EEEE-521-621-Lab11

1) From mycourses download on your desktop this pdf file.
2) The goal of this lab is to augment the I/O-P unit with the necessary tri-states and registers to connect to the DE0-Nano board switches, push-button, and 8 LEDs.
3) **<u>If your pipelined processor and/or cache memory are not working at this time, you can use your non-pipelined processor and/or "monolithic" memory block with the penalties shown below.</u>**
4) **<u>Part 1:</u>** The input peripheral you will read from consists of:
    a. The PB value; this is input in the LSbit position; a logic '1' value means the PB is depressed; a logic '0' value means the PB is pressed;
    b. The four switches; these are input in the next bit positions;
    c. And the remaining bit positions; these are tied to GND or logic 0.
5) The output peripheral consists of eight LEDs.
6) Assign unique addresses to the IP and OP.
7) Add the necessary tri-states and register to your I/O-P unit.
8) Prepare your processor for instantiation in the FPGA, as described further in the appendix.
9) **<u>Part 2:</u>** Write a sequence of code that will enable the user to enter 16 4-bit values one after the other. Here's a suggested sequence:
    a. The user selects a new or the next 4-bit value on the switches.
    b. The CPU should check continuously for a depressed – pressed – depressed PB. When it finds the PB pressed, it means that the value on the switches is a new value, and stores it in an array. To check the state of the PB:
        i. Input the IP value.
        ii. Rotate right through carry. Now the value of the carry holds the value of the PB.
    c. If the value of the PB was '1' and it is now '0', then the value set by the switches is valid. Save it to memory.
    d. If the value of the PB was '1' and it is still '1', then the value is not valid.
    e. If the value of the PB was '0', check again until it becomes '1'.
10) Demonstrate your working code by running it on the board, i.e. enter 16 different values and than display each value on the LEDs. The significance of the LEDs is as follows:
    a. LED ON = logic '1';
    b. LED OFF = logic '0'.
11) **<u>Part 3:</u>**
    a. **521** – Write a sequence of code that would add two 4 x 4 matrices, and then transpose the result matrix. The values of the elements of these two matrices are entered as described in part 2. The eight LSbits of the value of each element of the result matrix are displayed for ~1 second on the LEDs.
    b. **621** – Write a sequence of code that would multiply two 4 x 4 matrices. The values of the elements of these two matrices are entered as described in part 2. The eight LSbits of the value of each element of the result matrix are displayed for ~1 second on the LEDs.

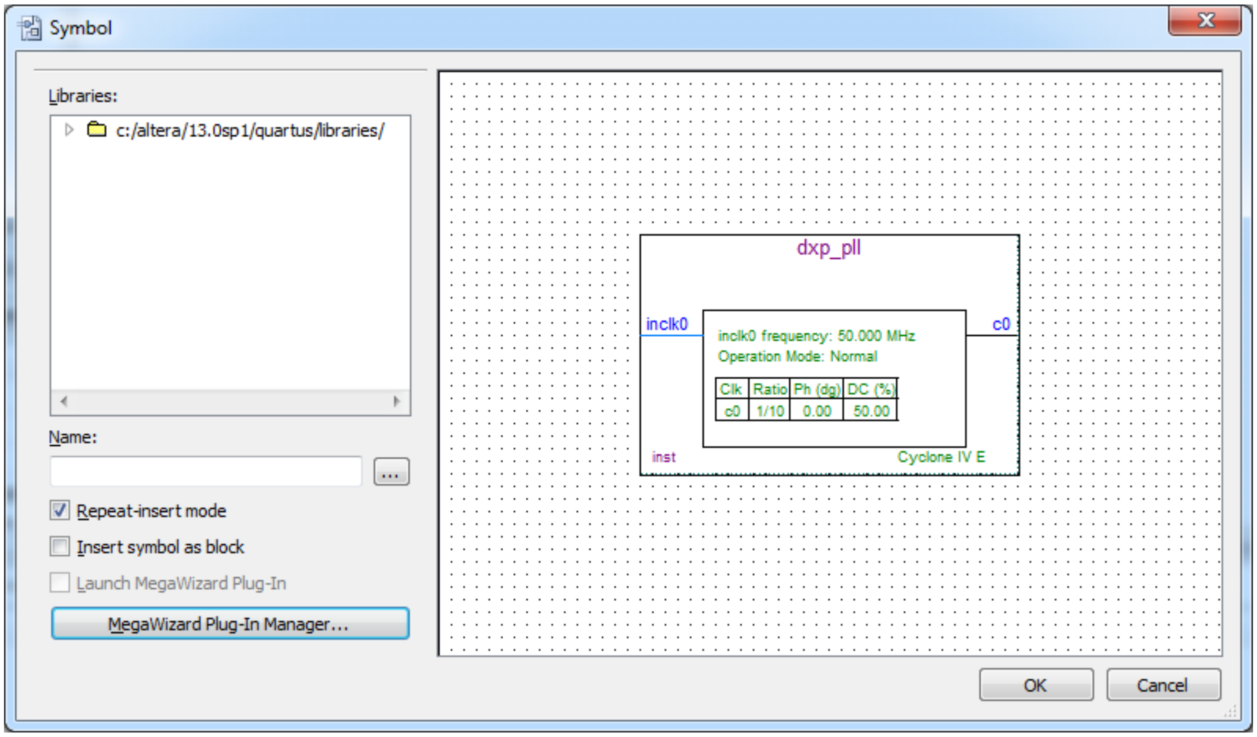
12) **Part 4:** How many clock cycles does the matrix multiplication take?
13) Because the code and data set size in this case are known, you could read this information from the simulation or calculate it analytically. However, in real large benchmarks these methods are not practical.
14) Instead, processors have built-in “performance counters”.
15) Create such a “counter” within the behavioral control block. Alternatively, you can instantiate it external to the CPU and use an OP register to store its control signal values. Read its current value as an IP.
16) Your code should initialize it to zero and enable it to count up just before the matrix multiplication starts.
17) At the end of the matrix multiplication: stop counting, read, and display its value.
18) Record this value in your lab report.
19) If time permits, show your demonstration to your TA. **Note**: only the TA will assign a grade.
20) Archive your project.
21) Write your report and upload it along with your archived project(s) in the dropbox on mycourses, as described in the lab policy.
22) This concludes this week’s lab.
23) **Grading**:
    a. 10 points for each part.
    b. -5p if not using the pipelined processor.
    c. -5p if not using the cache memory.

24) **<u>APPENDIX</u> – HOW TO INSTANTIATE YOUR DESIGN ON THE DE0-NANO BOARD AND RUN YOUR PROGRAMS**

25) You should create a top level module/entity that contains the necessary PLL and your CPU. The PLL should generate all necessary clocks for your system. The CPU should contain your memory, monolithic or cache. From here you can skip to point ___.

26) Alternatively, you can follow the steps below to create a schematic at the very top level. This is not uncommon. This still requires you to create a PLL and CPU as described above, and the associated symbols.

27) In your CPU project create a new block diagram/schematic file. Add the symbol of your PLL to your schematic.



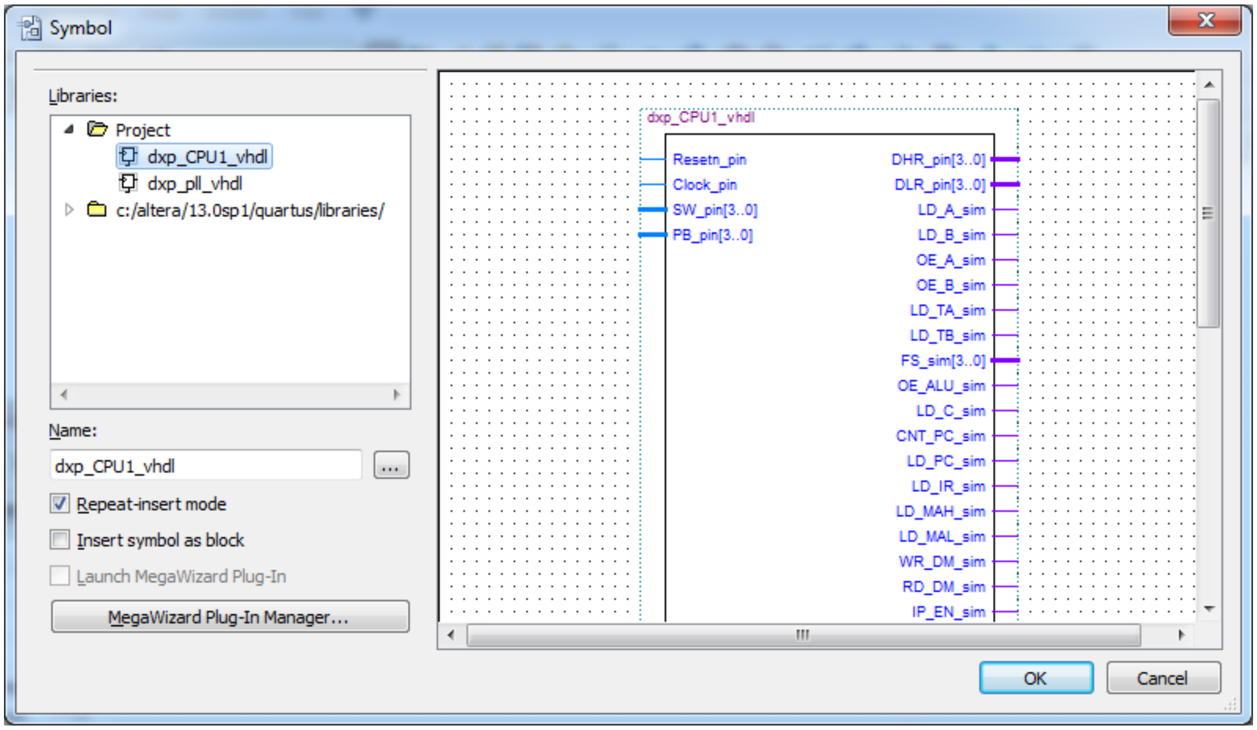
[✓] Repeat-insert mode
[ ] Insert symbol as block
[ ] Launch MegaWizard Plug-In

inclk0 frequency: 50.000 MHz
Operation Mode: Normal

Clk	Ratio	Ph (dg)	DC (%)
c0	1/10	0.00	50.00

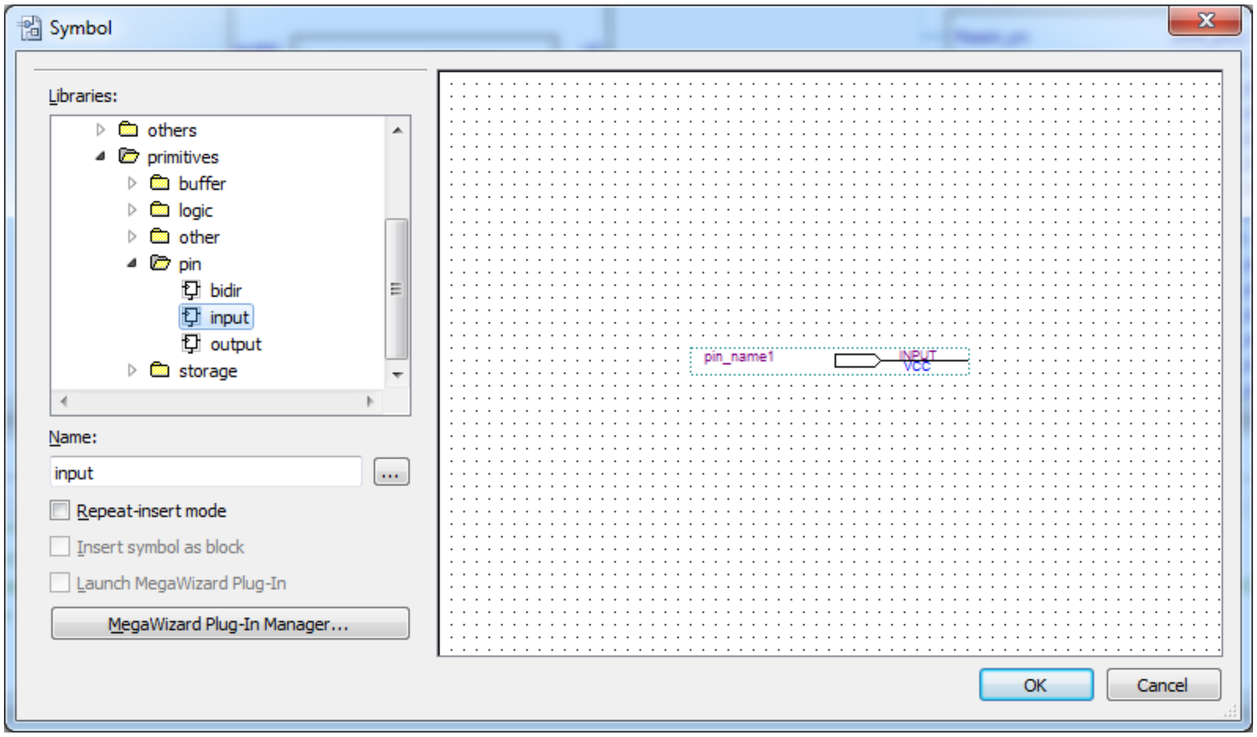
28) Now, right-click on your top-level CPU file and ask to "Create a Symbol" for it. At this point you may be asked to save the schematic. Do so and name it fml_CPU_emu.bdf.

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

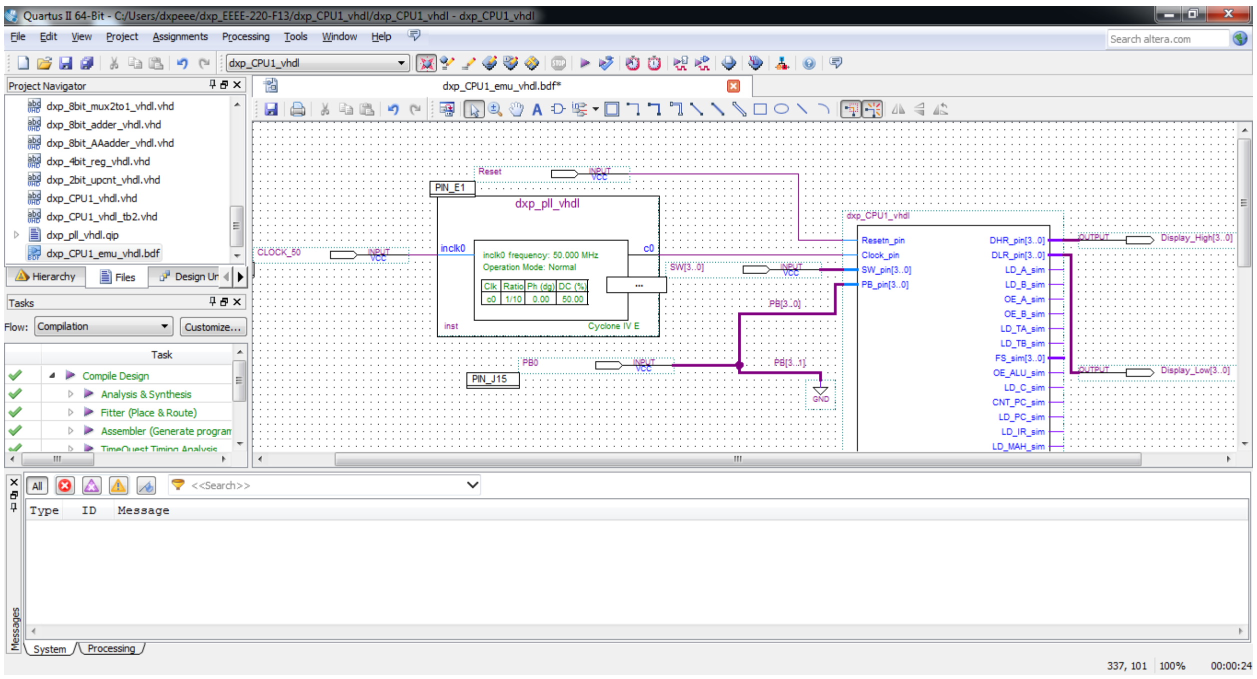


30) Connect c0 and any other necessary clocks to your CPU clock input(s).

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

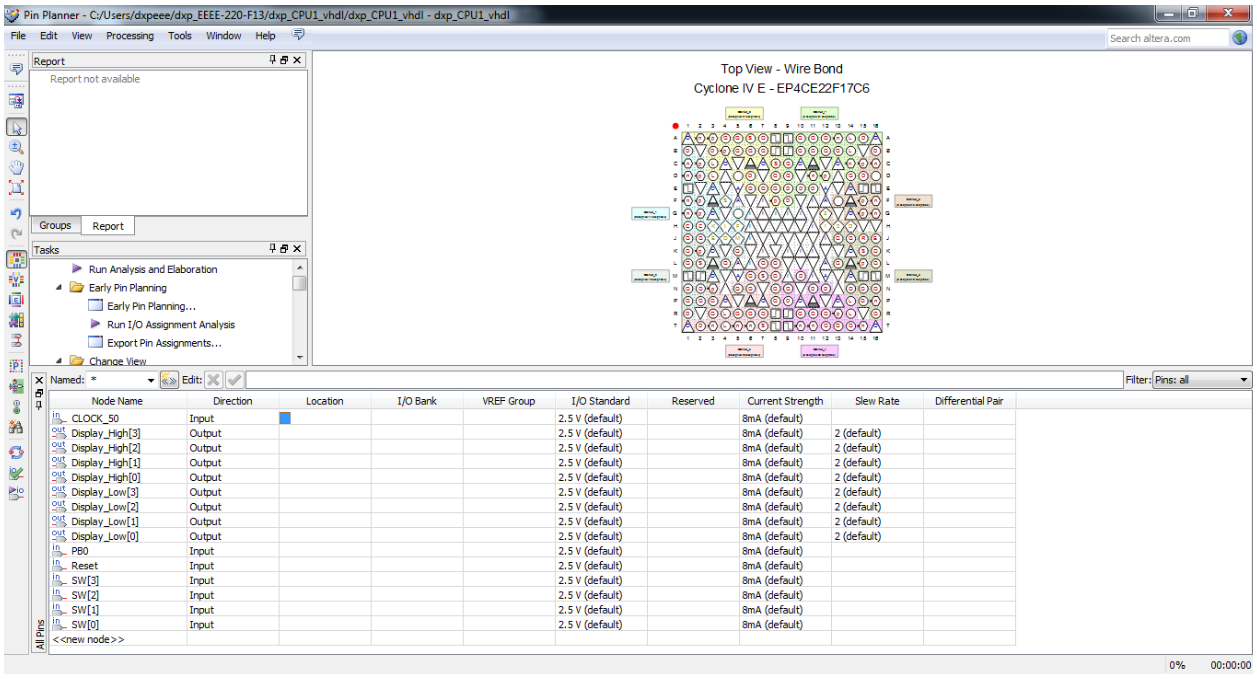


32) After adding input and output pins, your schematic should look similar to this (without the pin assignments yet):



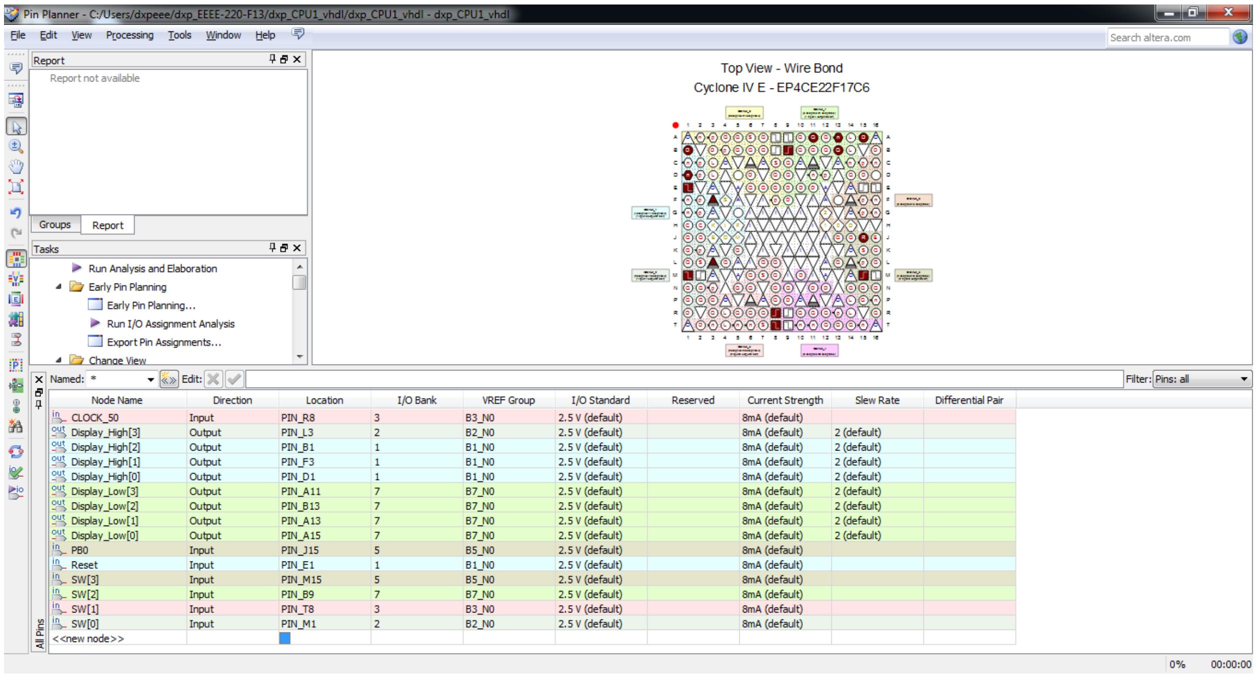
33) In preparation for pin assignments, right-click on the *.bdf file and set it as top-level entity. Then execute **Processing > Start > Start Analysis & Elaboration**.

34) Now, you'll make the necessary pin assignments. Select **Assignments > Pin Planner**. **<u>Note</u>**: If your top-level is not a schematic, you need to synthesize first before opening the Pin Planner.


Pin Planner - C:/Users/dxpeee/dxp_EEEE-220-F13/dxp_CPU1_vhdl/dxp_CPU1_vhdl - dxp_CPU1_vhdl
File Edit View Processing Tools Window Help
Report
Report not available
Groups
Report
Tasks
Run Analysis and Elaboration
Early Pin Planning
Early Pin Planning...
Run I/O Assignment Analysis
Export Pin Assignments...
Change View
Top View - Wire Bond
Cyclone IV E - EP4CE22F17C6
Named: *
Edit:
Filter: Pins: all
All Pins
0% 00:00:00

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair
CLOCK_50	Input	🟦			2.5 V (default)		8mA (default)		
Display_High[3]	Output				2.5 V (default)		8mA (default)	2 (default)	
Display_High[2]	Output				2.5 V (default)		8mA (default)	2 (default)	
Display_High[1]	Output				2.5 V (default)		8mA (default)	2 (default)	
Display_High[0]	Output				2.5 V (default)		8mA (default)	2 (default)	
Display_Low[3]	Output				2.5 V (default)		8mA (default)	2 (default)	
Display_Low[2]	Output				2.5 V (default)		8mA (default)	2 (default)	
Display_Low[1]	Output				2.5 V (default)		8mA (default)	2 (default)	
Display_Low[0]	Output				2.5 V (default)		8mA (default)	2 (default)	
PB0	Input				2.5 V (default)		8mA (default)		
Reset	Input				2.5 V (default)		8mA (default)		
SW[3]	Input				2.5 V (default)		8mA (default)		
SW[2]	Input				2.5 V (default)		8mA (default)		
SW[1]	Input				2.5 V (default)		8mA (default)		
SW[0]	Input				2.5 V (default)		8mA (default)		
<<new node>>									

35) 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):


Pin Planner - C:/Users/dxpeee/dxp_EEEE-220-F13/dxp_CPU1_vhdl/dxp_CPU1_vhdl - dxp_CPU1_vhdl
File Edit View Processing Tools Window Help
Report
Report not available
Groups Report
Tasks
Run Analysis and Elaboration
Early Pin Planning
Early Pin Planning...
Run I/O Assignment Analysis
Export Pin Assignments...
Change View
Top View - Wire Bond
Cyclone IV E - EP4CE22F17C6
Named: *
Edit:
Filter: Pins: all
All Pins
0%
00:00:00

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair
CLOCK_50	Input	PIN_R8	3	B3_N0	2.5 V (default)		8mA (default)		
Display_High[3]	Output	PIN_L3	2	B2_N0	2.5 V (default)		8mA (default)	2 (default)	
Display_High[2]	Output	PIN_B1	1	B1_N0	2.5 V (default)		8mA (default)	2 (default)	
Display_High[1]	Output	PIN_F3	1	B1_N0	2.5 V (default)		8mA (default)	2 (default)	
Display_High[0]	Output	PIN_D1	1	B1_N0	2.5 V (default)		8mA (default)	2 (default)	
Display_Low[3]	Output	PIN_A11	7	B7_N0	2.5 V (default)		8mA (default)	2 (default)	
Display_Low[2]	Output	PIN_B13	7	B7_N0	2.5 V (default)		8mA (default)	2 (default)	
Display_Low[1]	Output	PIN_A13	7	B7_N0	2.5 V (default)		8mA (default)	2 (default)	
Display_Low[0]	Output	PIN_A15	7	B7_N0	2.5 V (default)		8mA (default)	2 (default)	
PB0	Input	PIN_J15	5	B5_N0	2.5 V (default)		8mA (default)		
Reset	Input	PIN_E1	1	B1_N0	2.5 V (default)		8mA (default)		
SW[3]	Input	PIN_M15	5	B5_N0	2.5 V (default)		8mA (default)		
SW[2]	Input	PIN_B9	7	B7_N0	2.5 V (default)		8mA (default)		
SW[1]	Input	PIN_T8	3	B3_N0	2.5 V (default)		8mA (default)		
SW[0]	Input	PIN_M1	2	B2_N0	2.5 V (default)		8mA (default)		
<<new node>>									

36) 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.

37) 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

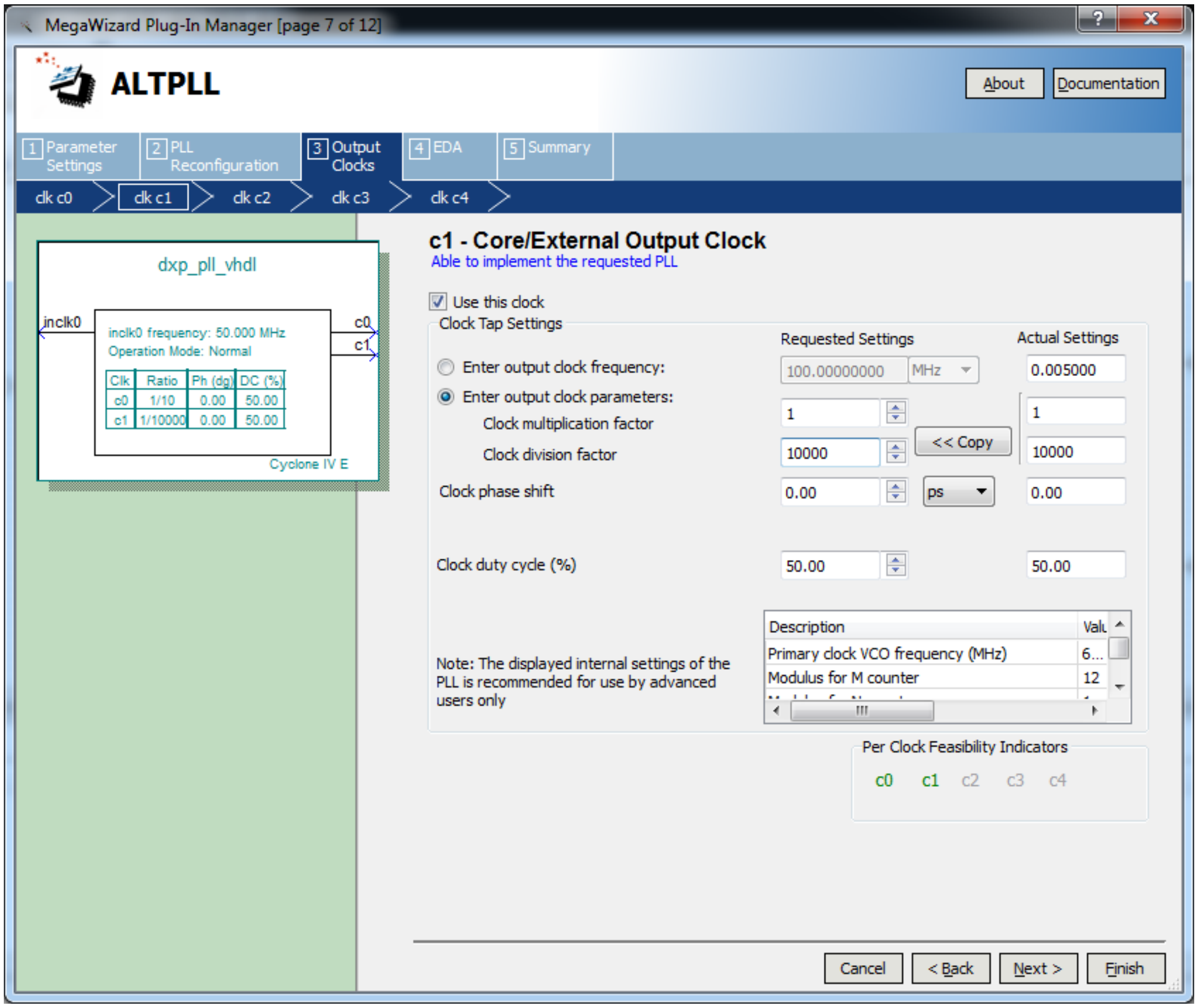
derive_pll_clocks

derive_clock_uncertainty

   d. Save this file as fml_CPU_emu.sdc

38) 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.

39) Now you can fully compile and then program the FPGA with your design.
40) To compile press on the play button. Unlike in the last several labs, we are performing a full compilation, i.e. ultimately generating a FPGA programming file.
41) To program the circuit onto the FPGA, select Tools > Programmer. If you connect your board, the programmer tool will automatically recognize it. Click program.
42) If you want you can choose to use a slower clock. You can do this by dividing the clock first by an arbitrary value. The maximum allowable value is 10,000. Double-click the fml_pll component and choose to output a second clock as shown below:



inclk0 frequency: 50.000 MHz
Operation Mode: Normal

Clk	Ratio	Ph (dg)	DC (%)
c0	1/10	0.00	50.00
c1	1/10000	0.00	50.00

c1 - Core/External Output Clock
Able to implement the requested PLL

☑ Use this clock

Clock Tap Settings

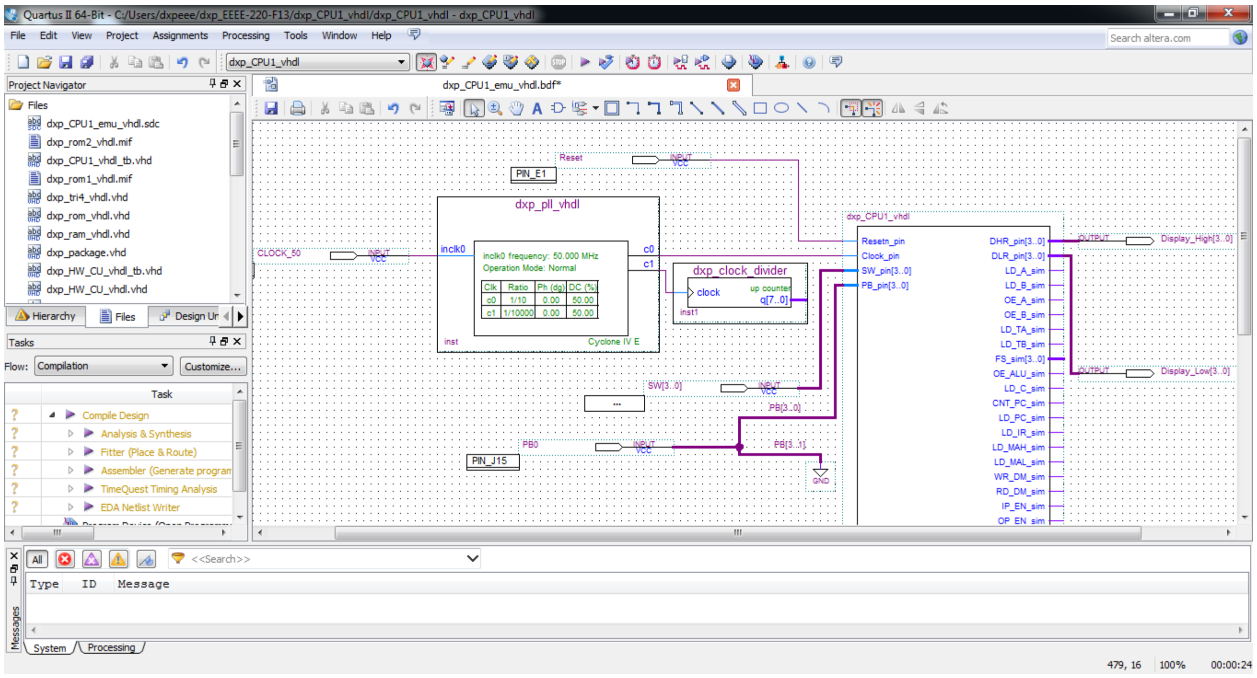
	Requested Settings		Actual Settings
○ Enter output clock frequency:	100.00000000	MHz	0.005000
● Enter output clock parameters:			
Clock multiplication factor	1		1
Clock division factor	10000		10000
Clock phase shift	0.00	ps	0.00
Clock duty cycle (%)	50.00		50.00

Note: The displayed internal settings of the PLL is recommended for use by advanced users only

Description	Valu
Primary clock VCO frequency (MHz)	6...
Modulus for M counter	12

Per Clock Feasibility Indicators
c0 c1 c2 c3 c4

43) 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.



44) 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.

45) You'll need to re-compile the design for the different test programs, and manually switch the clock source in the schematic.

46) Please be gentle with the switches when you set arbitrary values on the switches, as these are very small and may break easily.