Laboratory Exercise 1

A Simple Computer System

The purpose of this exercise is to learn how to create and use a simple computer system. The system will consist of an Altera Nios II processor and an application program. We will use the Quartus II and SOPC Builder software to generate the hardware portion of the system. We will use the *Altera Debug Client* software to compile, load and run the application program.

Part I

In this exercise, you will use the SOPC Builder to create the system in Figure 1, which consists of a Nios II/e processor and a memory block. The Nios II/e processor processes data. The memory block stores instructions and data.

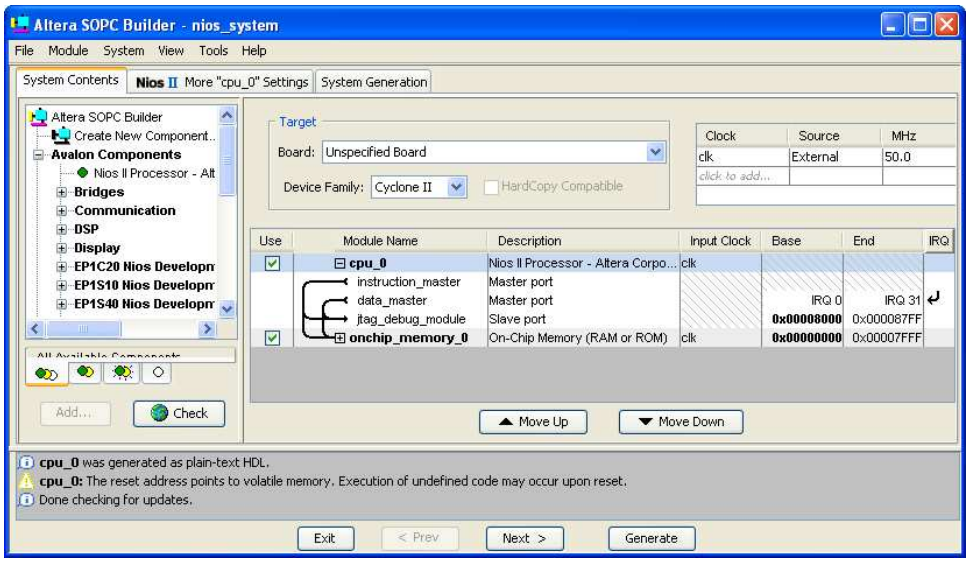

Figure 1. The Nios II system in SOPC Builder.

Implement the system in Figure 1 as follows:

1. Create a new Quartus II project. Select Cyclone II EP2C35F672C6 as the target chip, which is the FPGA chip on the Altera DE2 board.
2. Use the SOPC Builder to create a system named *nios_system*, which includes the following components:
   - Nios II/e processor with JTAG Debug Module Level 1
   - On-chip memory - RAM mode and 32 Kbytes in size with width of 32 bits
3. From the System menu, select Auto-Assign Base Addresses. You should now have the system shown in Figure 1.
4. Generate the system, exit the SOPC Builder and return to the Quartus II software.

5. Instantiate the generated Nios II system within a Verilog/VHDL module.
6. Assign the pin connections:
   - clk - PIN_N2 (This is a 50 MHz clock)
   - reset_n - PIN_G26 (This is the pushbutton switch KEY_0)
7. Compile the Quartus II project.
8. Program and configure the Cyclone II FPGA on the DE2 Board to implement the generated system. To use the system, we have to give the processor a program to execute, which will we do in Part II of this exercise.

Part II

In a digital computer all data is represented as strings of 1s and 0s. The Nios II assembly-language program in Figure 2 examines a word of data and determines the maximum number of consecutive 1s in that word. For example, the word 0x937a (1001001101111010) has a maximum of 4 consecutive 1s. The code in the figure calculates the number of consecutive 1s for the data 0x90abcedf.

```
.include "nios_macros.s"

.text
.equ  TEST_NUM, 0x90abcdef   /* The number to be tested */

.global _start
_start:

    movia   r7, TEST_NUM      /* Initialize r7 with the number to be tested */
    mov     r4, r7            /* Copy the number to r4 */

STRING_COUNTER:
    mov     r2, r0            /* Initialize the counter to zero */

STRING_COUNTER_LOOP:    /* Loop until the number has no more ones */
    beq     r4, r0, END_STRING_COUNTER

    srli    r5, r4, 1         /* Calculate the number for ones by shifting the */
    and     r4, r4, r5        /*   number by 1 and anding the result with itself. */
    addi    r2, r2, 1         /*   Increment the counter. */
    br      STRING_COUNTER_LOOP

END_STRING_COUNTER:
    mov     r16, r2           /* Store the result into r16 */

END:
    br      END               /* Wait here once the program has completed */
.end
```

Figure 2. Assembly-language code that counts consecutive ones.

Assemble and execute the program in Figure 2 as follows:

1. Open the Altera Debug Client and configure it to use the system created in Part I and the application program in Figure 2.

2. Compile and load the program.

3. Single step through the program. Watch how the instructions change the data in the processor's registers. Notice that when the end of the program has been reached, a result of 4 is in the register *r16*.

4. Set the Program Counter to 0x00000008. This will allow us to execute the program again (in step 6 below), while skipping the first two instructions.

5. This time add a breakpoint at address 0x28, so that the program will automatically stop executing at the end of the program.

6. Set register *r7* to 0xabcdef90. How many consecutive 1s are there in this number? Rerun the program by pressing F3 (Continue) to see if you are correct.

Part III

Instructions are also represented as strings of 1s and 0s, similar to data. In this part, we will examine how instructions are formed.

Perform the following:

1. Reload your program (by selecting Actions > Load) to remove the memory edits done in Part II. Then, execute the program once, stopping at the end.

2. Use the *Nios II Processor Reference Handbook*, which is available on Altera's website, to determine the machine instruction representation of the following assembly language instructions: `and r3, r7, r16` and `sra r7, r7, r3`.

3. Use the Altera Debug Client's memory-fill functionality to place these two instructions at memory locations 0 and 4. We should note that you will not see these updated values in the disassembly view of the Debug Client.

4. Set the Program Counter to 0x00000000. What will happen this time? To verify your answer, single step the instructions you placed at addresses 0 and 4 (to see their effect) and then execute the rest of the program.

5. Using the memory-fill feature change the last branch instruction to point to the start of the program instead of to itself. This will eliminate the need to manually edit the Program Counter.

6. Rerun the program until the number of 1s and the data being tested remain constant.

7. Now repeat steps 1 to 6, but use the instruction `srl r7, r7, r3` instead of `sra r7, r7, r3`. What is the difference?

Part IV

In most application programs there are portions of the code that will be executed multiple times from various locations inside a program. Such portions of code can be realized in the form of subroutines. A subroutine can be run from anywhere in the program by using a `call` instruction. The program execution returns to the call location after the subroutine has finished executing, if the subroutine ends with a `ret` instruction. We will now create a subroutine to calculate the number of consecutive 1s, and use it to calculate the number of consecutive 1s and the number of consecutive 0s in a given data word.

Start with the program for Part II and edit it as follows:

1. Take the code which calculates the number of consecutive 1s and make it into a subroutine. Have the subroutine use register *r4* to receive the input data and register *r2* for outputting the result.

2. Call the newly created subroutine twice, once to calculate the number of consecutive 1s and once to calculate the number of consecutive 0s. To calculate the number of consecutive 0s the input data must be inverted before running the subroutine.

3. Write the number of consecutive 1s into register *r16* and the number of consecutive 0s into register *r17*.

Part V

One might be interested in the longest string of alternating 1s and 0s. For example, the binary number 101101010001 has a string of 6 alternating 1s and 0s, as highlighted here: 101**101010**001. Use the subroutine created in Part IV to count the number of consecutive bits of alternating 1s and 0s. Write the result to register *r18*. Assume that the two end bits can be part of the longest string. For example, 1010 has 4 consecutive bits of alternating 1s and 0s. (Hint: What happens when the number is shifted to the right or left by 1 and XORed with the original number.)

Part VI

Perform the previous parts of this exercise using the C programming language. Create a function called *count_ones*, which counts the number of consecutive 1s. Search through the disassembled code to find both the *main* and *count_ones* subroutines. Did you write your assembly code in a similar way? What registers did the compiler use and why?