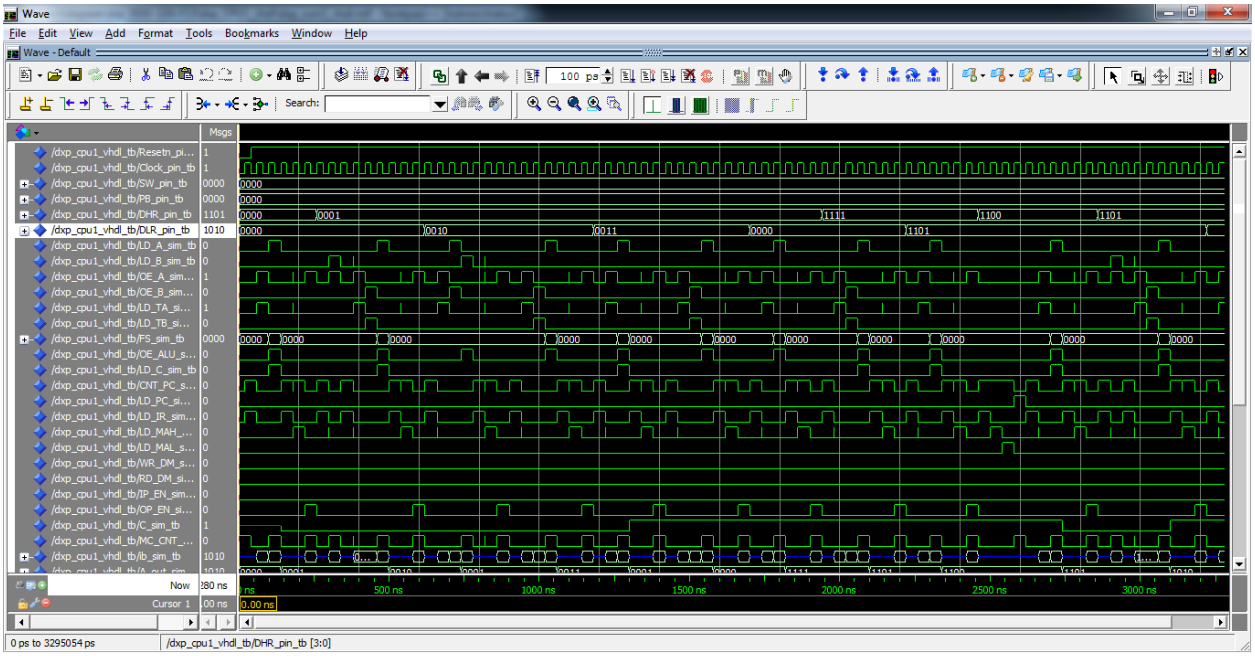
EEEE-220-Lab11

1) From mycourses download on your desktop this pdf file. This lab consists of three parts, which are due as per the syllabus and lab policy.
2) **Objective**: The objective of this lab exercise is to capture and verify the functionality of your fmlRISC Central Processing Unit (CPU) design. You can use either VHDL or Verilog.
3) **Part 1: fmlRISC – CPU – capture and verification with test program 1 (TP1).**
4) Make sure you have all DP and CU related lecture handouts and notes.
5) Create a new project fml_CPU_RISC. Copy/paste using a file manager (Explorer) all files used in the design of your DP and CU. Add all these to your project.
6) For VHDL ONLY: If you used a different name for the package in your DP and CU projects, no need to change anything in either one. If you used the same name, add one of these files to the project and copy/paste the other component declarations. Next, add in one of these the declarations of your DP and CU.
7) The top-level entity should be fml_CPU_RISC. It should contain the structural description of your CPU, which for now consists of just two components: the DP and CU.
8) In principle, you would need only the following ports/pins:
    a. Inputs: Resetn, Clock, SW_in, PB_in
    b. Outputs: DHR_out, DLR_out
9) However, even if both your DP and CU have been verified independently, when put together you might still encounter some bugs, which you have to discover and fix. To avoid time-consuming guessing, I strongly recommend you create an output port for all your CU and DP outputs. I know it is time consuming, but it is the only way you can debug a possible problem.
10) Call your testbench fml_CPU_RISC_TP1 or fml_CPU_TP1.
11) The good news is that your testbench has to run one reset cycle, followed by as many cycles in a loop, for which you only have to control the values of the input pins listed above. In fact, for TP1 you'll only have to control Resetn and Clock.
12) Your testbench doesn't have to assert. You'll assert "manually/visually" in the timing diagram if the values that are output to the displays are the once expected.
13) Only if these are not correct, you'll have to check the other signals. Usually, starting after the cycle that outputs the last good value.
14) TP1 is captured in the file dxp_rom1_vhdl.mif. It is similar to the program that you used to verify your DP and CU. However, I eliminated several "difficult" instructions. If this program works, we consider that the manipulation (ALU), the register-to-register, and the OUT/WR (I/O-P/Memory Mapped) work.
15) Modify it to fit your instruction set and design specifications. Determine the content of register A, which is alternatively displayed on the high and left displays, after each instruction.

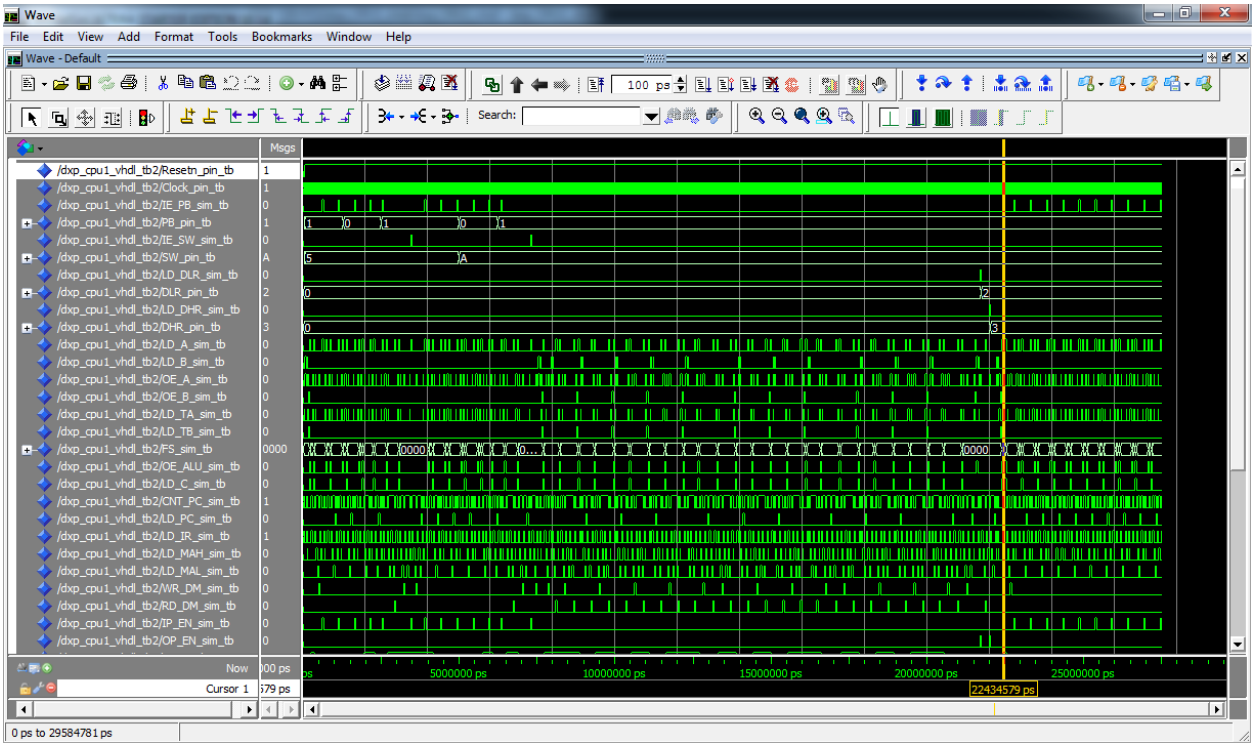
16) Once it simulates correctly, your timing diagram will look as shown below. The first five signals are the relevant ones.

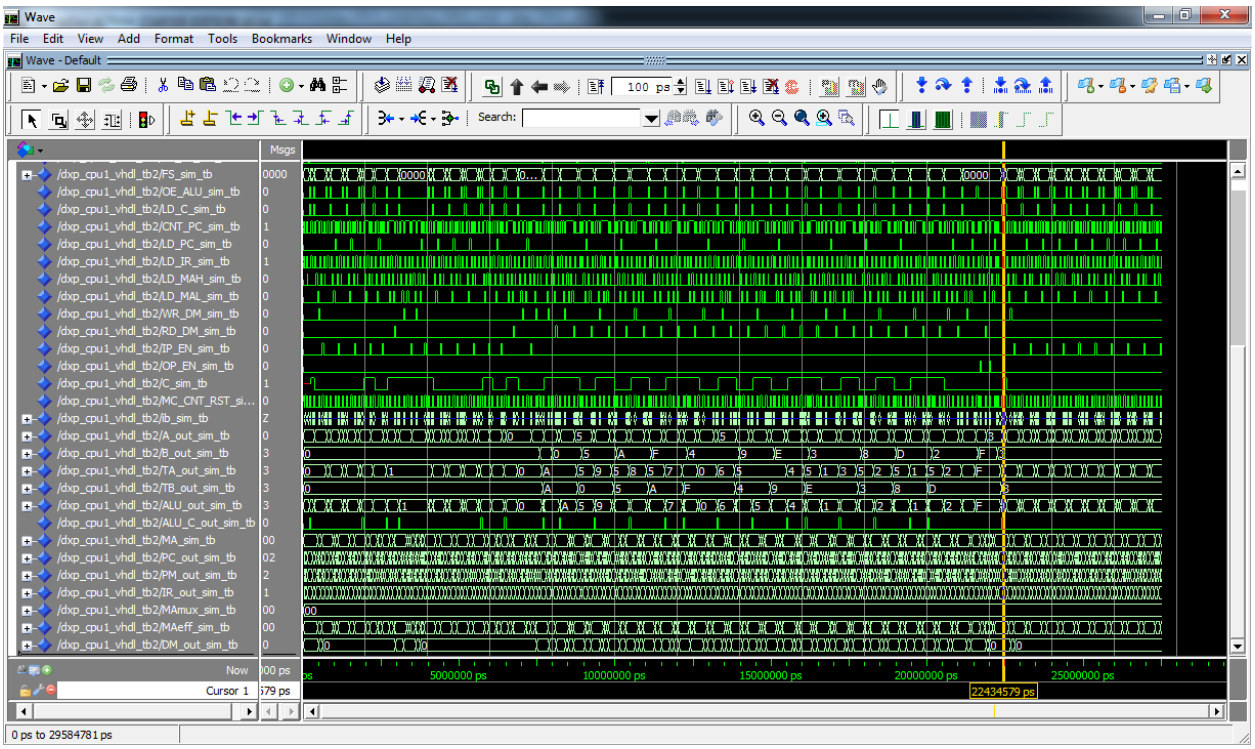


17) Once the simulation is successful, you'll have to run it on the prototyping board. Follow the instructions in the appendix on how to do this.
18) **<u>Part 2</u>: fmlRISC – CPU – verification with test program 2 (TP2).**
19) This is a program that multiplies two 4-bit operands, input by the user via the PB and SW. It uses exclusively the direct addressing mode to store data in RAM. All instruction categories are used.
20) The program is captured in the dxp_rom2_vhdl.mif file. It is thoroughly commented. In addition, we will discuss it during the lecture.
21) To verify this program, you'll need to change the testbench you used for TP1. You'll have to use several for loops or test vectors to mimic the state of the pushbutton and switches. Specifically:
    a. Keep the circuit in reset for at least a clock cycle.
    b. Set SW = "**0101**" and the PB = "**0001**" (PB is depressed). Keep this state for 20-30 clock cycles.
    c. Set SW = "**0101**" and the PB = "**0000**" (PB is pressed). Keep this state for 20-30 clock cycles.
    d. Set SW = "**0101**" and the PB = "**0001**" (PB is depressed). Keep this state for 20-30 clock cycles.

e. The above sequence PB = 1 → 0 → 1 mimics the depressing → pressing → depressing of the PB. This tells the CPU that a new operand value is ready to be read from the switches SW.
f. Continue with the following settings.
g. Set SW = "**0101**" and the PB = "**0001**" (PB is depressed). Keep this state for 20-30 clock cycles.
h. Set SW = "**1010**" and the PB = "**0000**" (PB is pressed). Keep this state for 20-30 clock cycles.
i. Set SW = "**1010**" and the PB = "**0001**" (PB is depressed). Keep this state for 20-30 clock cycles.
j. Now the CPU reads the second operand.
k. To perform the multiplication, allow it to run for another 500 clock cycles with SW = "**1010**" and the PB = "**0001**". With the above operand values the result will be 0x5 * 0xA = 0x32, to be read on DHR and DHL.

22) The simulation results are shown in the two figures below. I output all control and status signals for debugging purposes. The relevant signals are at the top. I recommend you view bus values in hexadecimal. If you want, you can try first smaller operand values, such as 0x3 * 0x6 = 0x12.



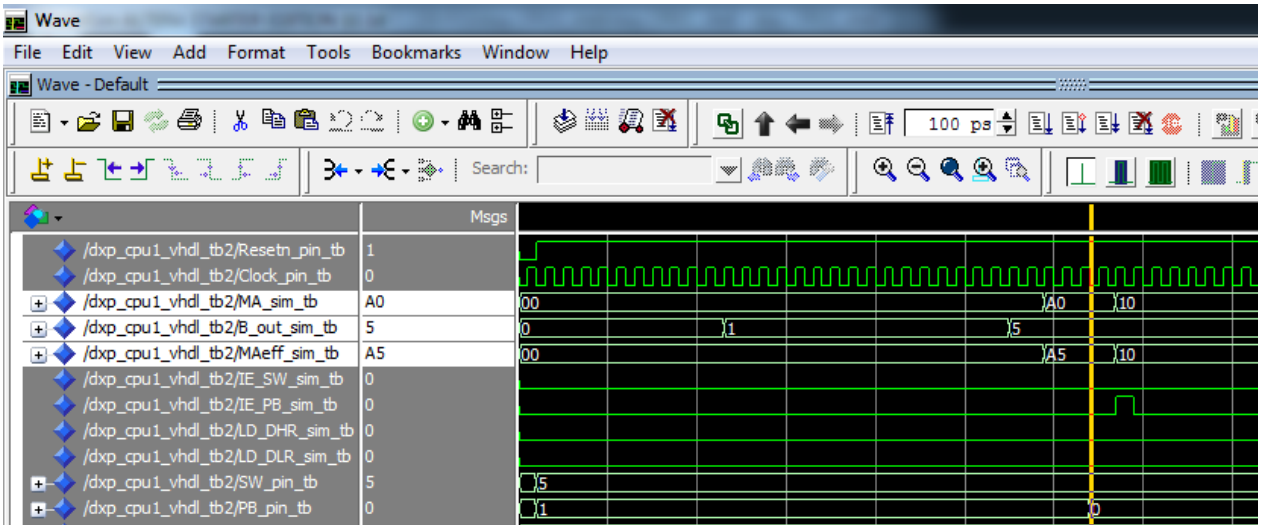


23) **<u>Part 3</u>: fmlRISC – CPU – verification with test program 3 (TP3).**

24) This is the same program as TP2, except that it uses exclusively the indexed/PC_relative addressing mode to store the Read_Count_Variable in RAM.

25) The two templates are provided in the files dxp_rom3_Indexed_vhdl.mif and dxp_rom3_PCrel_vhdl.mif.

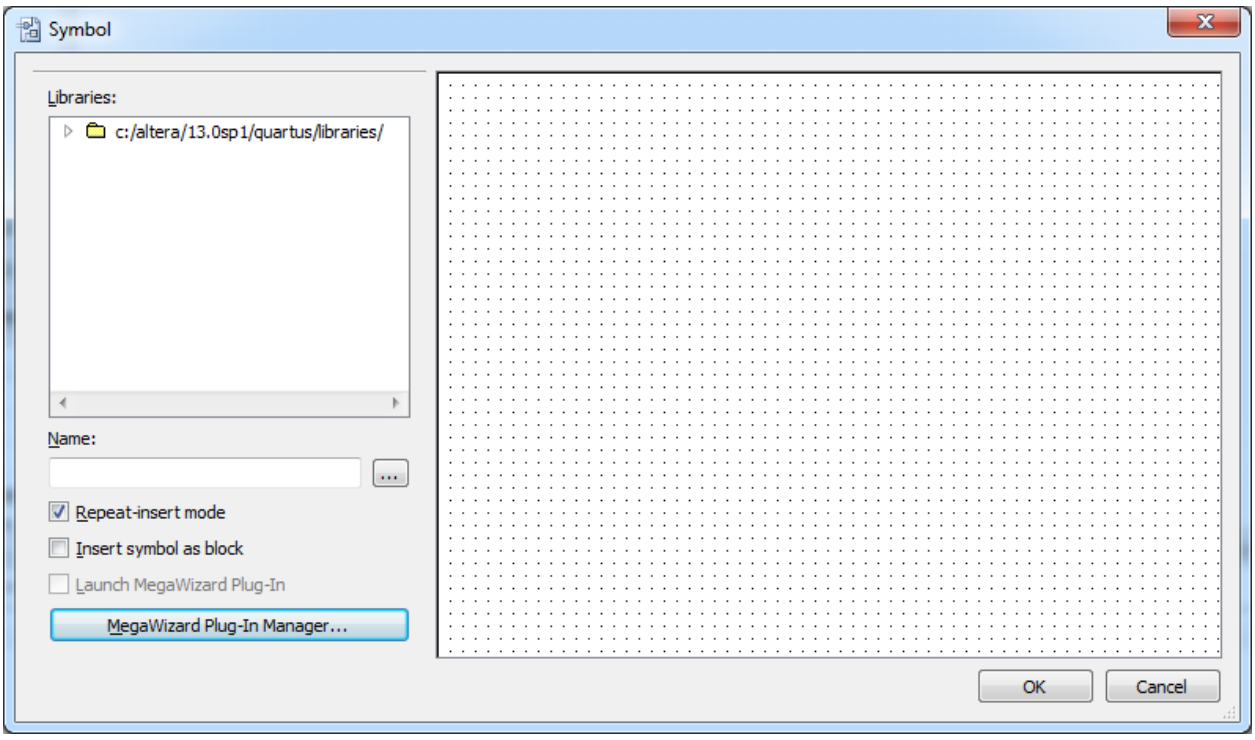
26) You can re-use the same testbench as for TP2. The simulation results are shown below. I emphasize a portion of the run in which the effective address is computed as the sum of MA and regB.

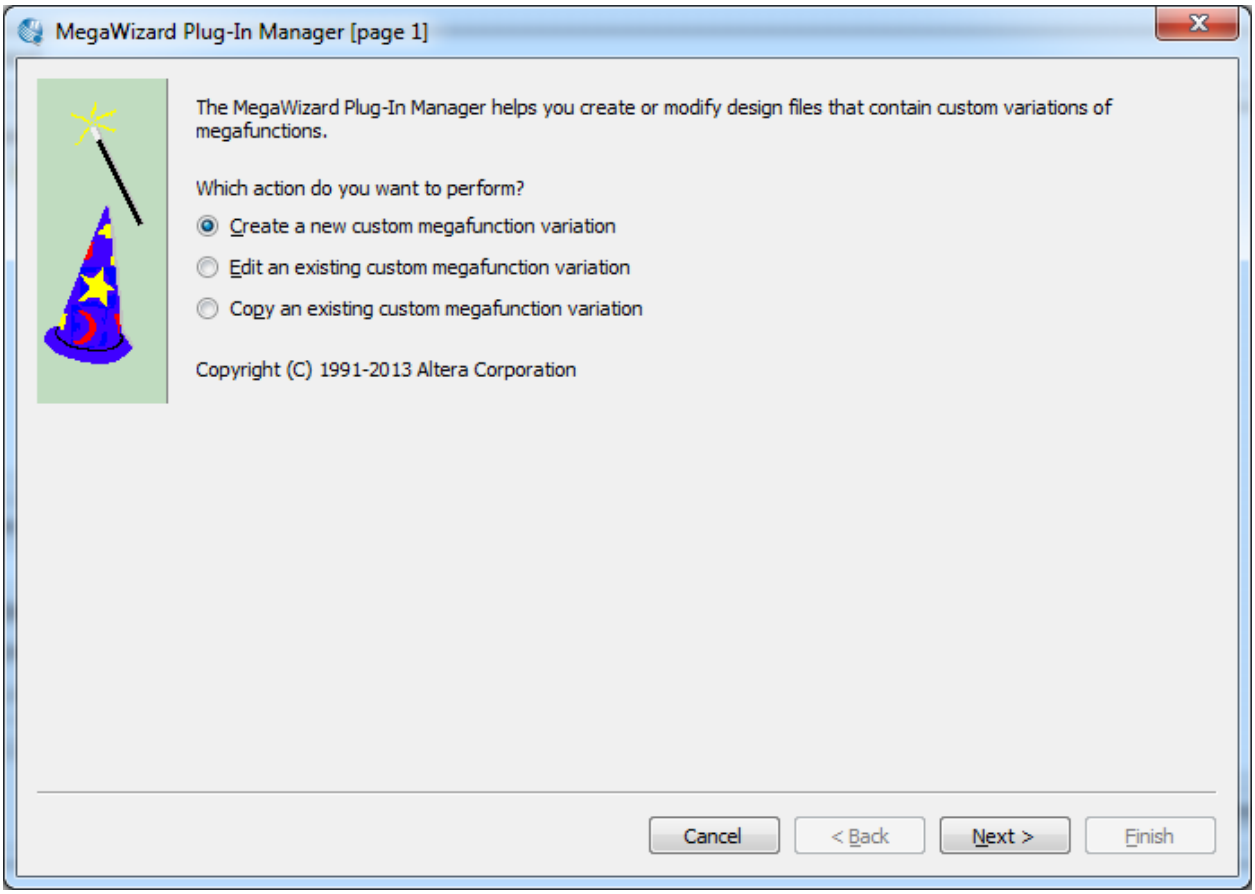


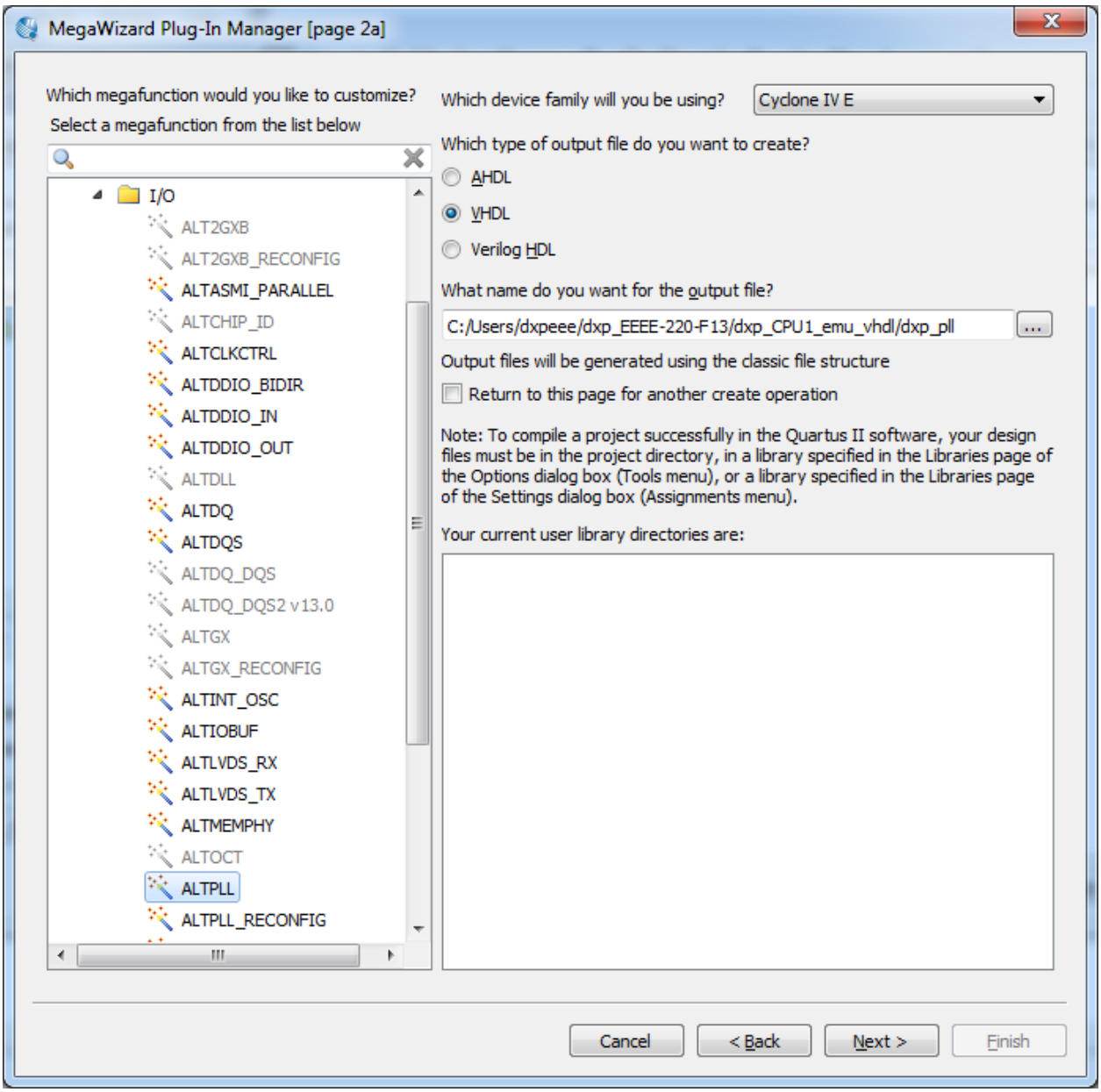
27) **<u>Grading – as per the syllabus and lab policy</u>**:
    a. 20 points for part1 or TP0 – board demo. If it works in simulation, there's no reason for it not to work on the board. Nonetheless, the simulation alone is only 10 points.
    b. 30 points for part2 or TP2 – board demo. The simulation alone is only 15 points.
    c. 20 points for part3 or TP3 – board demo. The simulation alone is only 10 points.
    d. 10 points for the defense of your design during a ~10 min. oral examination with the instructor during the last week of lab 12/04/13 and 12/06/13.
28) The instructor will grade labs 11 & 12. However, I'm sure that your TAs will enjoy seeing what you have accomplished.
29) These two labs require a single report. You can find the details in the lab policy.
30) Even if you don't complete your entire design before the due date and time, you should still upload whatever you have up to that point on mycourses!
31) This concludes this week's lab.

32) <u>APPENDIX</u> – HOW TO RUN EACH OF THESE DESIGNS ON THE DE0-NANO BOARD

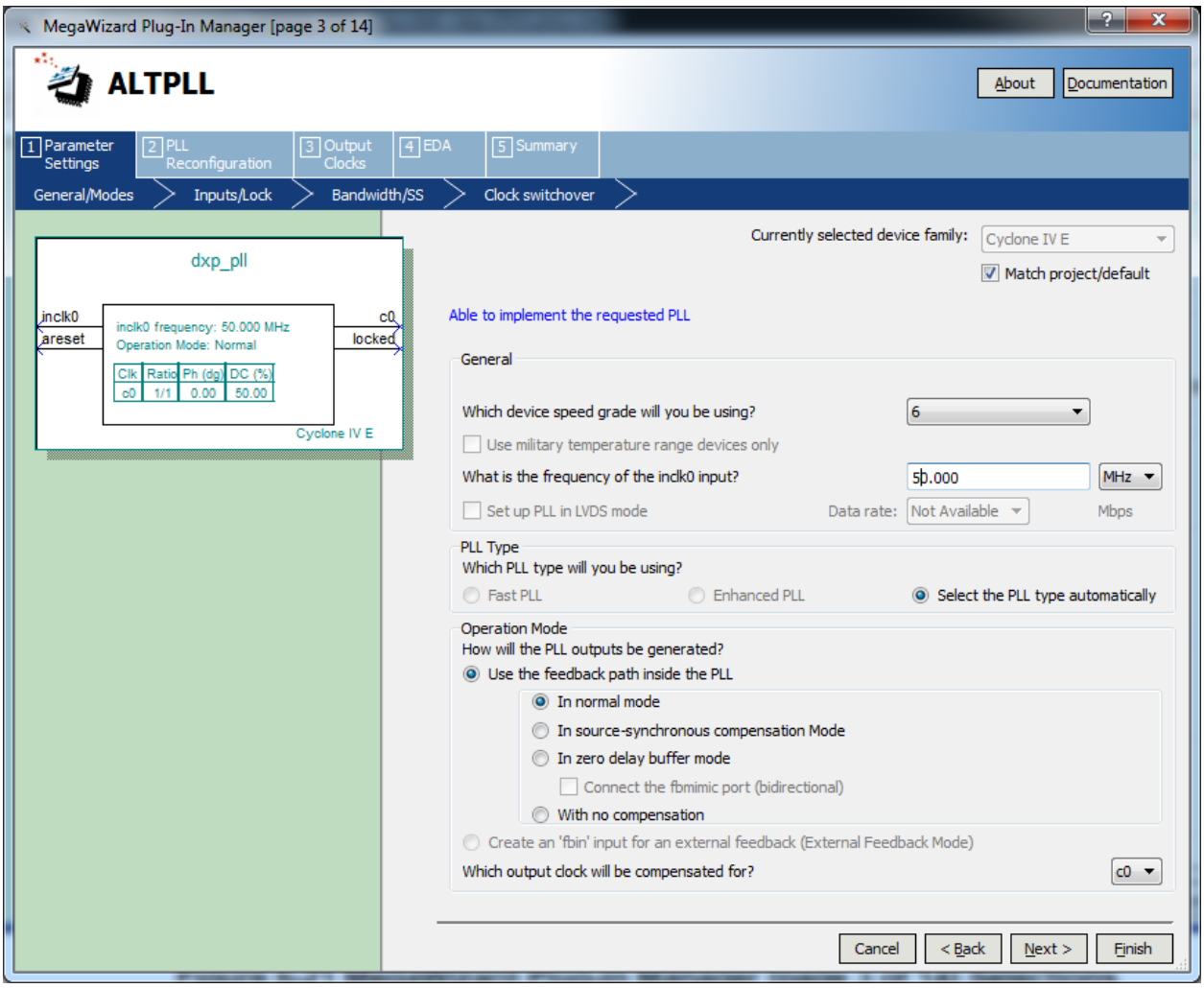
33) In your CPU project create a new block diagram/schematic file.
34) Use the wizard to create a new clock generator block (known as a PLL = Phase-Locked-Loop), as described in the DE0-Nano manual starting on page 54. Below are the sequential screen captures and selections. This clock is useful only for test program 2 and 3. If either of these work, you won't need to demo TP1 on the board. The clock source for TP1 is described later in this appendix. However, you'll still need to go through the next steps.

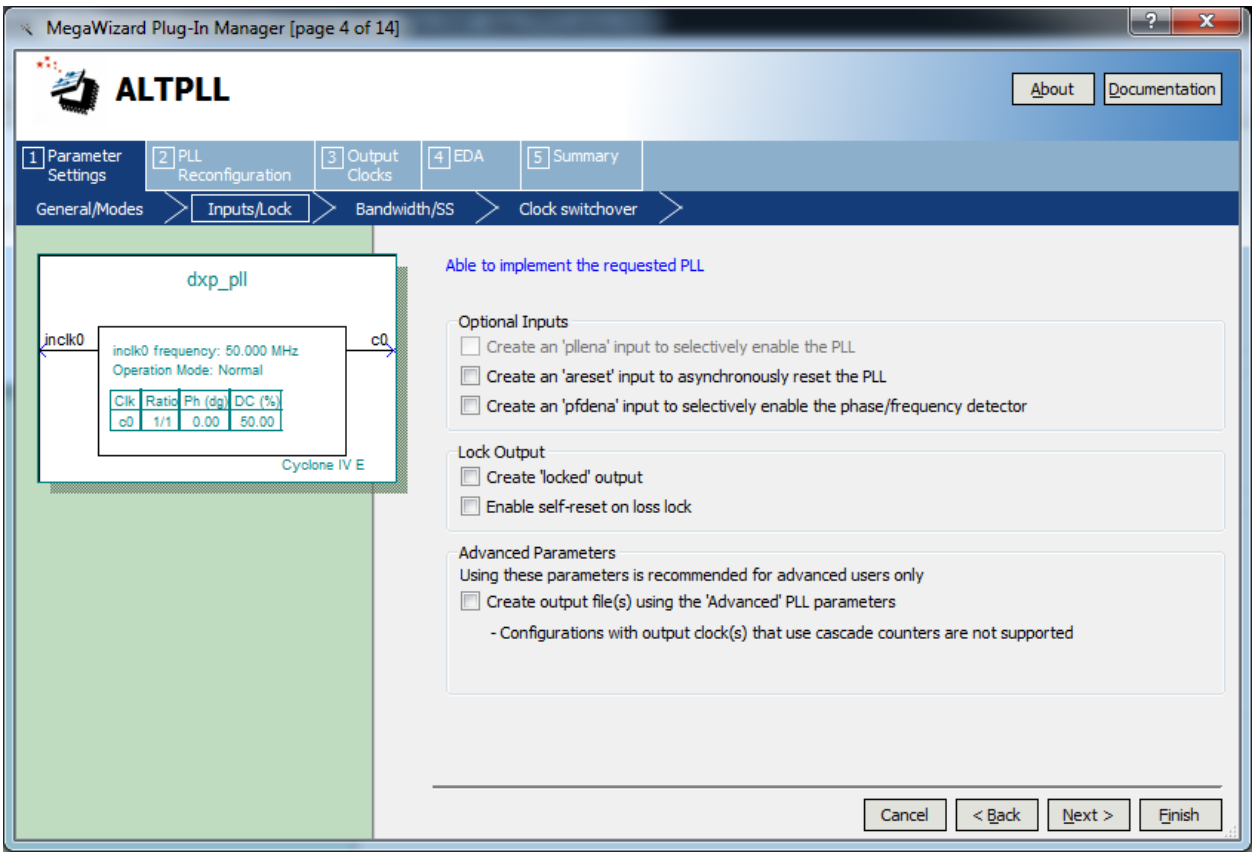

Symbol
Libraries:
c:/altera/13.0sp1/quartus/libraries/
Name:
...
Repeat-insert mode
Insert symbol as block
Launch MegaWizard Plug-In
MegaWizard Plug-In Manager...
OK
Cancel


MegaWizard Plug-In Manager [page 1]
The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions.
Which action do you want to perform?
Create a new custom megafunction variation
Edit an existing custom megafunction variation
Copy an existing custom megafunction variation
Copyright (C) 1991-2013 Altera Corporation
Cancel
< Back
Next >
Finish

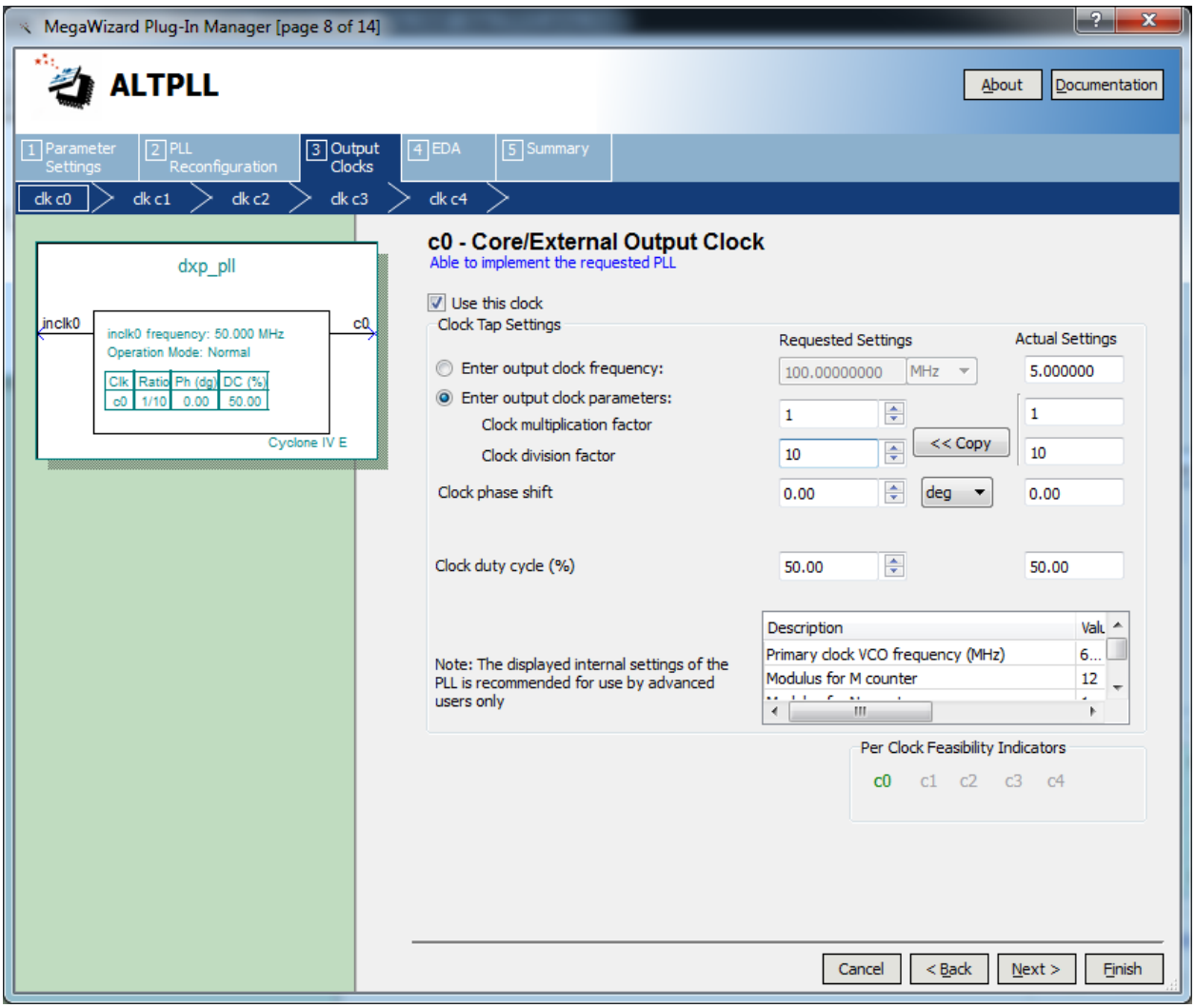


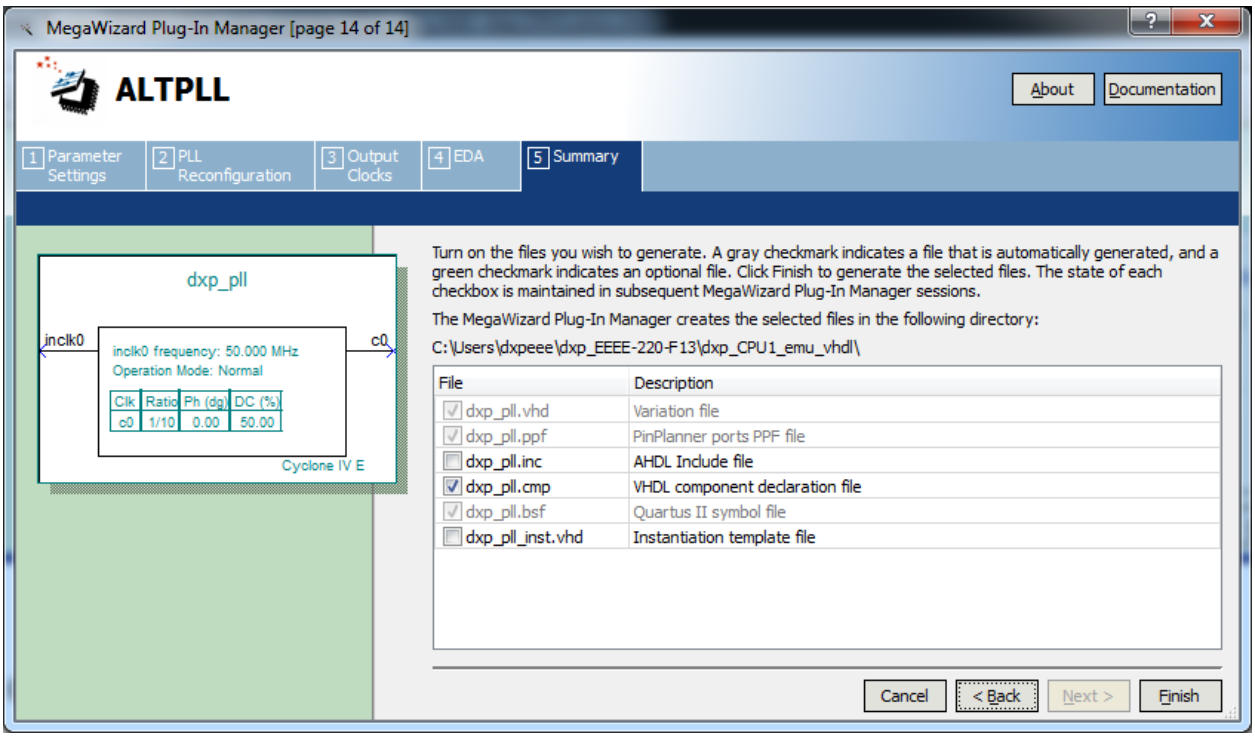
Call this component fml_pll. Change the output file to Verilog if the rest of your design is using that language. Select ALTPLL.

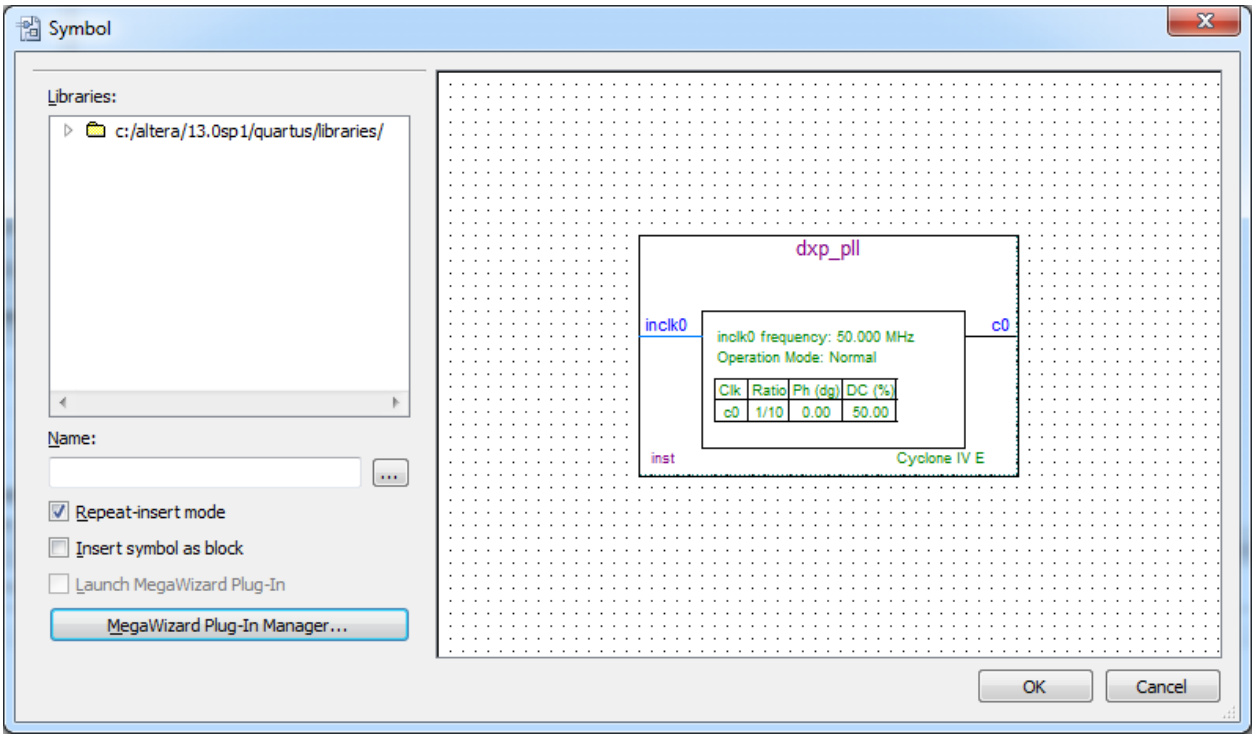

MegaWizard Plug-In Manager [page 3 of 14]
ALTPLL
About
Documentation
1 Parameter Settings
2 PLL Reconfiguration
3 Output Clocks
4 EDA
5 Summary
General/Modes
Inputs/Lock
Bandwidth/SS
Clock switchover
dxp_pll
inclk0
areset
c0
locked
inclk0 frequency: 50.000 MHz
Operation Mode: Normal
Clk
Ratio
Ph (dg)
DC (%)
c0
1/1
0.00
50.00
Cyclone IV E
Currently selected device family:
Cyclone IV E
Match project/default
Able to implement the requested PLL
General
Which device speed grade will you be using?
6
Use military temperature range devices only
What is the frequency of the inclk0 input?
50.000
MHz
Set up PLL in LVDS mode
Data rate:
Not Available
Mbps
PLL Type
Which PLL type will you be using?
Fast PLL
Enhanced PLL
Select the PLL type automatically
Operation Mode
How will the PLL outputs be generated?
Use the feedback path inside the PLL
In normal mode
In source-synchronous compensation Mode
In zero delay buffer mode
Connect the fbmimic port (bidirectional)
With no compensation
Create an 'fbin' input for an external feedback (External Feedback Mode)
Which output clock will be compensated for?
c0
Cancel
< Back
Next >
Finish



Click next four times.


MegaWizard Plug-In Manager [page 8 of 14]
ALTPLL
About
Documentation
1 Parameter Settings
2 PLL Reconfiguration
3 Output Clocks
4 EDA
5 Summary
clk c0
clk c1
clk c2
clk c3
clk c4
dxp_pll
inclk0
c0
inclk0 frequency: 50.000 MHz
Operation Mode: Normal
Clk
Ratio
Ph (dg)
DC (%)
c0
1/10
0.00
50.00
Cyclone IV E
c0 - Core/External Output Clock
Able to implement the requested PLL
Use this clock
Clock Tap Settings
Requested Settings
Actual Settings
Enter output clock frequency:
100.00000000
MHz
5.000000
Enter output clock parameters:
Clock multiplication factor
1
1
Clock division factor
10
<< Copy
10
Clock phase shift
0.00
deg
0.00
Clock duty cycle (%)
50.00
50.00
Note: The displayed internal settings of the PLL is recommended for use by advanced users only
Description
Value
Primary clock VCO frequency (MHz)
6...
Modulus for M counter
12
Per Clock Feasibility Indicators
c0
c1
c2
c3
c4
Cancel
< Back
Next >
Finish

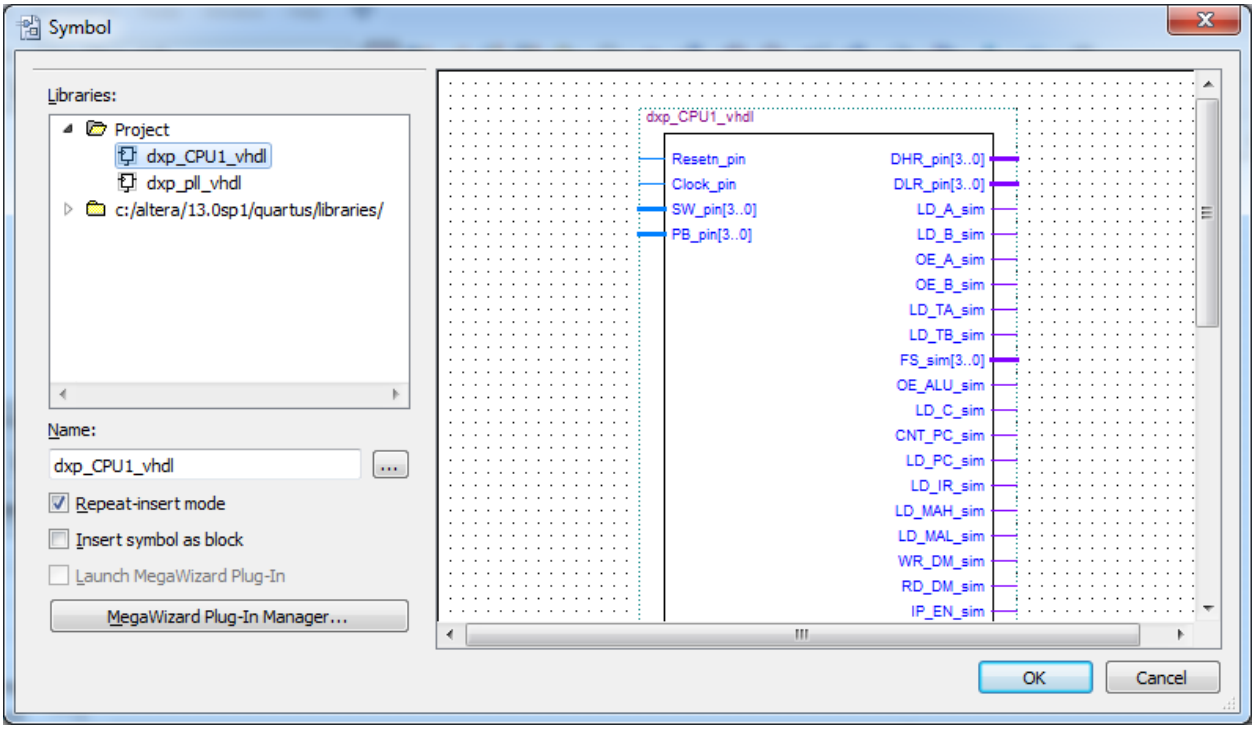




Add this symbol to your schematic.

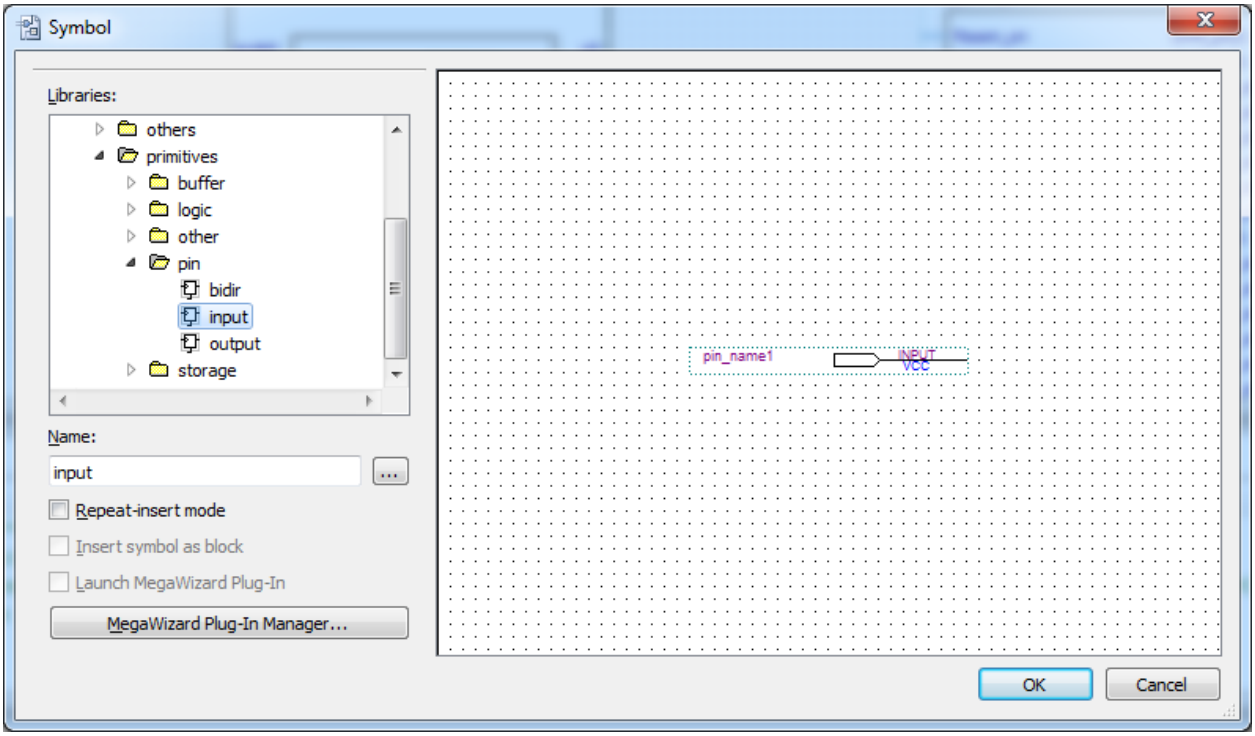
35) Now, right-click on your top-level CPU file and ask to Create a Symbol for it. At this point you may be asked to save the schematic. Do so and name it fml_CPU_emu.bdf.

36) Once the symbol has been created, return to the schematic and instantiate it. You'll find it under the Project symbols/components.

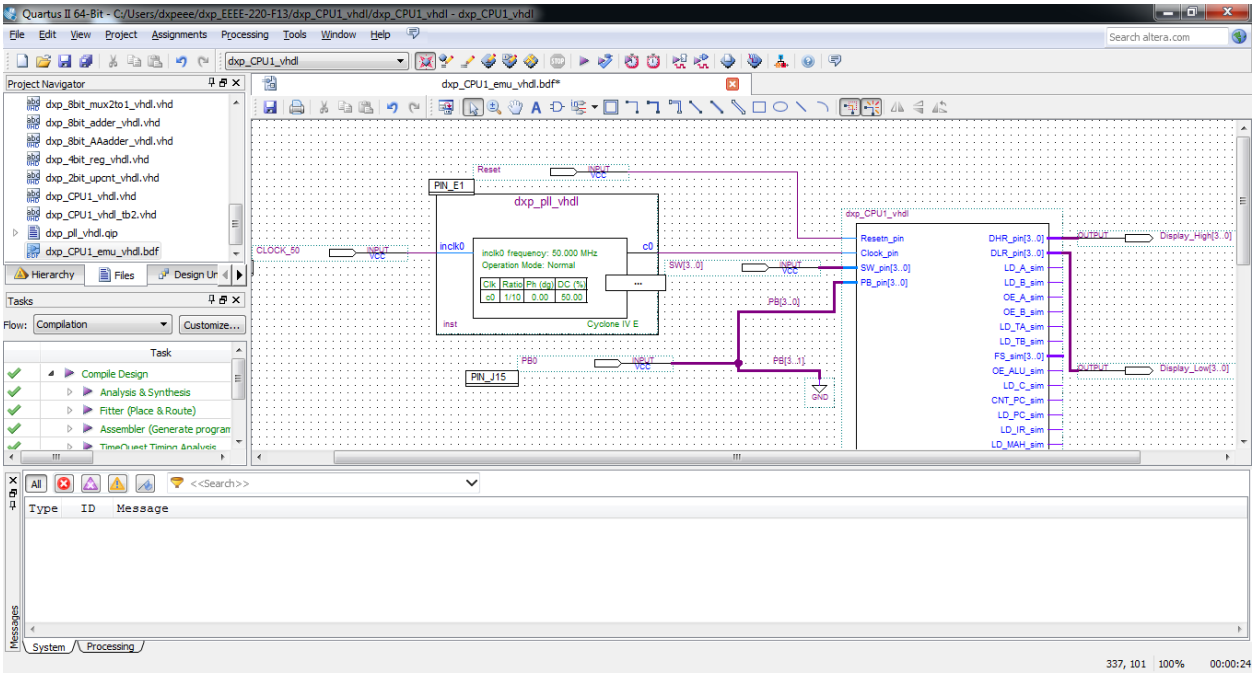


37) Connect c0 to your CPU clock input.

38) Next, you'll add the necessary pins to the schematic.

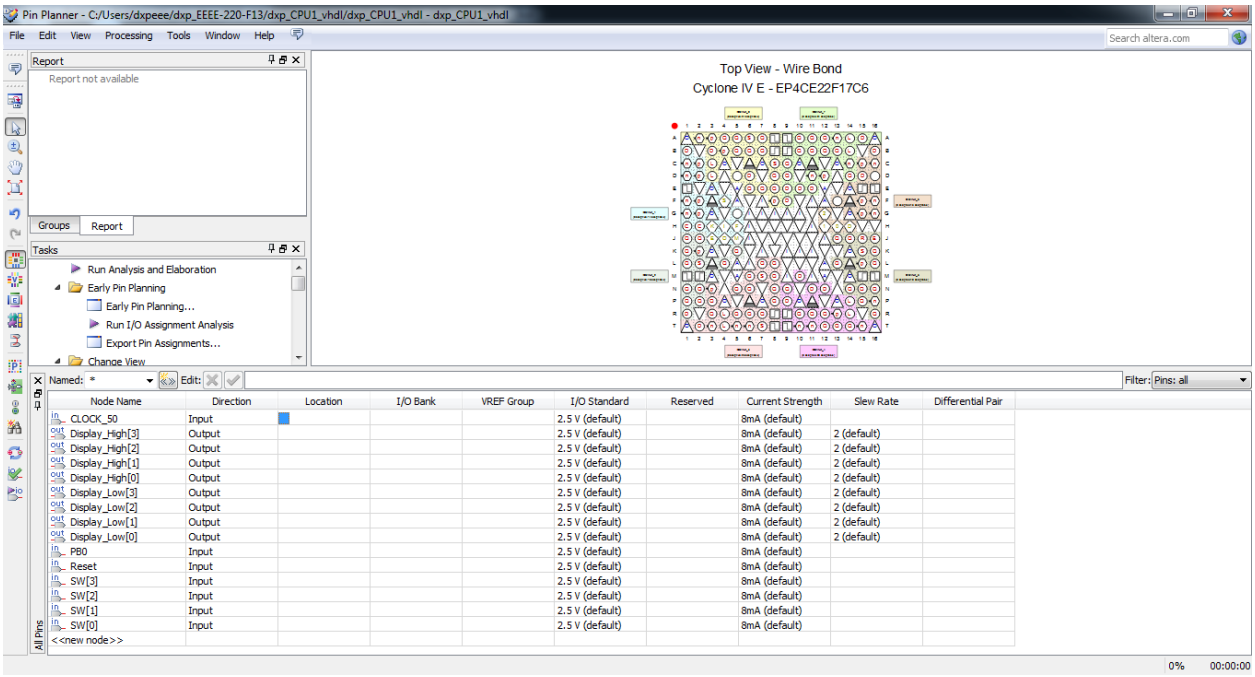


39) After adding input and output pins, your schematic should look like this (without the pin assignments yet):

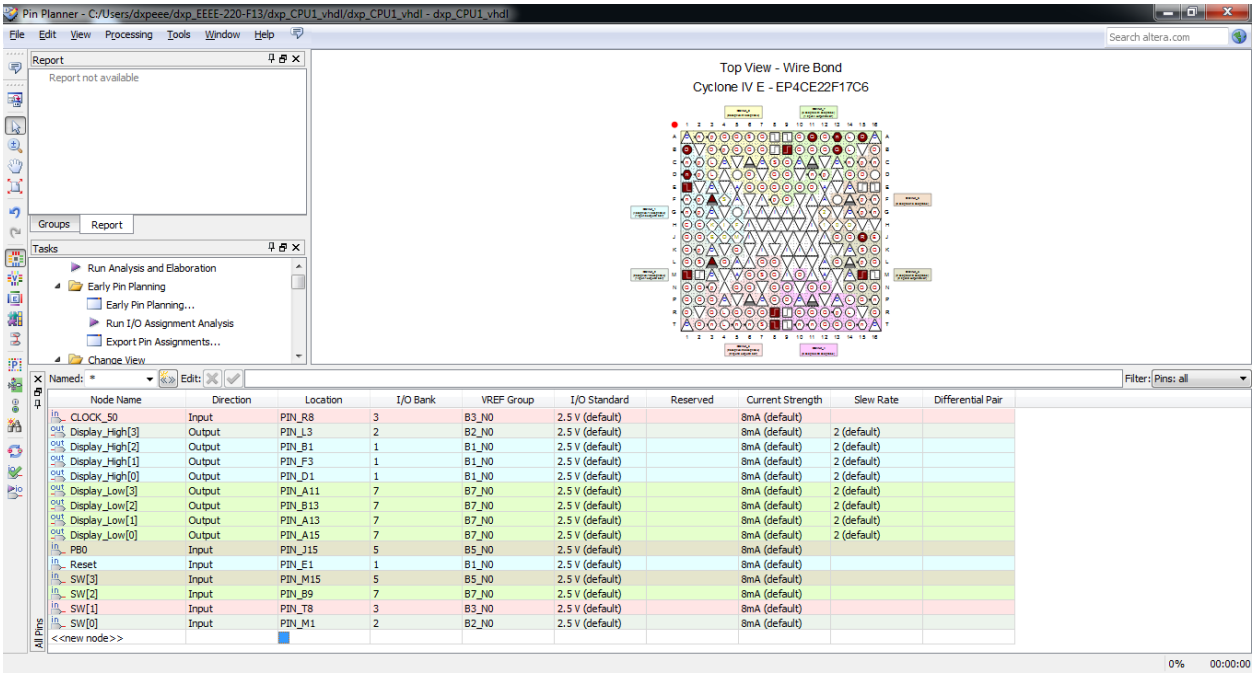


40) In preparation for pin assignments, right-click on the *.bdf file and set it as top-level entity. Then execute **Processing > Start > Start Analysis & Elaboration**.

41) Now, you'll make the necessary pin assignments. Select **Assignments > Pin Planner**.



42) The push-buttons PB0 = KEY[0] and Reset = KEY[1] are normally (depressed) high. All assignments are as follows (pages 11-14 of the DE0-Nano manual):



43) Timing settings are critically important for a successful high-speed design. In our case, the clock frequency is low enough to accommodate the propagation delay through the most critical path. Nonetheless, to avoid warnings from Quartus, we'll create a bare-bone Synopsys Design Constraints File (.sdc) that the Quartus II TimeQuest Timing Analyzer uses during design compilation. For more complex designs, you will need to consider the timing requirements more carefully.

44) To create an SDC, perform the following steps:

a. Open the TimeQuest Timing Analyzer by choosing Tools > TimeQuest Timing Analyzer.

b. Select File > New SDC file. The SDC editor opens.

c. Type the following code into the editor:

```
create_clock -period 20.000 -name CLOCK_50

derive_pll_clocks

derive_clock_uncertainty
```

d. Save this file as fml_CPU_emu.sdc

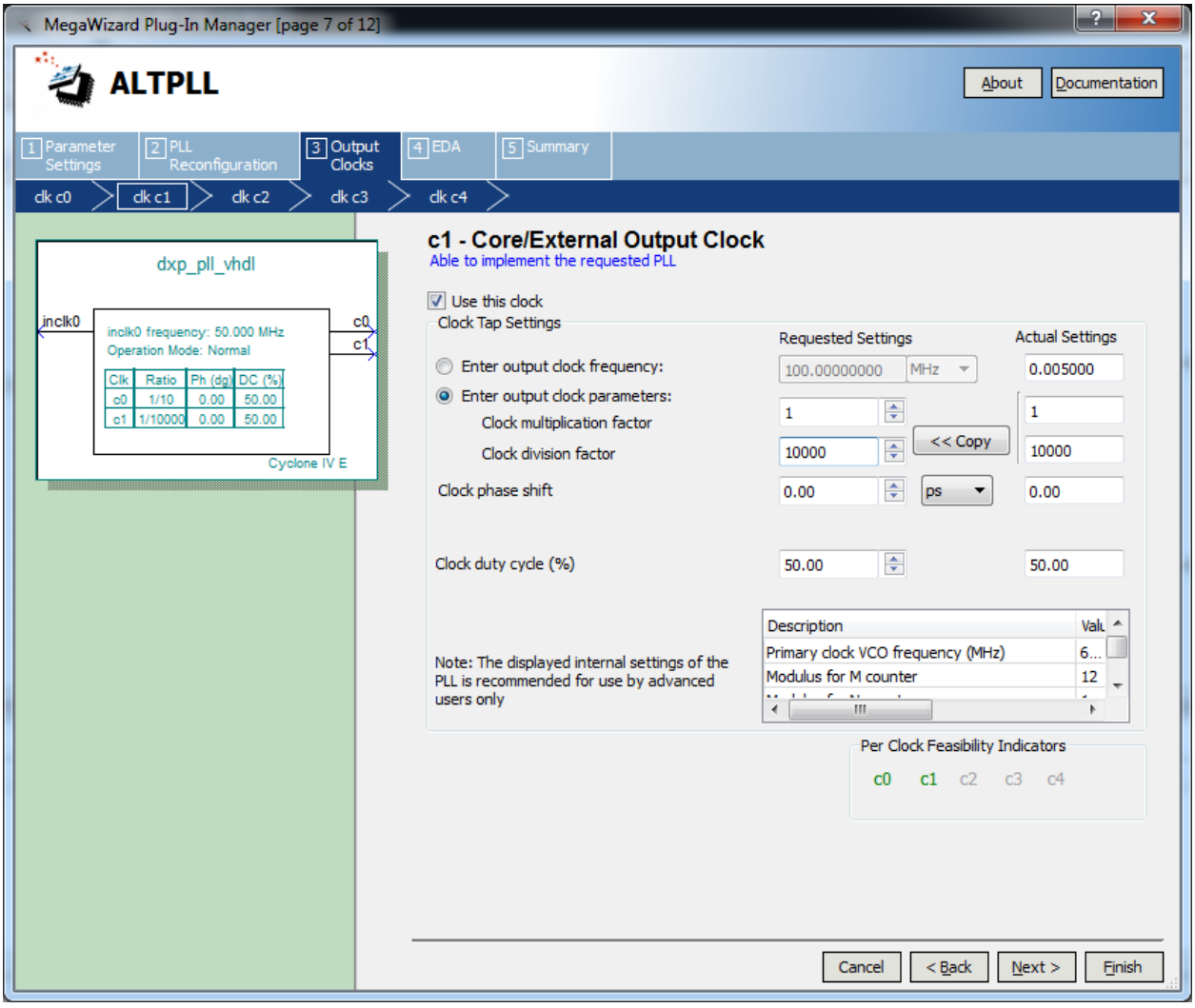
45) Naming the SDC with the same name as the top-level file causes the Quartus II software to use this timing analysis file automatically by default. If you used another name, you would need to add the SDC to the Quartus II assignments file.

46) Now you can fully compile and then program the FPGA with your design.

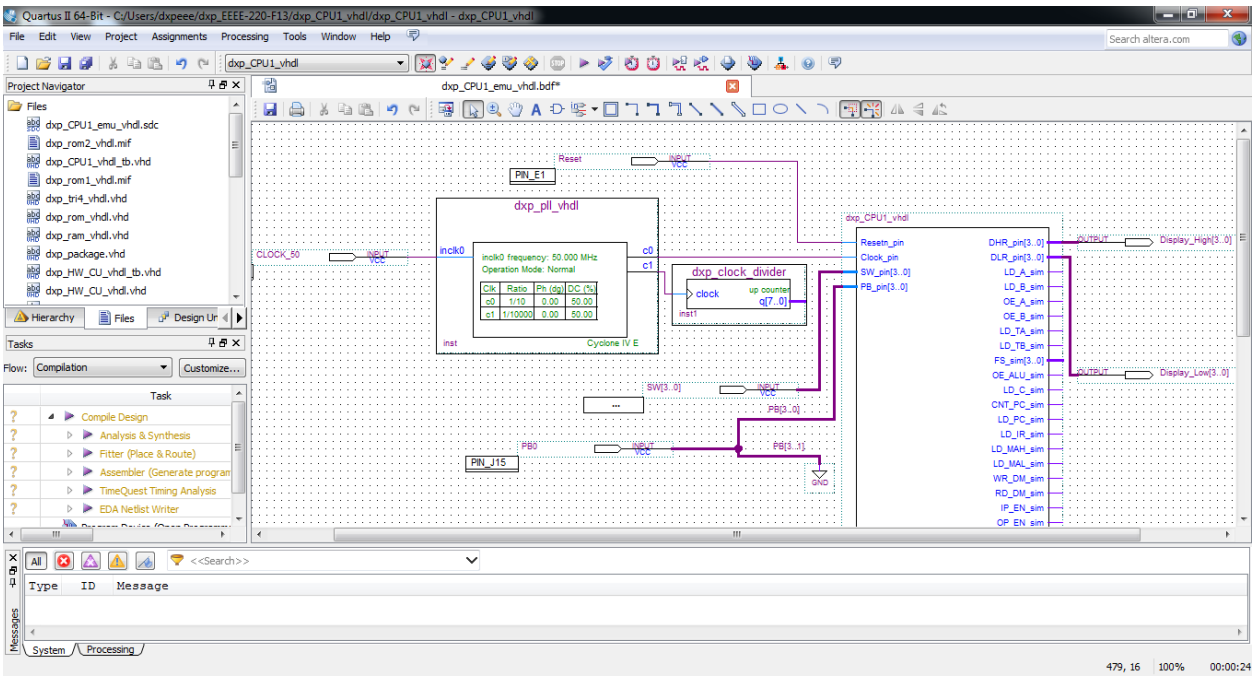
47) To compile press on the magenta play button. Unlike in the last several labs, we are performing a full compilation, i.e. ultimately generating a FPGA programming file.

48) To program, select Tools > Programmer. If you connect your board, the programmer tool will automatically recognize it. Click program.

49) TP1 will run automatically. However, due to the high speed of the clock, you'll see the LEDs on at all times. To be able to see the changes in the displayed values, you may need to divide the clock first by the maximum allowable value 10,000. Double-click the fml_pll component and choose to output a second clock as shown below:



50) This will bring the clock frequency down to 5 KHz. If it is still too fast, create an 8-bit plain binary counter using the wizard. Use lpm_counter. Take the clock from Q5 or Q6 output.



51) Alternatively, you can connect KEY[0] to your clock input. This gives you the opportunity to clock your design manually, clock cycle by clock cycle.
52) You'll need to re-compile the design for the different test programs, and manually switch the clock source in the schematic.
53) For TP1 and TP2 use the high-speed clock (50 MHz).
54) Set some arbitrary value on the switches and then press and depress KEY[0]. Do the same for the second operand. The two operands can also be equal. Please be gentle with the switches, as these are very small and may break easily.