COL216 Computer Architecture

Lab Assignments 5 : Simulate CPU Design and Prepare for Synthesis

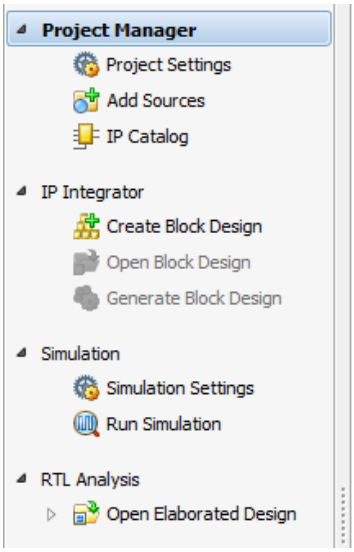
This assignment involves three activities.

- Use memory generator to design program and data memories
- Design a test bench and simulate CPU with memories
- Design display circuit for testing synthesized design on FPGA board

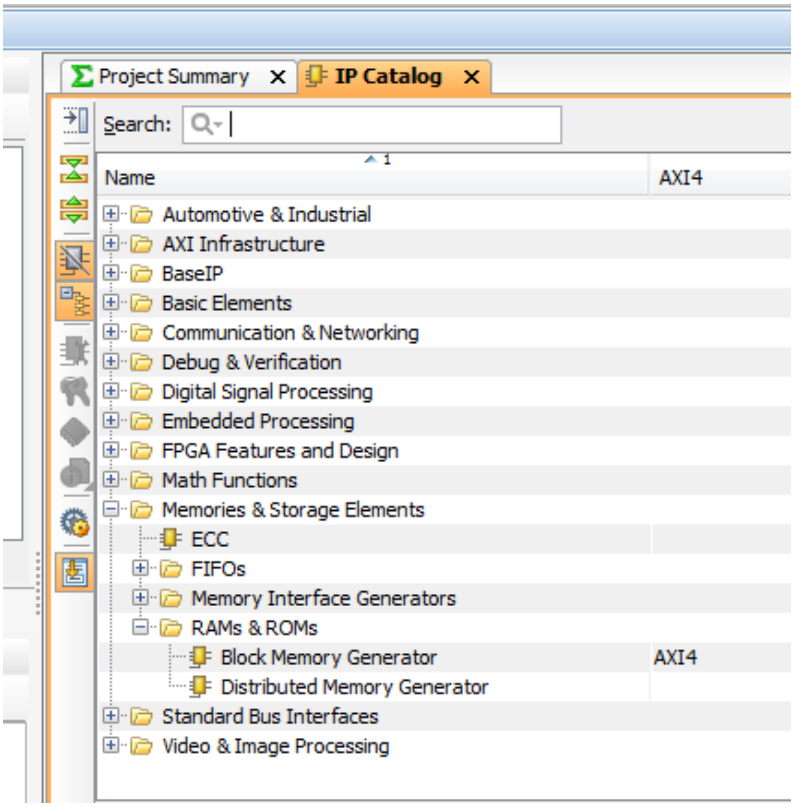
These are described in detail in the following paragraphs.

Memory generation

In Vivado, go to the Project Manager window on the left side of the screen which looks like the screen-shot shown on right. Open IP catalog. You can also open IP Catalog through a drop down menu after clicking on Window in the main menu on the top of the screen. The term IP stands for "Intellectual Property" and here it refers to the pre-designed modules available for use.



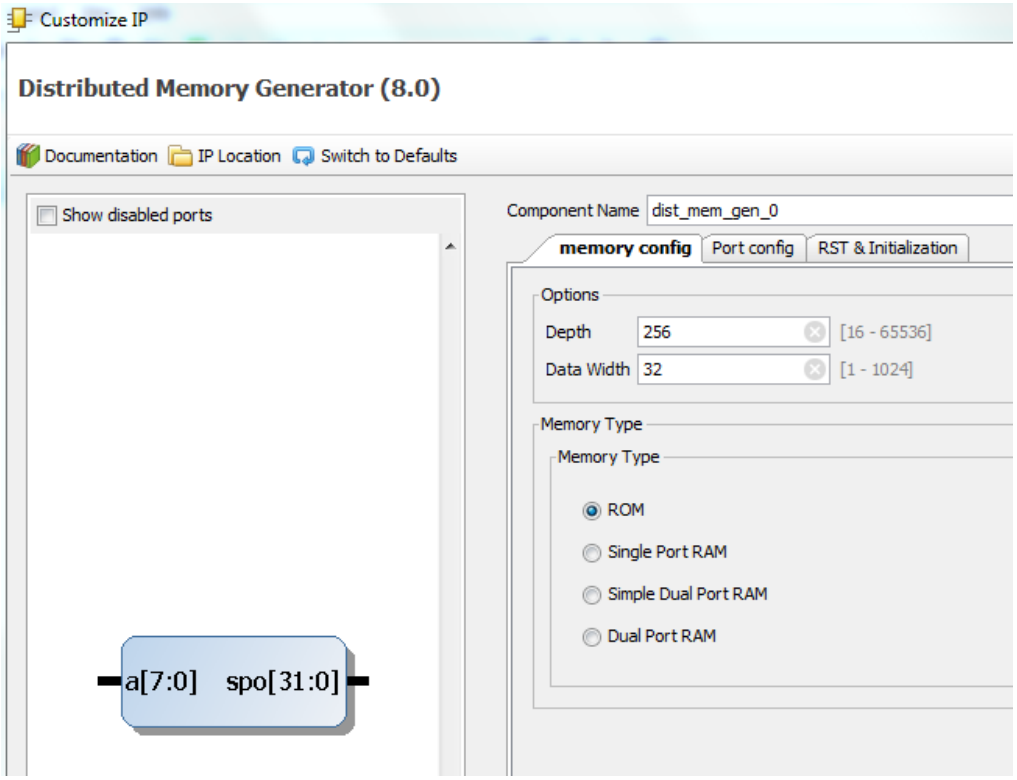
The IP catalog lists various IPs category-wise. Select the category "Memories & Storage Elements" and sub-category "RAMs & ROMs", as shown below.



Now choose “Distributed Memory Generator”, to create a ROM as your program memory and a RAM as your data memory. This generator creates memory modules of specified sizes using LUTs of FPGA. The reason for using the Distributed Memory Generator here is that “Block Memory Generator” creates a memory with an address register, which is not suitable for the present design.

The Distributed Memory Generator opens a window with three tabs.

In the “memory config” tab, specify memory size and type. The screen-shot shown below is for a ROM with 256 words, each of size 32 bits.

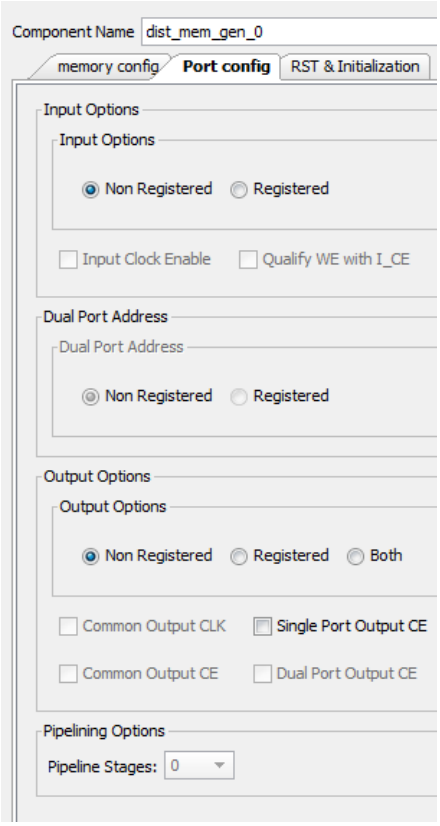


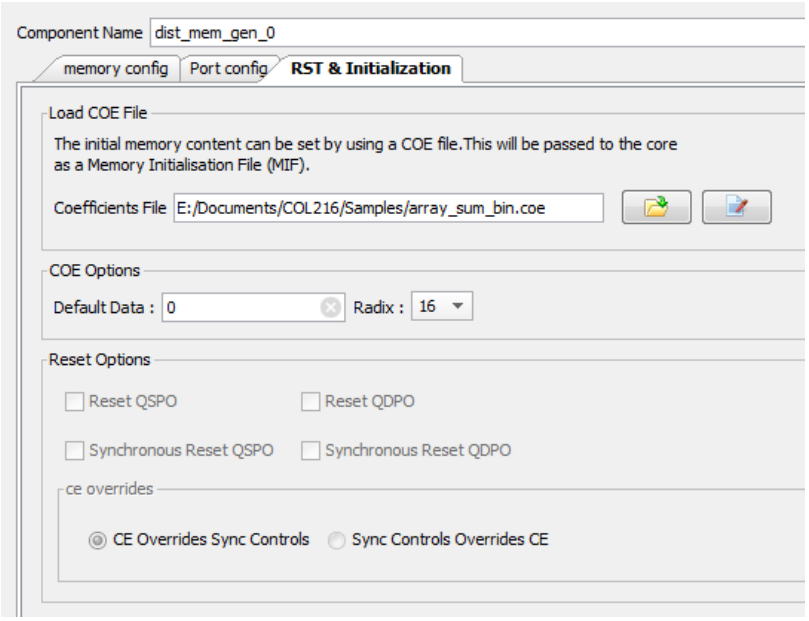
In the “Port config” tab, choose all ports as non-registered. This will probably be the default setting.

In the “RST & Initialization” tab, specify name of the file that defines the ROM contents. A sample file named “array_sum_bin.coe” is available on moodle.

Screen-shots of these two tabs are shown on the next page.

Do a similar exercise for creating a RAM. For this you need not specify initial contents. The generated components will show up among your Design Source files. The default names of these are dist_mem_gen_0, dist_mem_gen_1. But you can specify names of your choice while specifying configuration parameters.





Simulation

For simulation, create a test bench in which the CPU and the two memories are instantiated and interconnected. Remember that in the present design, the CPU generates byte addresses, whereas the memories are word addressable (data width was chosen as 32 while generating the memories). Therefore, the two LSBs of the addresses output by the CPU are to be ignored at present. Further, since the sizes of the memories are much less than the address space size of 2^{32} bytes, several MSBs of the addresses will also get ignored.

The program memory will get initialized with the specified coe file. All that needs to be done for simulation now is to generate clock and reset signals. These could be generated by using wait statements in the test bench or by directly using "Force clock" and "Force constant" in the simulator. Observe the waveforms of various signals as the program execution progresses. Interesting signals to observe are the addresses and data flowing between the CPU and the two memories and the result of DP instructions to the register file.

Design display circuit

This is a preparatory step for synthesis and testing on board. While working with the board, no signals are visible without explicitly making arrangements to display these. The simplest way is to use LEDs to display 16 bits at a time, using slide switches to select what is to be displayed. The display interface circuit is simply a multiplexer (described using a selected signal assignment or a conditional signal assignment), connecting LEDs to various signals to be observed.

One more feature you need to add before going for synthesis is a small FSM to facilitate step-by-step operation so that the signal may be observed on LEDs. This will be described as a part of Assignment 6.

Sample test program

A small test program that uses all the instructions in the instruction subset being implemented has been provided, both in source assembly form (array_sum.s) and in machine code form (array_sum_bin.coe). The machine code was generated using ARMSim#. This program first fills an array with numbers 1 to 10 and then sums these. You will not be able to simulate it on ARMSim# in the present form, because it takes 100 as the starting address of the array. In order to simulate it, the array address may be changed to something like 0x1200.

Try writing your own assembly programs and use these to test your CPU design further. One caution: if you have a load/store instruction with offset part of the address as 0, write the address as [Rn, #0], not as [Rn]. The latter will be interpreted as [Rn], #0 (which is not implemented in the present design) by the assembler.