

Laboratory Exercise 1

Using an Altera Nios II System

This is an introductory exercise that involves Altera's Nios II processor. It uses a predefined computer system, called the *DE-series Media Computer*, which includes the Nios II processor. The system is implemented as a circuit that is downloaded into the FPGA device on an Altera DE-series board. This exercise illustrates how programs written in the Nios II assembly language or in the C language can be executed on the DE-series board. We will use the *Altera Monitor Program* software to compile, load and run the application programs.

For this exercise you have to know the Nios II processor architecture and its assembly language, as well as the C language. Read the tutorial *Introduction to the Altera Nios II Soft Processor*. You also have to become familiar with the monitor program; read the tutorial *Altera Monitor Program*. Both tutorials are available on Altera's University Program web site. The monitor tutorial is also included in the Altera Monitor Program package – it can be accessed by selecting **Help > Tutorial** in the monitor window.

Part I

In this part you will use the Altera Monitor Program to download the DE-series Media Computer circuit into the FPGA device and execute a sample program. Perform the following:

1. Turn on the power to the Altera DE-series board.
2. Open the Altera Monitor Program, which leads to the window in Figure 1.

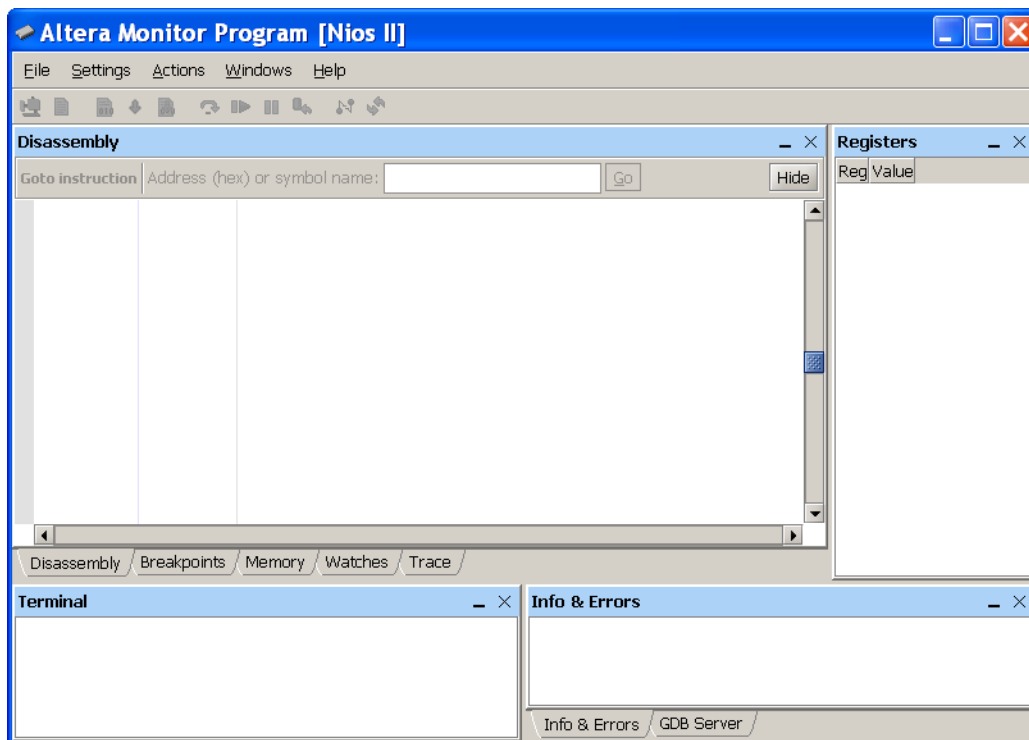


Figure 1. The Altera Monitor Program window.

To run an application program it is necessary to create a new project. Select **File > New Project** to reach the window in Figure 2. Give the project a name and indicate the directory for the project; we have chosen the project name *lab1_part1* in the directory *embedded_lab1_part1*, as indicated in the figure. Click **Next**, to get the window in Figure 3.

3. Now, you can select your own custom system (if you have one) or a predesigned (by Altera) system. Choose the DE-series Media Computer and click **Next**. The display in the window will now show where files that implement the chosen system are located. This is for information purpose only; if you wanted to use a system that you designed by using Altera's Quartus II software, you would have to provide such files. Click **Next**.
4. In the window in Figure 4 you can specify the type of application programs that you wish to run. They can be written in either the Nios II assembly language or the C programming language. Specify that an assembly language program will be used. The Altera Monitor Program package contains several sample programs. Select the box **Include a sample program with the project**. Then, choose the **Test Media Computer** program, as indicated in the figure, and click **Next**.
5. The window in Figure 5 is used to specify the source file(s) that contain the application program(s). Since we have selected the *Test Media Computer* program, the window indicates the files that are used by this program. This window also allows the user to specify the starting point in the selected application program. The default symbol is *_start*, which is used in the selected sample program. Click **Next**.
6. The window in Figure 6 indicates some system parameters. Note that the *USB-Blaster* cable is selected to provide the connection between the DE-series board and the host computer. Click **Next**.

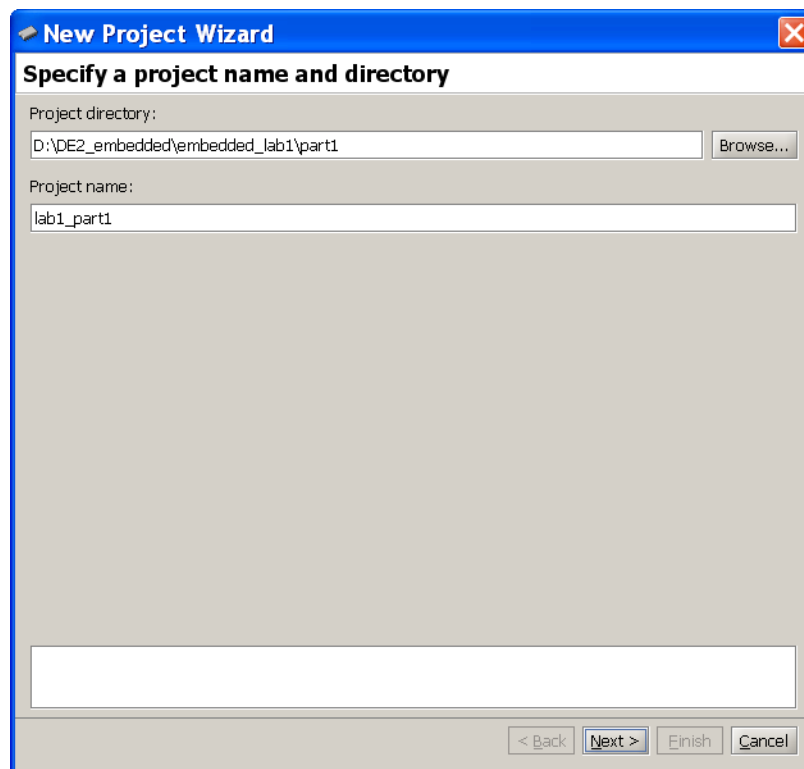


Figure 2. Specify the directory and the name of the project.

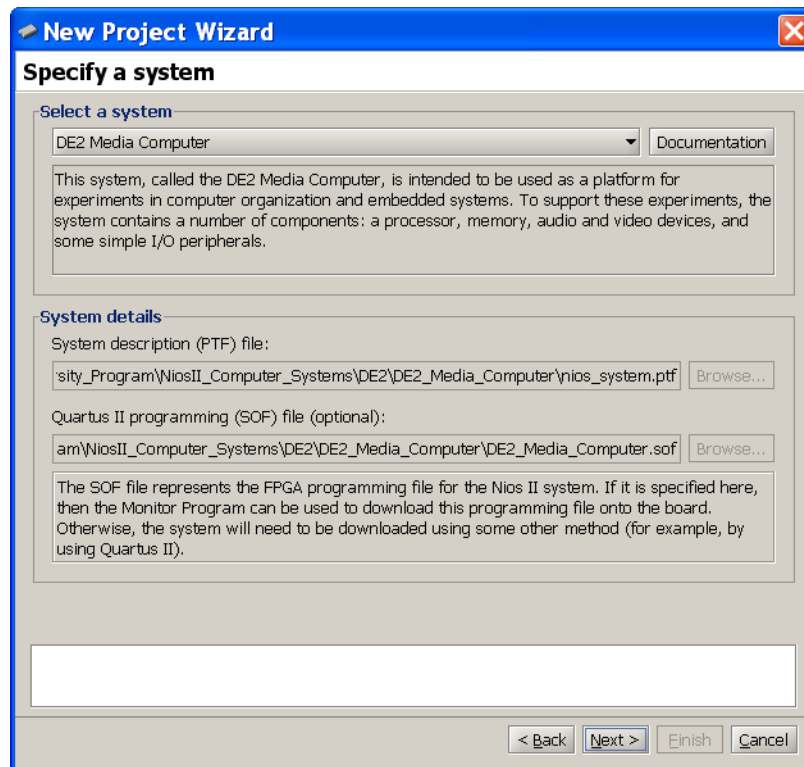


Figure 3. Specification of the system.

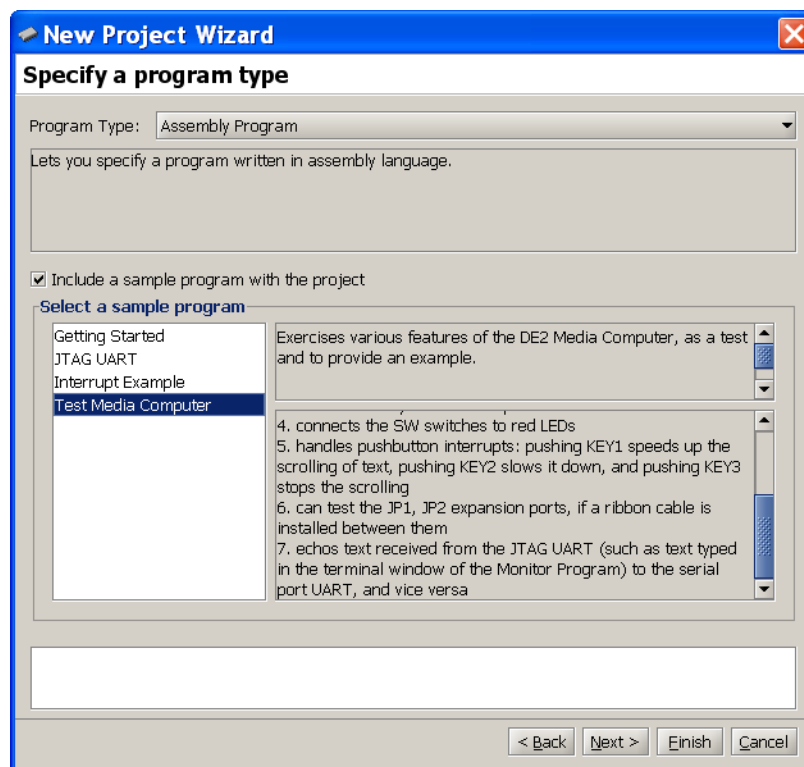


Figure 4. Selection of an application program.

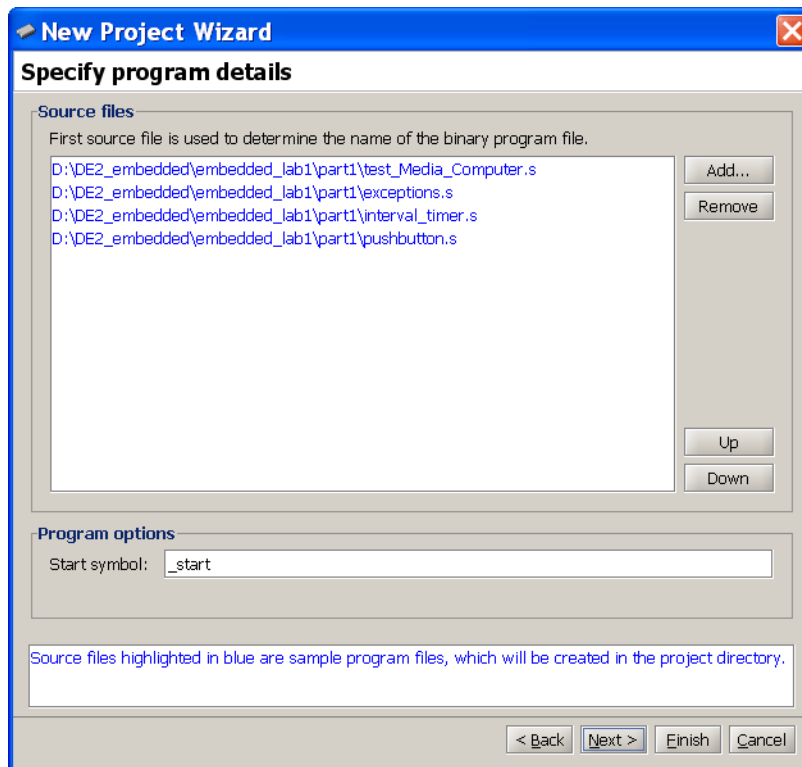


Figure 5. Source files used by the application program.

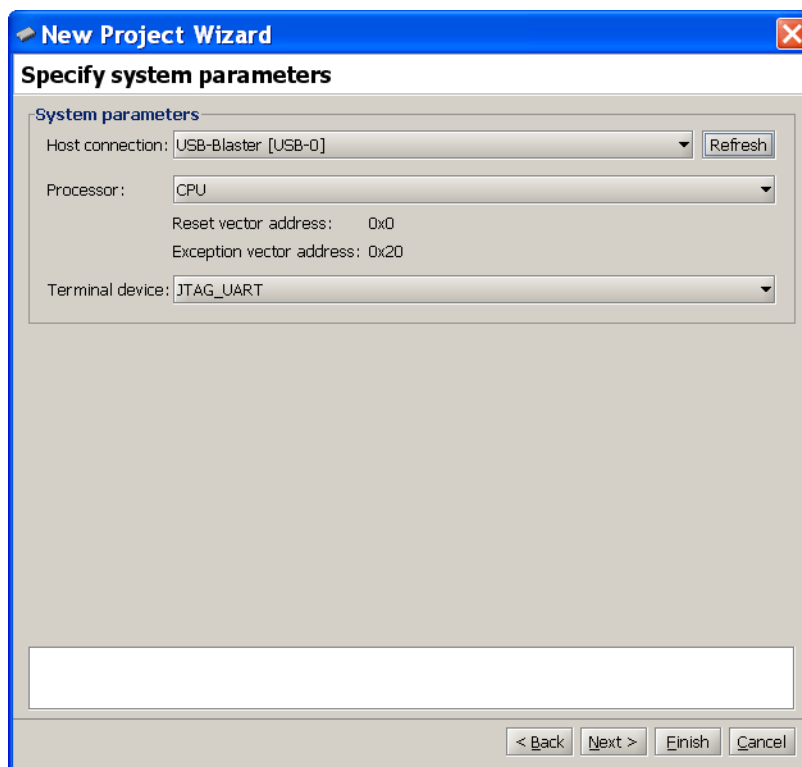


Figure 6. Specify the system parameters.

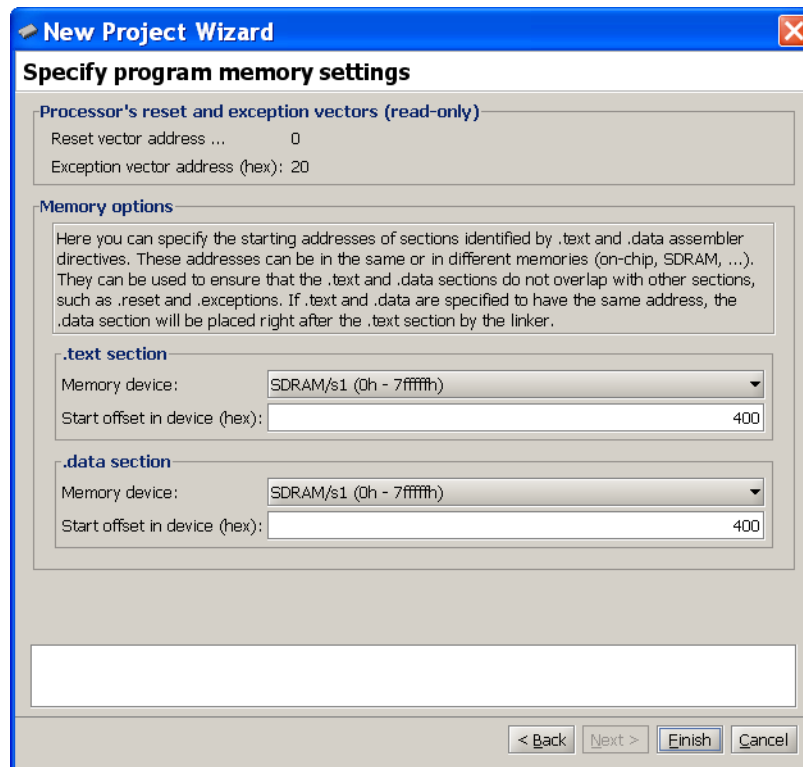





Figure 7. Specify the program memory settings.

7. The window in Figure 7 displays the preselected components of the DE-series Media Computer. Observe that the SDRAM is selected as the memory device to be used. The start offset is 0x400 (hex 400), which means that the application program will be loaded in the memory locations that begin at address 400. Since this choice was made by the designer of the sample program, you cannot change the selection in Figure 7. Click Finish to complete the specification of the new project.
8. Since you specified a new project, a pop-up box will appear asking you if you want to download the system associated with this project onto the DE-series board. Make sure that the power to the DE-series board is turned on and click **Yes**. Watch the change in state of the blue LEDs on the DE-series board that correspond to LOAD and GOOD, which will blink as the circuit is being downloaded. A pop-up box will appear informing you that the circuit has been successfully downloaded - click **OK**. If the circuit is not successfully downloaded, make sure that the USB connection, through which the USB-Blaster communicates, is established and recognized by the host computer. (If there is a problem, a possible remedy may be to unplug the USB cable and then plug it back in.)
9. Having downloaded the DE-series Media Computer into the FPGA chip on the DE-series board, we can now load and run programs on this computer. In the main monitor window, shown in Figure 8, select **Actions > Compile & Load** to load the selected sample program into the FPGA chip. Figure 9 shows the monitor window after the sample program has been loaded.
10. Run the program by selecting **Actions > Continue** or by clicking on the toolbar icon , and observe the test displayed on the LEDs and 7-segment displays. This test provides an indication that the DE-series board is functioning properly.
11. Stop the execution of the sample program by clicking on the icon , and disconnect from this session by clicking on the icon .

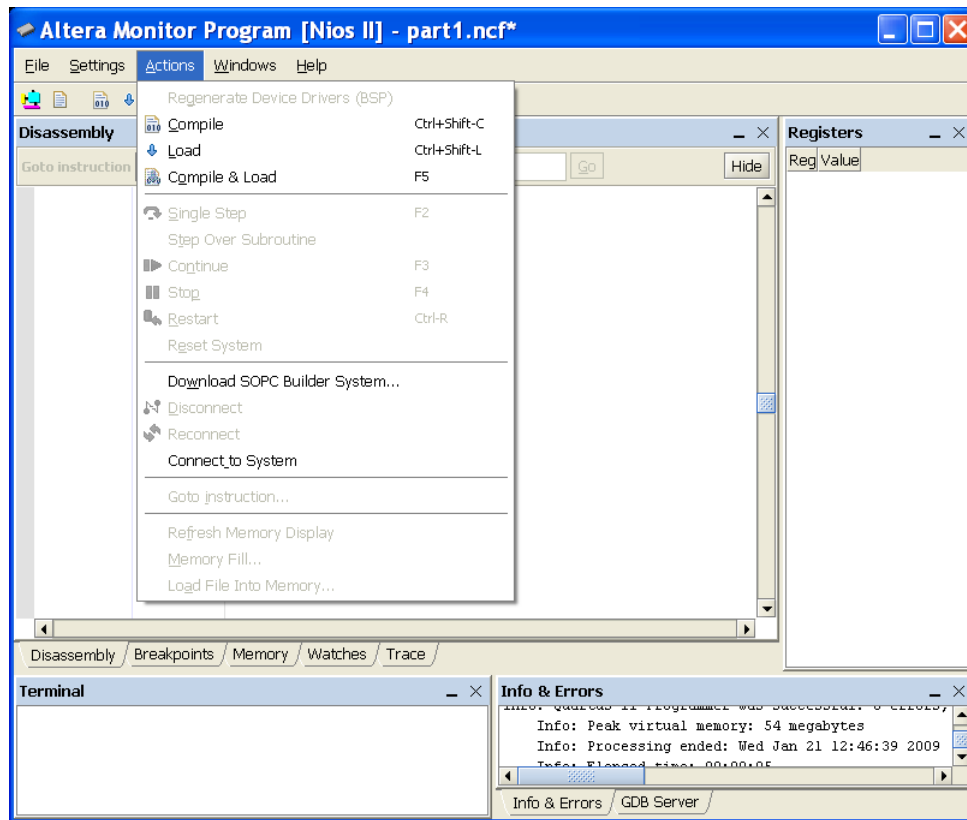


Figure 8. Specify an action in the monitor window.

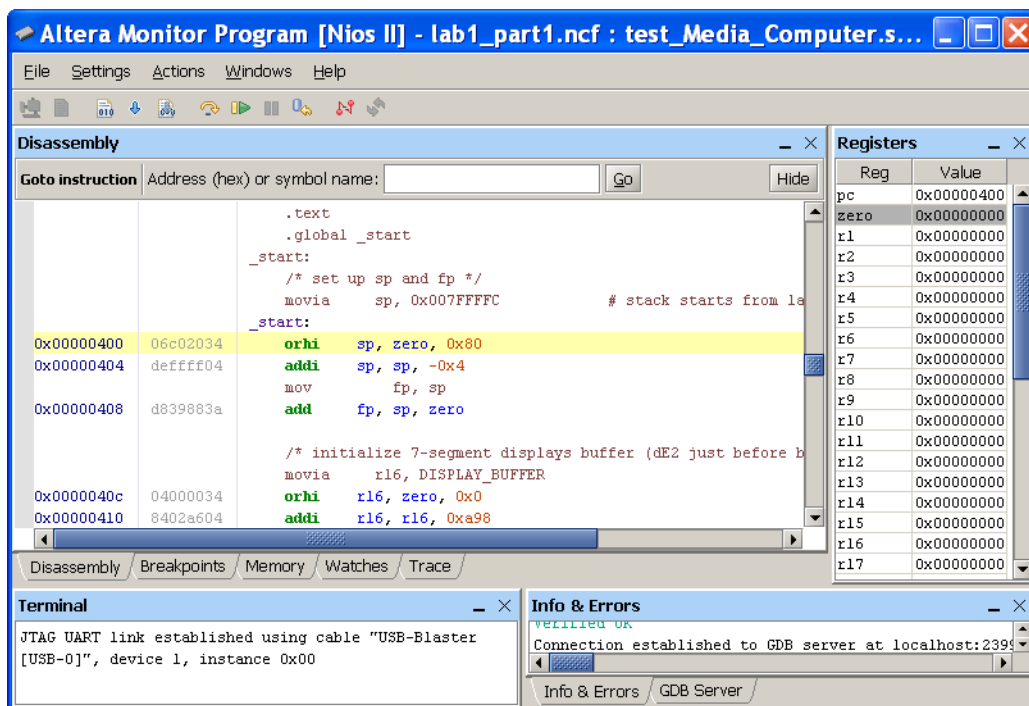


Figure 9. The monitor window showing the loaded sample program.

Part II

Now, we will explore some features of the Altera Monitor Program by using a simple application program written in the Nios II assembly language. Consider the program in Figure 10, which finds the largest number in a list of 32-bit integers that is stored in the memory. This program is available in the file *embedded_lab1_part2.s*.

```
/* Program that finds the largest number in a list of integers */
.equ LIST, 0x800          /* Starting address of the list */

.global _start
_start:
    movia    r4, LIST      /* r4 points to the start of the list */
    ldw      r5, 4(r4)      /* r5 is a counter, initialize it with n */
    addi     r6, r4, 8      /* r6 points to the first number */
    ldw      r7, (r6)       /* r7 holds the largest number found so far */
LOOP:
    subi     r5, r5, 1      /* Decrement the counter */
    beq      r5, r0, DONE   /* Finished if r5 is equal to 0 */
    addi     r6, r6, 4      /* Increment the list pointer */
    ldw      r8, (r6)       /* Get the next number */
    bge      r7, r8, LOOP    /* Check if larger number found */
    add      r7, r8, r0      /* Update the largest number found */
    br       LOOP
DONE:
    stw      r7, (r4)        /* Store the largest number into RESULT */
STOP:
    br       STOP           /* Remain here if done */

.org    0x800
RESULT:
.skip    4                  /* Space for the largest number found */
N:
.word    7                  /* Number of entries in the list */
NUMBERS:
.word    4, 5, 3, 6, 1, 8, 2 /* Numbers in the list */
.end
```

Figure 10. Assembly-language program that finds the largest number.

Note that some sample data is included in this program. The list starts at hex address 800, as specified by the **.org** assembler directive. The first word (4 bytes) is reserved for storing the result, which will be the largest number found. The next word specifies the number of entries in the list. The words that follow give the actual numbers in the list.

Make sure that you understand the program in Figure 10 and the meaning of each instruction in it. Note the extensive use of comments in the program. You should always include meaningful comments in programs that you will write!

Perform the following:

1. Create a new directory; we have chosen the directory name *embedded_lab1_part2*. Copy the file *embedded_lab1_part2.s* into this directory.

2. Use the Altera Monitor Program to create a new project in this directory; we have chosen the project name *part2*. When you reach the window in Figure 4 choose **Assembly Program** but do not select a sample program, as shown in Figure 11. Click **Next**.
3. Upon reaching the window in Figure 5, you have to specify your program. Click **Add** and in the pop-up box that appears indicate the desired file name, *embedded_lab1_part2.s*, and its location. This should lead to the image in Figure 12. Click **Next** to get the window in Figure 6. Again click **Next** to get to the window in Figure 7. Make sure that the SDRAM is selected as the memory device. Note that the *Start offset in device* will be 0, because the program in Figure 10 does not indicate that it should be loaded at a location that is different from the default location 0. Click **Finish**.
4. Compile and load the program.

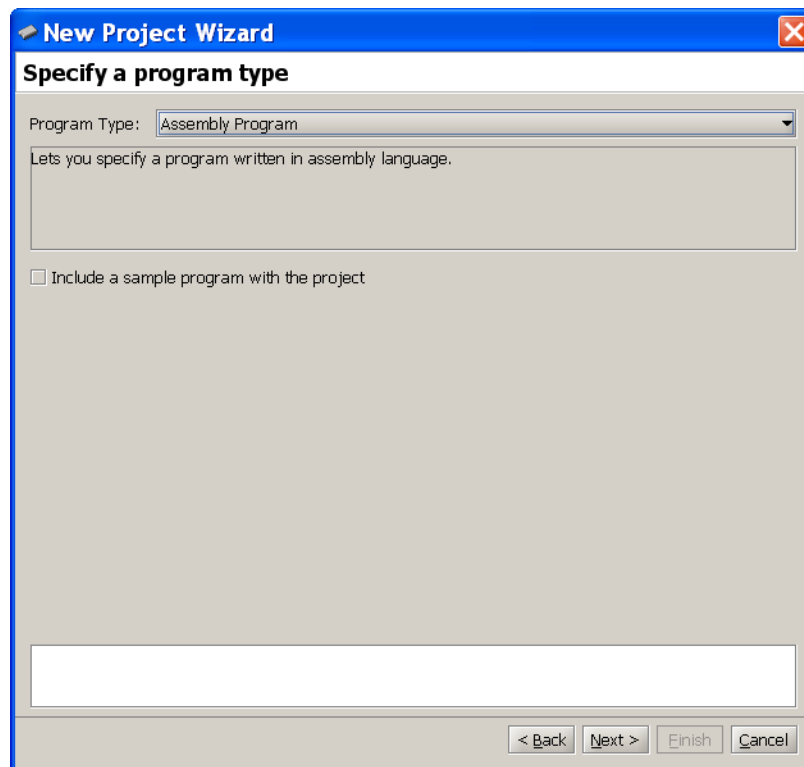


Figure 11. Select the assembly language program.

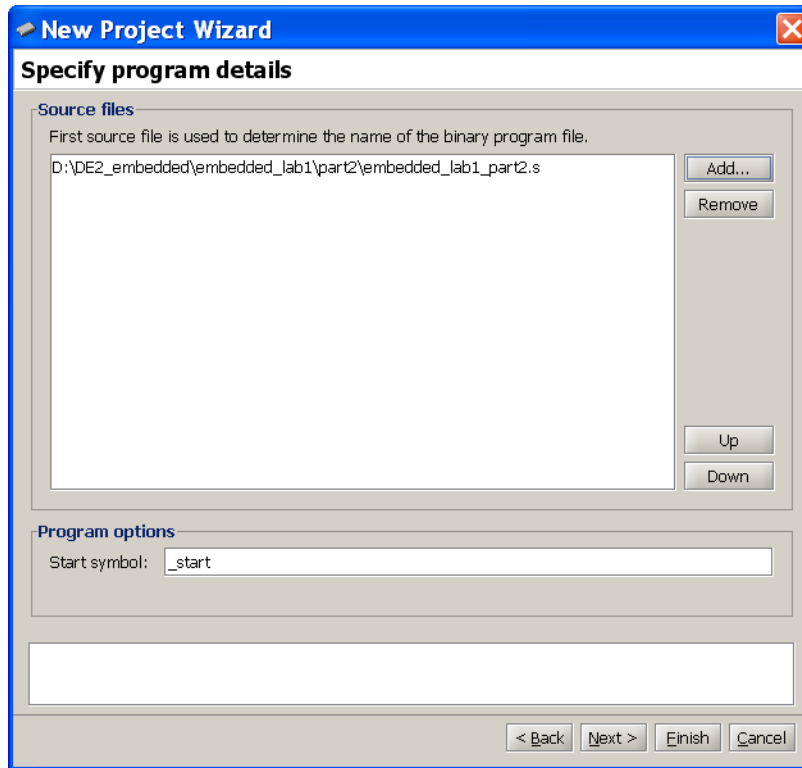




Figure 12. Select your source program.

5. The Monitor Program will display the disassembled view of the code loaded in the memory, as indicated in Figure 13. Note that the pseudoinstruction **movia** in the original program has been replaced with two machine instructions, **orhi** and **addi**, which load the 32-bit address LIST into register *r4* in two 16-bit parts (because an immediate operand value is restricted to 16 bits). Examine the disassembled code to see the difference in comparison with the original source program. Make sure that you understand the meaning of each instruction. Observe also that your program was loaded into memory locations with the starting address 0. These addresses correspond to the SDRAM memory, which was selected when specifying the system parameters. The DE-series Media Computer has two more memories which are the *on-chip* memory (i.e. the memory on the FPGA chip) and the SRAM chip on the DE-series board. See the document *Media Computer System for Altera DE-series Board* for full information. This document can be accessed by clicking on the Documentation button in Figure 3.
6. Run the program. When the program is running, you will not be able to see any changes (such as the contents of registers or memory locations) in the monitor windows, because the monitor program cannot communicate with the processor system on the DE-series board. But, if you stop the program, the present state of these components will be displayed. Do so and observe that the program has stopped executing at the last Branch instruction which is loaded in the memory location 0x34. Note that the largest number found in the sample list is 8 as indicated by the contents of register *r7*. This value is also stored in the memory location 0x800, which can be seen by opening the Memory tab of the monitor window (in Figure 1.13).
7. Return to the beginning of the program by clicking on the icon . Now, single step through the program by clicking on the icon . Watch how the instructions change the data in the processor's registers.

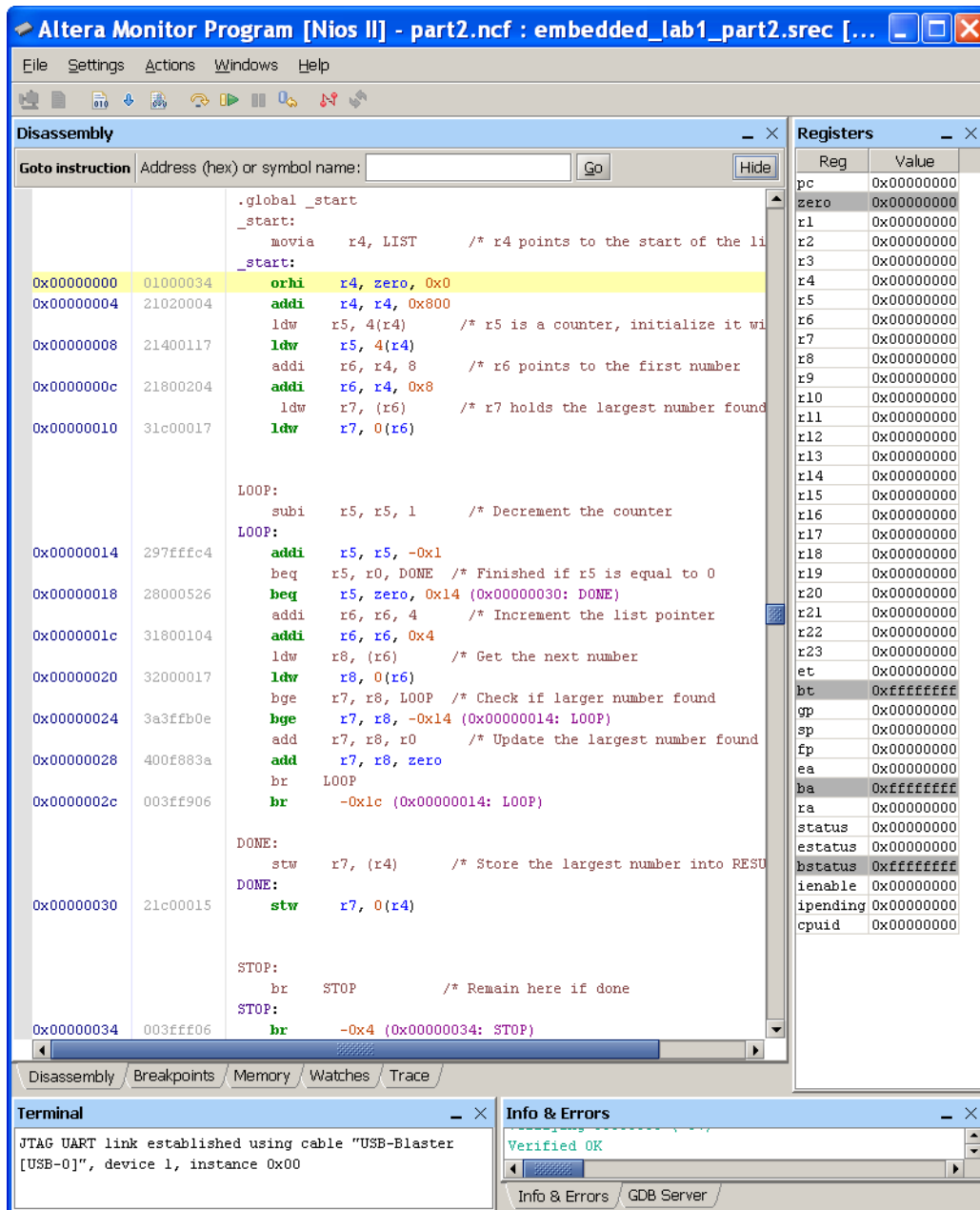




Figure 13. The disassembled view of the program in Figure 10.

8. Set the Program Counter to 0. Note that this action has the same effect as clicking on the restart icon .
9. This time add a breakpoint at address 0x2C (by clicking on the gray bar to the left of this address), so that the program will automatically stop executing whenever the Branch instruction at this location is about to be executed. Run the program and observe the contents of register *r7* each time this breakpoint is reached.
10. Remove the breakpoint (by clicking on it). Then, set the Program Counter to 0x8, which will bypass the first two instructions which load the address LIST into register *r4*. Also, set the value in register *r4* to 0x804. Run the program by clicking on the icon . What will be the result of this execution?

Part III

Implement the task in Part II by rewriting the program in Figure 10 in the form of a subroutine. The subroutine, `LARGE`, has to find the largest number in a list. The calling program should pass the number of entries and the address of the first number in the list as parameters to the subroutine via registers `r5` and `r6`. The subroutine should pass the value of the largest number to the calling program via register `r7`.

Create a new directory and a new Monitor Program project to compile and download your program. The Monitor Program can have only one project per directory! Run your program to verify its correctness.

Part IV

Modify your program from Part III so that the parameters and the result are passed between the calling program and the subroutine via the processor stack. Initialize the stack pointer, `sp`, to an appropriate address value.

Run your program to verify its correctness.

Part V

Write a C-language program to implement the task in Part II, in a file called `embedded_lab1_part5.c`. Use the `printf` statement to display the result in the Monitor Program's terminal window.

Create a new directory and copy your file into it. Then, create a new Monitor Program project. Choose C Program in the window in Figure 11. In the window in Figure 12, include your file in the project. Use the default Program Options as indicated in Figure 14.

Compile, download and run your program. Examine the disassembled code and compare it to the code produced in Part II.

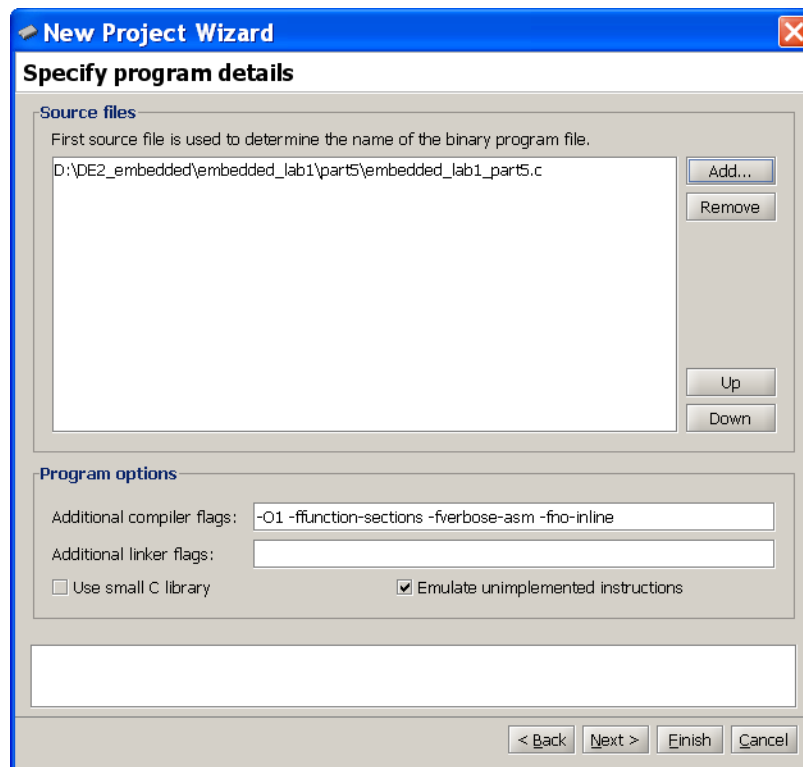


Figure 14. Select the C program.

Part VI

Using the **printf** statement results in a fairly large number of assembly-language instructions, because a standard library routine is used. Augment your program to display the result in a specific memory location, e.g. 0x6000, instead of using the **printf** statement. Compile and run this program, and observe the difference.

Preparation

Your preparation should include the following:

1. Read the tutorials *Introduction to the Altera Nios II Soft Processor* and *Altera Monitor Program*.
2. Write the assembly-language programs for Parts III and IV.
3. Write the C-language programs for Parts V and VI.

Copyright ©2011 Altera Corporation.