Laboratory Exercise 6

Using C code with the Nios II Processor

This is an exercise in using C code with the Nios II processor. We will use the *Aardvark 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 DE0-Nano-SoC or DE-Nano Computers (depending on which board you are using), including the parallel ports connected to the green LEDs, and pushbutton switches, as well as the Interval Timer port. These I/O ports are described in the documentation for the DE0-Nano-SoC and DE0-Nano Computers.

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 green *LED* lights to display the result produced by the program. The parallel port in the DE0-Nano-SoC Computer connected to the green lights is memory-mapped at the address 0xFF200000, as illustrated in Figure 5. If you are using the DE0-Nano board, the green *LED* lights are memory-mapped at address 0x10000010, not 0xFF200000.

   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 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 result produced by the program should be displayed in binary format on the green *LED* lights.

```
/* 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.

Part II

On the DE0-Nano and DE0-Nano-SoC Computers, the Nios II processor does not have access to enough memory to use library functions like *printf* (in the *stdio.h* library). In this section you will write C language functions to display the result of your program written in Part I on the JTAG terminal window. You should write functions to:

- Print a single ASCII character to the JTAG UART terminal
- Print a null terminated string (*char** or *char*[])
- Convert an integer into a string
- Print an integer to the JTAG UART terminal

Theses functions should be similar to the subroutines used to display the *COUNT* variable in Exercise 4, Part IV. Modify your program from Part I to print a message to the JTAG UART terminal with the results, instead of using the green lights *LED*.

Compile, download, and run this program.

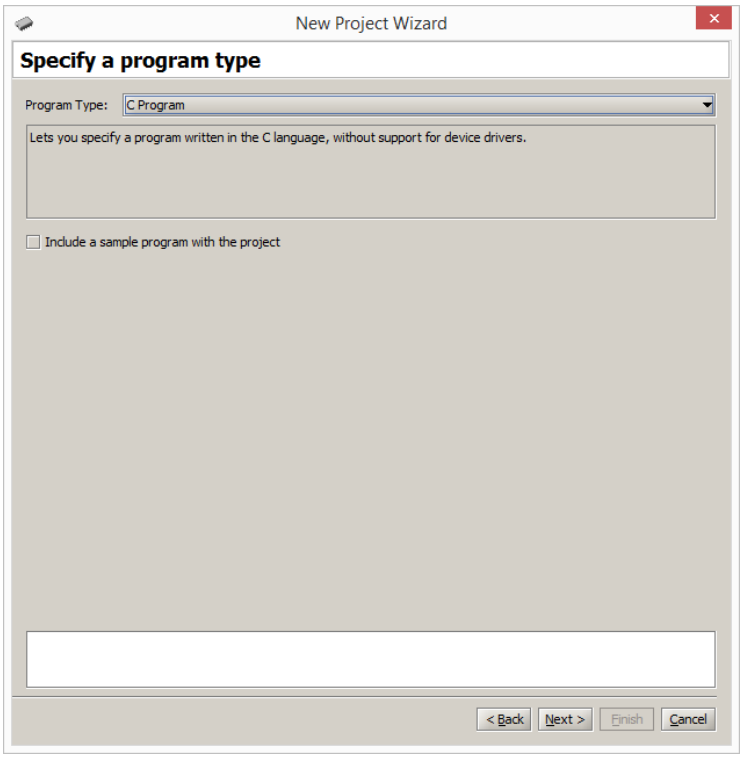

Figure 2: Setting the program type.

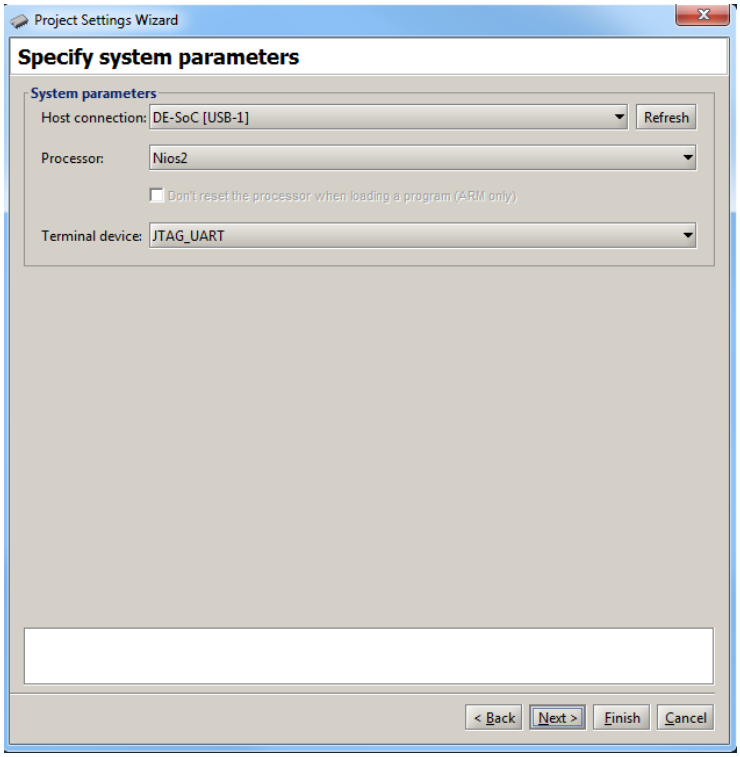

Figure 3: Configuring the *Terminal* window.

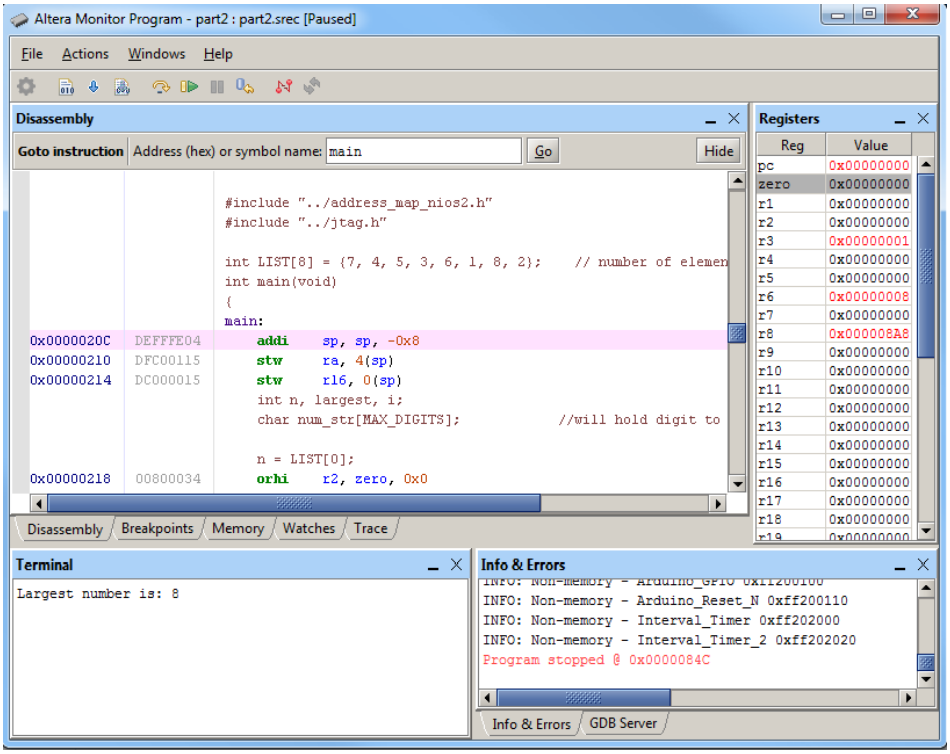

Figure 4: Displaying the code for the C program.

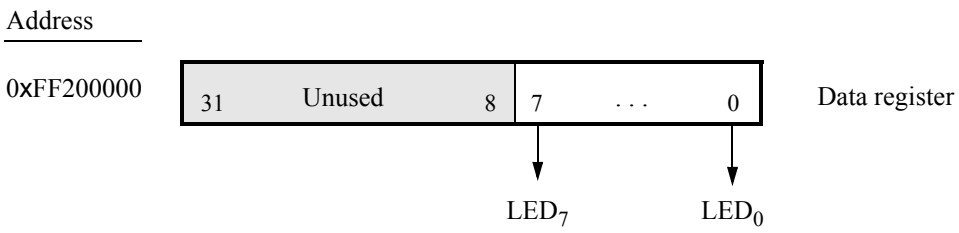

Figure 5: The parallel port connected to the green lights *LED* on the DE0-Nano-SoC Computer.

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 Terminal window of the Monitor Program. 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, 0's, and alternating 1's and 0's on the Terminal window.

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 Terminal window is:

Longest string of ones: 12
Longest string of zeros: 31
Longest string of alternating ones/zeros: 32

Part IV

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 Terminal window in the format MM:SS:DD, where *MM* are minutes, *SS* are seconds, and *DD* are hundredths of a second. Pressing pushbutton KEY_0 should reset the time and display 00:00:00 on the next line of the Terminal window. Since you cannot update the Terminal window at the rate of 1/100 seconds (the communication speed with the Terminal is too slow), you should display the current time, on the next line of the Terminal, only when pushbutton KEY_1 is pressed. Measure time intervals of 0.01 seconds in your program by using polled I/O with the Interval Timer (a similar exercise was described in Part IV of Exercise 4, but using time intervals of 1 second). When the clock reaches 59:59:99 it should wrap around to 00:00:00. Note that the DE0-Nano is not able to use KEY_0 because it is hardwired to reset the processor. If you are using the DE0-Nano board (and not the DE0-Nano-SoC board), simply ignore the resetting with KEY_0 feature.

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 message DE0-Nano-SoC (or any message of your choice) in the right-to-left direction on the Terminal window. The contents of the display as the message scrolls should appear as illustrated in Table 1. You should scroll the display at a rate of 0.25 seconds per character. You should be able to stop/run the scrolling message by pressing any pushbutton *KEY*. It is not necessary to scroll the message all the way from the right-hand side of the Terminal window; just start the message far enough to the right so that you can achieve the scrolling-effect illustrated in Table 1.

Time slot	Display											
0												D
1											D	E
2										D	E	0
3									D	E	0	-
4								D	E	0	-	N
5							D	E	0	-	N	a
6						D	E	0	-	N	a	n
7					D	E	0	-	N	a	n	o
8				D	E	0	-	N	a	n	o	-
9			D	E	0	-	N	a	n	o	-	S
10		D	E	0	-	N	a	n	o	-	S	o
11	D	E	0	-	N	a	n	o	-	S	o	C
12	E	0	-	N	a	n	o	-	S	o	C	
13	0	-	N	a	n	o	-	S	o	C		
14	-	N	a	n	o	-	S	o	C			
15	N	a	n	o	-	S	o	C				
...	.	.	.									

Table 1. Scrolling the message DE0-Nano-SoC on the Terminal window.

You may want to make use of the *video-terminal* command shown below. Sending this string of characters to the Terminal window causes it to return the "cursor" to the top-left corner of the window. The command can be declared as a string, named *home* in this example, that can be sent to the Terminal window:

```
char home[ ] = "\033[H";
```

Note that scrolling a message across the Terminal window 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 display each time the A9 Private 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.