Laboratory Exercise 6

Using C code with the Nios II Processor

This is an exercise in using C code with the Nios II processor in a DE-Series computer system. We will use the *Intel FPGA Monitor Program* software to compile, load, and run application programs written in the C language. In this exercise you have to be familiar with both the C language and the Nios II assembly language. You should read the parts of the Monitor Program tutorial that discuss the use of C code. This tutorial can be accessed from Intel's FPGA University Program website, or by selecting Help > Tutorial within the Monitor Program software. You also need to be familiar with a number of I/O ports in the predesigned computer system for your DE-series board, including the parallel ports connected to the red LEDs, 7-segment displays, and pushbutton switches, as well as the Interval Timer port. These I/O ports are described in the documentation for your board's computer.

Part I

In Exercise 1, Part II, you were given a program in the Nios II assembly language that finds the largest number in a list of 32-bit integers that is stored in the memory. This code is reproduced in Figure 1. For this exercise you are to write a C-language program that implements this task.
Perform the following steps.

1. Write your C code in a file called *part1.c*. You should use the *printf* library function to display the result produced by the program. To use the *printf* function you have to include the *stdio.h* library header file in your C program by using the statement

   #include <stdio.h>

   To include a list of data words in the C program, you can declare them as an array using a statement such as

   int LIST[8] = {7, 4, 5, 3, 6, 1, 8, 2}; // number of elements, element 1, element 2, ...

2. Make a new Monitor Program project for this part of the exercise. In the Monitor Program screen shown in Figure 2 select C Program in the *Program Type* dropdown menu, and on the screen that follows select your *part1.c* file. In the screen of Figure 3 set the *Terminal device* to JTAG_UART. This setting causes the output of the *printf* library function to appear in the *Terminal* window of the Monitor Program graphical user interface.

   Compile and download your program. Examine the disassembled code and compare it to the code shown in Figure 1. To see the assembly code corresponding to your C source code, use the **Goto instruction** dialog box in the Monitor Program's Disassembly window. As illustrated in Figure 4,

type main in the dialog box and then click on the Go button to display your code. When you run the program, the results produced by the *printf* function should appear in the *Terminal* window as indicated in the figure.

```
/* Program that finds the largest number in a list of integers */
                .text
                .global  _start
_start:
                movia    r8, RESULT      # r8 points to result location
                ldw      r4, 4(r8)       # r4 is a counter, initialize it with N
                addi     r5, r8, 8       # r5 points to the first number
                ldw      r2, (r5)        # r2 holds the largest number found so far
LOOP:           subi     r4, r4, 1       # decrement the counter
                beq      r4, r0, DONE    # finished if r4 is equal to 0
                addi     r5, r5, 4       # increment the list pointer
                ldw      r6, (r5)        # get the next number
                bge      r2, r6, LOOP    # check if larger number found
                mov      r2, r6          # update the largest number found
                br       LOOP
DONE:           stw      r2, (r8)        # store the largest number into RESULT

STOP:           br       STOP            # remain here when done

RESULT:         .skip    4               # space for the largest number found
N:              .word    7               # number of entries in the list
NUMBERS:        .word    4, 5, 3, 6      # numbers in the list . . .
                .word    1, 8, 2         # . . .

                .end
```

Figure 1: Assembly-language program that finds the largest number.

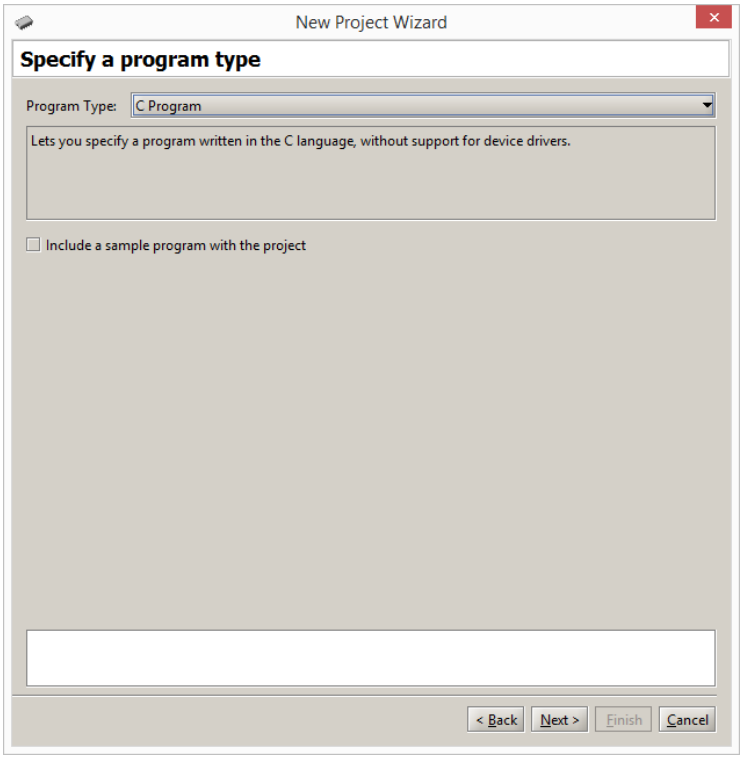

Figure 2: Setting the program type.

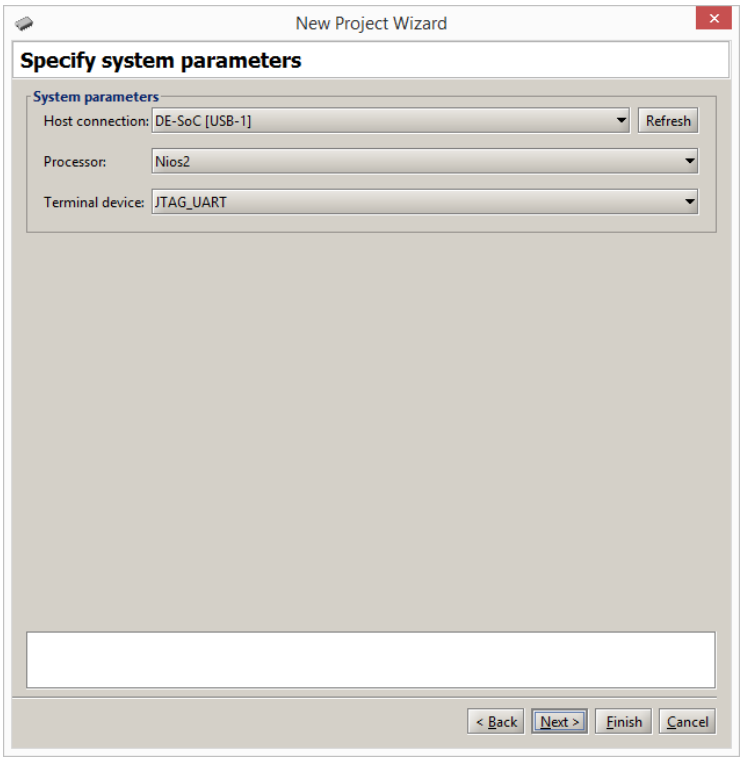

Figure 3: Configuring the *Terminal* window.

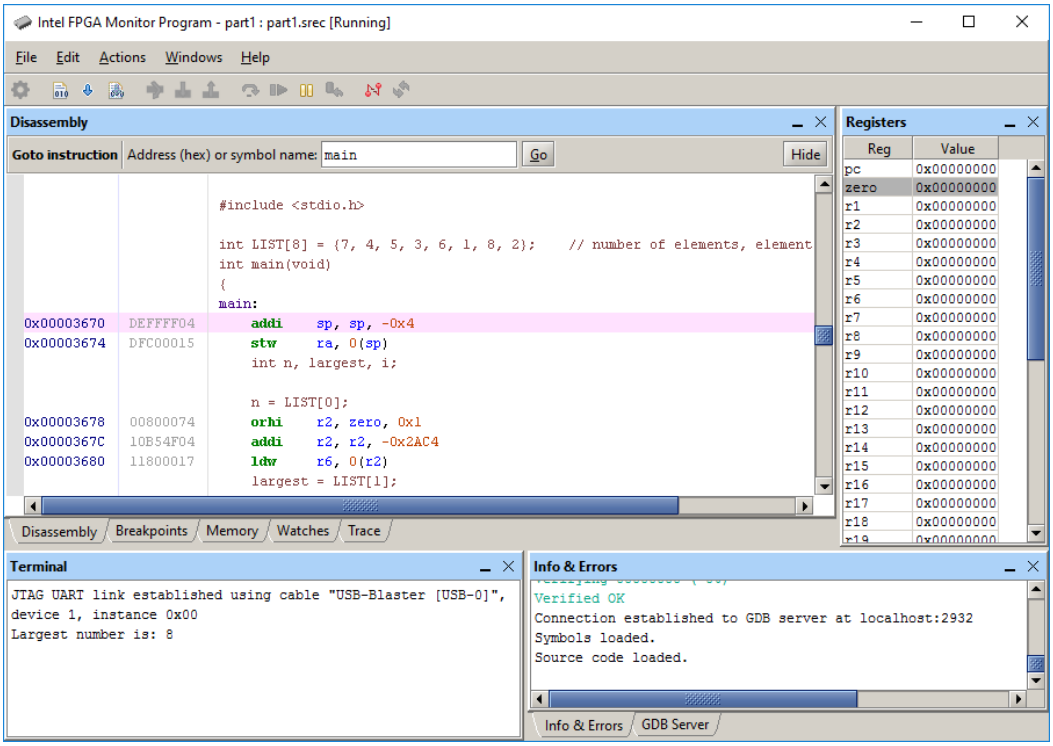

Figure 4: Displaying the code for the C program.

Part II

Using the *printf* function results in a fairly large number of assembly-language instructions, because the standard library routines are quite complex. Modify your program to display the result on the red lights *LEDR*, instead of using the *printf* statement. The parallel port in the computer systems is connected to the red lights is memory-mapped at the address 0xFF200000, as illustrated in Figure 5.

Compile, download, and run this program. Observe the difference in the size of the machine code for this program as compared to the one from Part I.
Perform the following steps.

1. Write your C code in a file called *part1.c*. You should use the *printf* library function to display the result produced by the program. To use the *printf* function you have to include the *stdio.h* library header file in your C program by using the statement

   #include <stdio.h>

   To include a list of data words in the C program, you can declare them as an array using a statement such as

   int LIST[8] = {7, 4, 5, 3, 6, 1, 8, 2}; // number of elements, element 1, element 2, ...

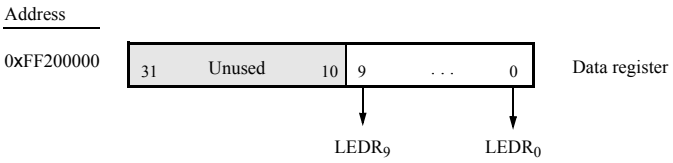

Figure 5: The parallel port connection to the red lights.

Part III

In Exercise 2, you were given a program that uses shift and AND operations to find the longest string of 1's in a word of data. The program is reproduced in Figure 6. In Parts III and IV of Exercise 2, you were asked to extend this program so that it processed a list of data words, rather than just one word. Also, the program was extended to compute the longest strings of 1's, the longest string of 0's, and the longest string of alternating 1's and 0's for any of the words in the list. The results of these computations were to be shown on the 7-segment displays of the computer. For this part of the exercise, you are to write a C-language program to implement these tasks.

```
/* Program that counts consecutive ones */
            .text
            .global  _start
_start:
            ldw      r9, TEST_NUM(r0)     /* Load the data into r9 */

            mov      r10, r0              /* r10 will hold the result */
LOOP:       beq      r9, r0, END          /* Loop until r9 contains no more 1s */
            srli     r11, r9, 0x01        /* Count the 1s by shifting the number and */
            and      r9, r9, r11          /* ANDing it with the shifted result */
            addi     r10, r10, 0x01       /* Increment the counter */
            br       LOOP

END:        br       END                  /* Wait here */

TEST_NUM:   .word    0x3fabedef           /* The number to be tested */
            .end
```

Figure 6: Assembly-language program that counts consecutive ones.

To include the list of data words in your C program, you can declare them as an array using a statement such as

```
int TEST_NUM[ ] = {0x0000e000, 0x3fabedef, 0x00000001, 0x00000002, 0x75a5a5a5,
                   0x01ffC000, 0x03ffC000, 0x55555555, 0x77777777, 0x08888888,
                   0x00000000};
```

Display the count for the longest string of 1's on 7-segment displays $HEX1 - 0$, for the longest string of 0's on $HEX3 - 2$, and for alternating 1's and 0's on $HEX5 - 4$. The parallel ports connected to the 7-segment

displays in the computer systems are illustrated in Figure 7.

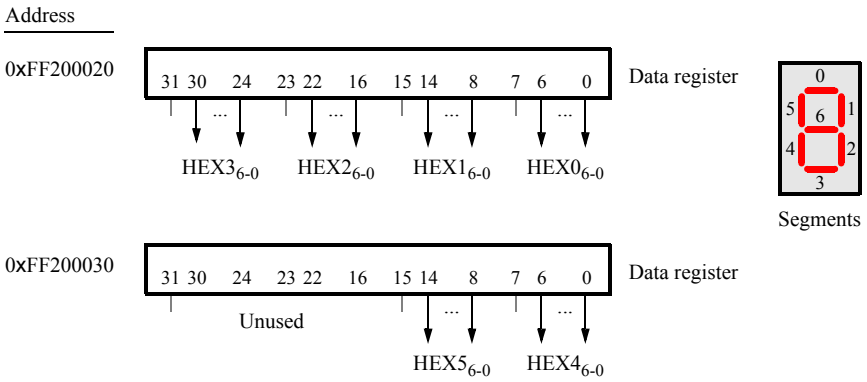

Figure 7: The parallel ports connected to the 7-segment displays.

Create a new folder and Monitor Program project for your C program, and then compile, download, and test the code. Using the ten words of test data shown above, the correct result that should appear on the $HEX5 - 0$ displays is 32 31 12.

Part IV

In Exercise 4 you were asked to implement a real-time clock in the DE1-SoC Computer. The clock-time was shown on the $HEX3-0$ seven-segment displays in the format SS:DD, with *SS* representing seconds and *DD* representing hundredths of a second. Time was measured in intervals of 0.01 seconds by using polled I/O with the Interval Timer, and the clock could be stopped/run by pressing one of the pushbutton *KEYs*.

In this part of the exercise you are to write a C program that implements a real-time clock. Display the clock-time on the 7-segment displays $HEX5-0$ in the format MM:SS:DD, where where *MM* are minutes, *SS* are seconds, and *DD* are hundredths of a second. Measure time intervals of 0.01 seconds in your program by using polled I/O with the Interval Timer. You should be able to stop/run the clock by pressing any pushbutton *KEY*. When the clock reaches 59:59:99, it should wrap around to 00:00:00.

Make a new folder to hold your Monitor Program project for this part. Create a file called *part4.c* and type your C code into this file. Make a new Monitor Program project for this part of the exercise, and then compile, download, and test your program.

Part V

Write a C program that scrolls the word intEL in the right-to-left direction across the 7-segment displays. An example of the scrolling behaviour is given in Table 1. You should scroll the display at a rate of 0.2 seconds per character. You should be able to stop/run the scrolling message by pressing the *KEY* pushbuttons.

Time slot	Display
0	i n t E L
1	n t E L
2	t E L
3	E L
4	L
5	
6	i
7	i n
...	...

Table 1: Scrolling the message intEL on $HEX5-0$.

Note that scrolling a message across the 7-segment displays is similar in nature to the task of implementing a real-time clock, from Part IV. You should be able to reuse most of your code from Part IV. But instead of updating the clock each time the Interval Timer expires, you need to update the scrolling message.

Make a new folder to hold your Monitor Program project for this part. Create a file called *part5.c* and type your C code into this file. Make a new Monitor Program project, compile, download, and test your program.