the differences between C and Assembly code. As a result, all objectives were addressed throughout the lab.

References

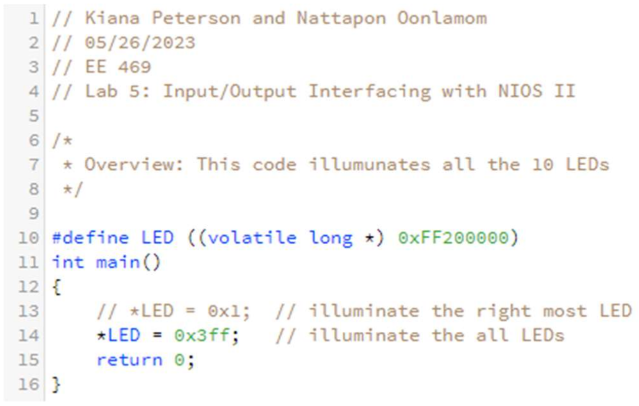
[1] “Introduction to the Volatile Keyword in C/C++.” Embedded.com. https://www.embedded.com/introduction-to-the-volatile-keyword/#:~:text=It%20tells%20the%20compiler%20that,of%20this%20are%20quite%20serious. [accessed May 27, 2023].

[2] “Nios II Processor Reference Guide.” intel.com. https://www.intel.com/programmable/technical-pdfs/683836.pdf. [accessed May 27, 2023].

[3] S. Smith. “The Scientist and Engineer's Guide to Digital Signal Processing.” dspguide.com. https://www.dspguide.com/ch28/5.htm. [accessed May 27, 2023].

Appendix

Task 1:

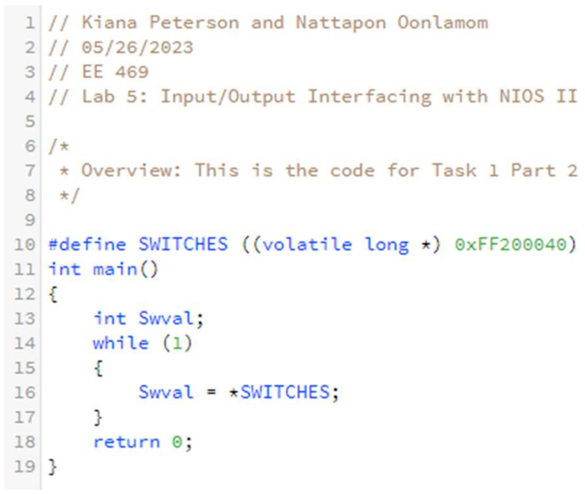


```c
// Kiana Peterson and Nattapon Oonlamom
// 05/26/2023
// EE 469
// Lab 5: Input/Output Interfacing with NIOS II

/*
 * Overview: This code illumunates all the 10 LEDs
 */

#define LED ((volatile long *) 0xFF200000)
int main()
{
    // *LED = 0x1;  // illuminate the right most LED
    *LED = 0x3ff;   // illuminate the all LEDs
    return 0;
}
```

Appendix 1 Modified Task1 Part 1



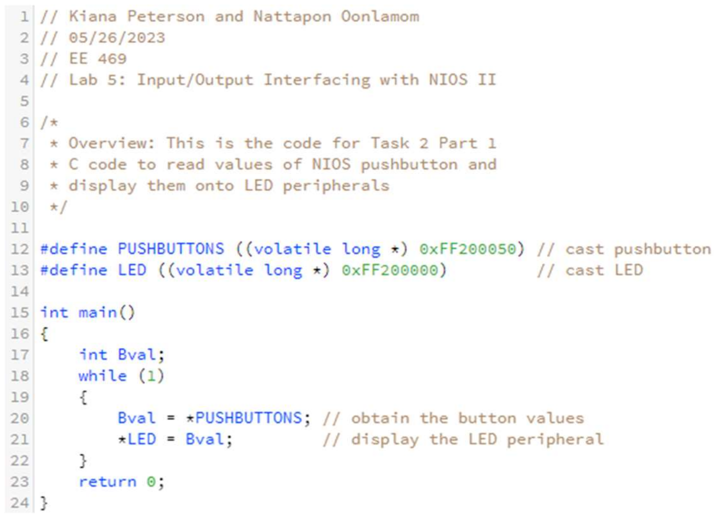
```c
// Kiana Peterson and Nattapon Oonlamom
// 05/26/2023
// EE 469
// Lab 5: Input/Output Interfacing with NIOS II

/*
 * Overview: This is the code for Task 1 Part 2
 */

#define SWITCHES ((volatile long *) 0xFF200040)
int main()
{
    int Swval;
    while (1)
    {
        Swval = *SWITCHES;
    }
    return 0;
}
```

Appendix 2 Task1 Part 2

Task 2:



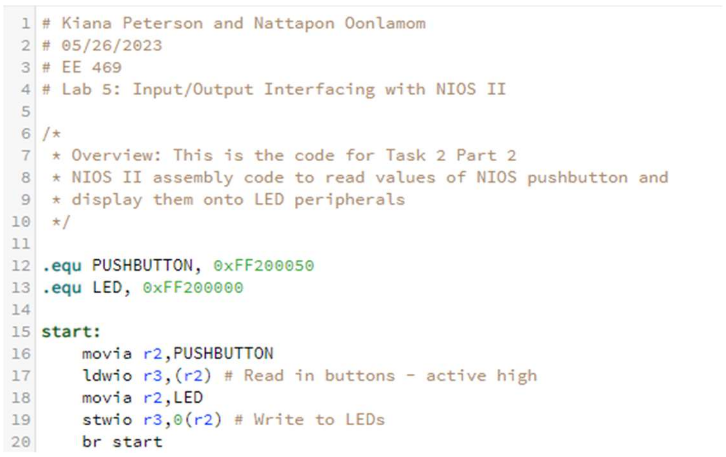
```
// Kiana Peterson and Nattapon Oonlamom
// 05/26/2023
// EE 469
// Lab 5: Input/Output Interfacing with NIOS II

/*
 * Overview: This is the code for Task 2 Part 1
 * C code to read values of NIOS pushbutton and
 * display them onto LED peripherals
 */

#define PUSHBUTTONS ((volatile long *) 0xFF200050) // cast pushbutton
#define LED ((volatile long *) 0xFF200000)         // cast LED

int main()
{
    int Bval;
    while (1)
    {
        Bval = *PUSHBUTTONS; // obtain the button values
        *LED = Bval;         // display the LED peripheral
    }
    return 0;
}
```

Appendix 3 Task2 Part 1



```
# Kiana Peterson and Nattapon Oonlamom
# 05/26/2023
# EE 469
# Lab 5: Input/Output Interfacing with NIOS II

/*
 * Overview: This is the code for Task 2 Part 2
 * NIOS II assembly code to read values of NIOS pushbutton and
 * display them onto LED peripherals
 */

.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
```

Appendix 4 Modified Task2 Part 2