Laboratory Exercise 3

Subroutines and Stacks

The purpose of this exercise is to learn about subroutines and subroutine linkage in the Nios II environment. This includes the concepts of parameter passing and stacks.

Part I

We will use a Nios II system that includes an on-chip memory block and a JTAG UART module for communication with the host computer. Implement the system as follows:

1. Create a new Quartus II project. Select Cyclone IV E EP4CE115F29C7 as the target chip, which is the FPGA chip on the Altera DE2_115 board.

2. Use the SOPC Builder to create a system named *nios_system*, which includes the following components:
   • Nios II/s processor with JTAG Debug Module Level 1
   • On-chip memory - RAM mode and 32 Kbytes in size

3. From the **System** menu, select **Auto-Assign Base Addresses**. You should now have the system shown in Figure 1.

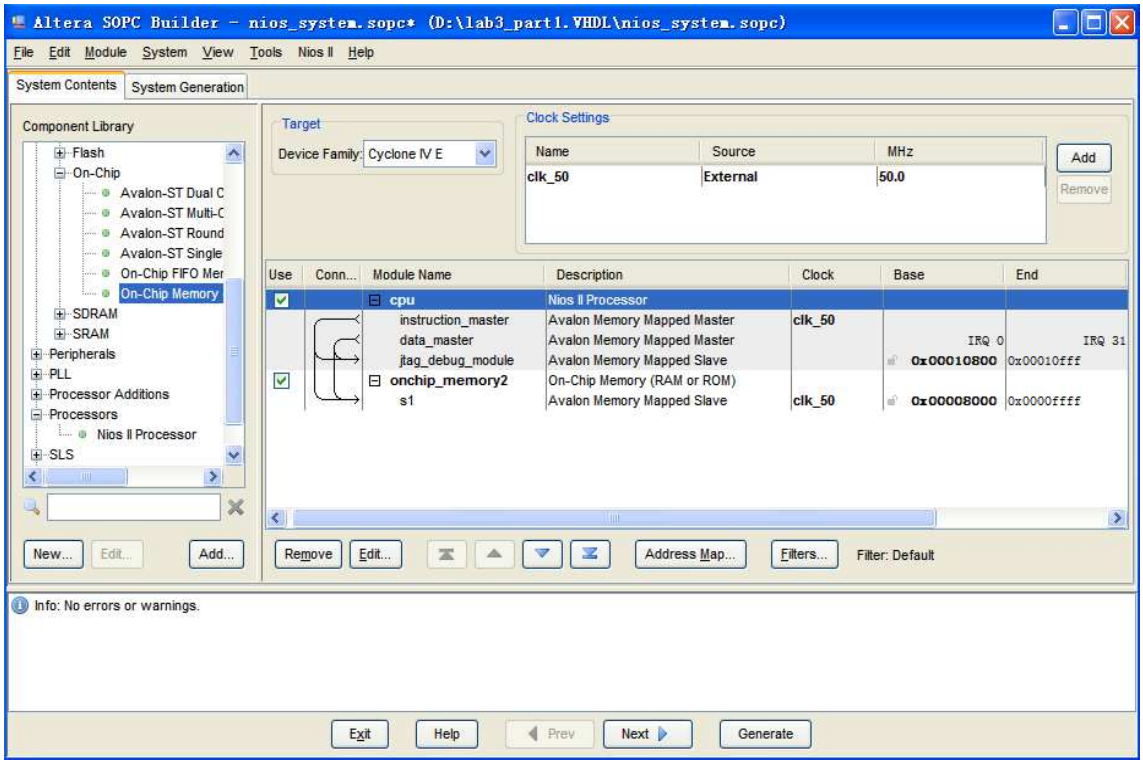

Figure 1. The Nios II system implemented by the SOPC Builder.

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 Y2** (This is a 50 MHz clock)
   • reset_n - **PIN M23** (This is the pushbutton switch *KEY*0)

7. Compile the Quartus II project.

8. Program and configure the Cyclone IV E FPGA on the DE2_115 board to implement the generated system.

Part II

We wish to sort a list of 32-bit positive numbers in descending order. The list is provided in the form of a file in which the first 32-bit entry gives the size (number of items) of the list and the rest of the entries are the numbers to be sorted. Implement the desired task, using the Nios II assembly language, as follows:

1. Write a program that can sort a list contained in a file that is loaded in the memory starting at location LIST_FILE. Assume that the list is large enough so that it cannot be replicated in the available memory. Therefore, the sorting process must be done “in place”, so that both the sorted list and the original list occupy the same memory locations.

2. Compile and download your program using the Altera Monitor Program.

3. Create a sample list, load it into the memory, and run your program.

Note: The file that contains the list can be loaded into the memory by using the Debug Client.

Part III

In this part, we will use a subroutine to realize the sorting task. To make the subroutine general, the contents of registers used by the subroutine have to be saved on the stack upon entering and restored before leaving the subroutine. Note that the stack has to be created by initializing the stack pointer *sp*, register *r27*. It is a common practice to start the stack at a high-address memory location and make it grow in the direction of lower addresses. The stack pointer has to be adjusted explicitly when an entry is placed on or removed from the stack. To make the stack grow from high to low addresses, the stack pointer has to be decremented by 4 before a new entry is placed on the stack and it has to be incremented by 4 after an entry is removed from the stack. Implement the previous task by modifying your program from Part II as follows:

1. Write a subroutine, called SORT, which can sort a list of any size placed at an arbitrary location in the memory. Assume that the size and the location of the list are parameters that are passed to the subroutine

via registers, such that

- The parameter *size* is given by the contents of Nios II register *r2*.
- The address of the first entry in the list is given by the contents of register *r3*.

2. Write the main program which initializes the stack pointer, places the required parameters into registers *r2* and *r3*, and then calls the subroutine SORT. The list is loaded into the memory at location LIST_FILE.

3. Compile and download your program.

4. Create a sample list, load it into the memory, and run your program.

Part IV

Modify your program from Part III so that the parameters are passed from the main program to the subroutine via the stack, rather than through registers. Compile, download, and run your program.

Part V

A Nios II processor uses the *ra* register (*r31*) to hold the return address when a subroutine is called. In the case of nested subroutines, where one subroutine calls another, it is necessary to ensure that the original return address is not lost when a new return address is placed into the *ra* register. This can be done by storing the original return address on the stack and then reloading it into the *ra* register upon return from the second subroutine.

To demonstrate the concept of nested subroutines, we will use the computation of the factorial of a given integer n. The factorial of n is determined as

$$n! = n(n-1)(n-2) \cdots \times 2 \times 1$$

It can also be computed recursively as

$$n! = n(n-1)!$$

Note that $0! = 1$.

Write a program that uses recursion to compute the factorial of n. The program has to include a subroutine, called FACTOR, which calls itself repeatedly until the desired factorial has been determined. The main program should pass n as a parameter to the subroutine by placing it on the stack.

Note: Since your subroutine will make use of the multiply instruction, mul, it is essential that in Part I you generated the Nios II/s version of the processor. The economy version, Nios II/e, does not implement the mul instruction.

Compile, download, and run your program. Verify its correctness by trying different values of n.