ECE 380: Lab #2

Part I. VHDL for a simple circuit

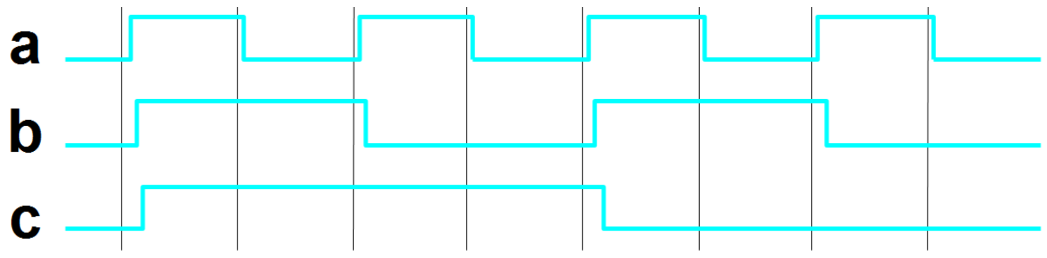
The objective of this practice is to generate and simulate (test) different type of design for a simple combinational logic. Please use three individual projects for the following tasks.

a	*b*	*c*	*f*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Design A: Write a VHDL model (entity and architecture) to describe the logic specify by the above truth table. Describe the circuit with AND, OR, NOT in your architecture body.

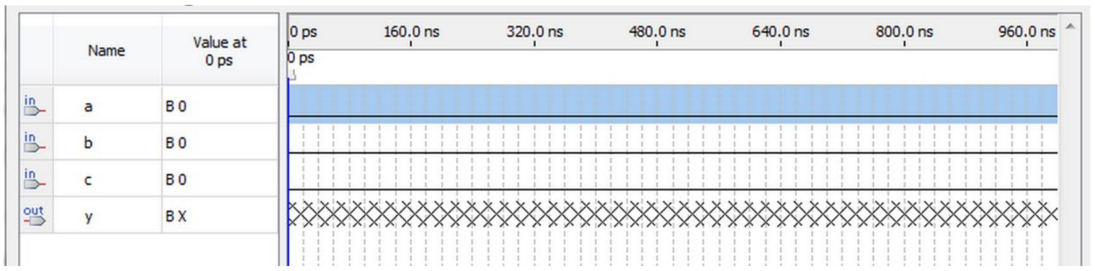
- Create a project using VHDL, following lab 01 related steps.

- Test your design using functional simulations. You need to generate all the combinations for those three inputs, i.e., 000, 001, 010, …,111.

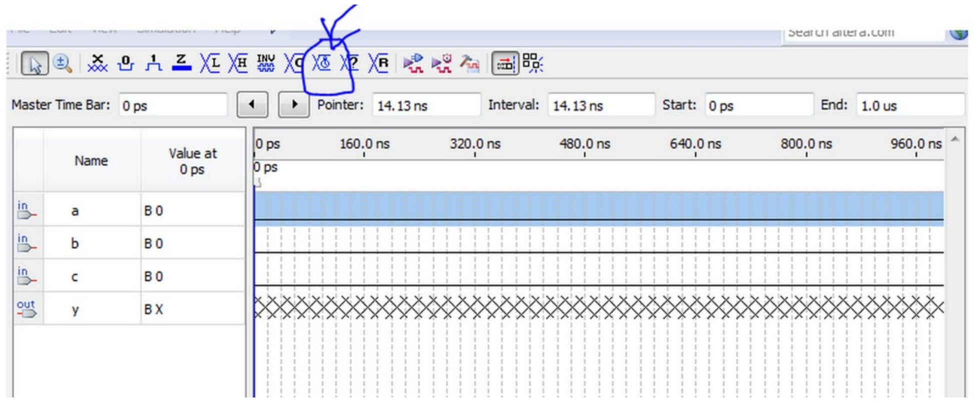


- With a ready VHDL code, compile (short-cut key, Ctrl+K).
- Generate waveform simulations. Follow the same procedure in lab 1 to open .vwf file. List all the input (A, B, C), and output (f) nodes and insert them in the waveform editor. Since there are 3 inputs, number of possible inputs are 2^3=8.

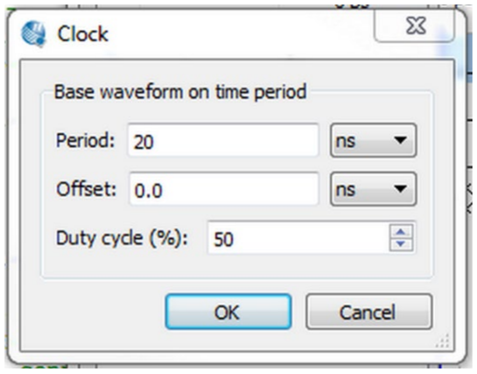
- Select the signal you want to edit in the waveform editor



- Click the icon "overwrite clock" on the toolbar



- Set the period of signal and click OK. A clock waveform will be generated in the waveform editor



Clock

Base waveform on time period

Period: 20 ns

Offset: 0.0 ns

Duty cycle (%): 50

- Set the period of signals A, B and C to 20, 40, and 80 ns respectively. Set the end time to 160 ns.

- Run functional simulations to generate the output waveform and save the screenshot of .vwf showing all the inputs and output.

Note: Similar tutorial materials can be found in *Introduction to Simulation of VHDL Designs,* supplement_quartusii_simulation_vhdl.pdf, in lab 01.

Design B: Create a schematic based on the NOR gate implementation.
- Test/simulate this design.
- Open .vwf and insert the clock signals for A, B, and C inputs as in Designs A and B.
- Perform functional simulations.
- Compare the generated output waveform with those from Designs A and B.

Part II. Generate a flashing LED

In this part, you are asked to follow a simple design and upload the design to the Altera's DE1 board. This simple design drives a LED flashing at a rate of one second.

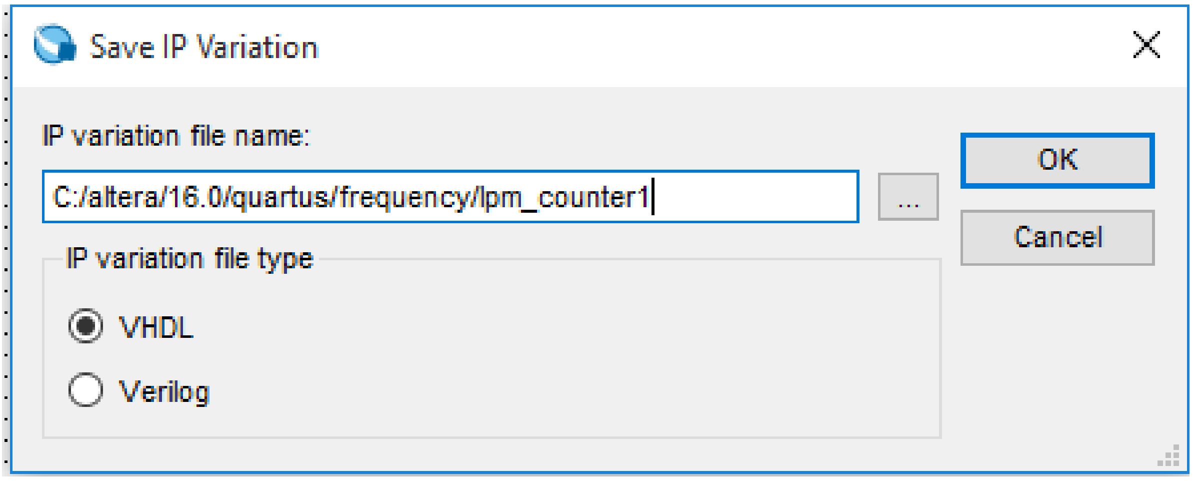
The DE1 board provides 50 MHz clock signals. To generate the required one-second period clock, you need to use a multi-bit counter, which can be implemented by a library module LPM_COUNTER, in Quartus II, to divide the onboard 50 MHz clock. The customized LPM_COUNTER uses 26 bits can count from 0 to 50,000-1. The counter reduces the frequency of the 50 MHz clock signal and its output behaves as a clock signal changing at the rate of one second.

Requirement: Demonstrate a LED flashing at 1 second rate.

Step 1: Create a new project named frequency. Under the project, create a new BDF file and save the file as frequency.bdf.

Step 2: Configure lpm_conter and draw the schematics
- Go to **Library** → Arithmetics → **lpm_Counter**

When you click on lpm_counter, a dialogue window "Save IP variation" appears as below:


Save IP Variation

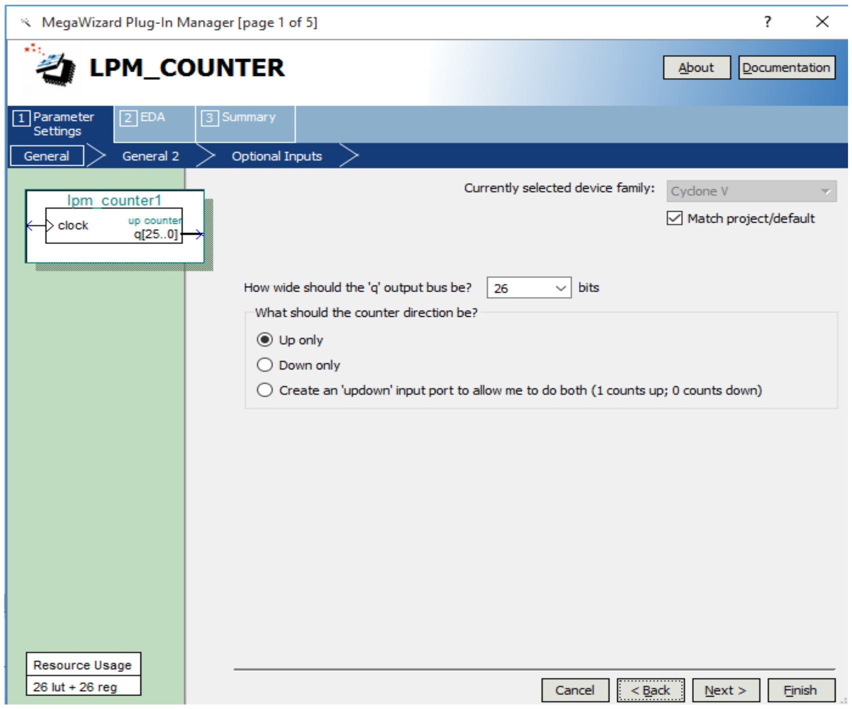
IP variation file name: C:/altera/16.0/quartus/frequency/lpm_counter1

IP variation file type: [✓ VHDL] Verilog

Save it under the project 'frequency' as lpm_counter1(C:\altera\16.0\quartus\frequency\lpm-counter1). Make sure you save it as VHDL and click OK.

After clicking OK, MegaWizard Plug-in manager appears where parameters of the counter can be changed. It goes through the following six steps:

1. Change the number of bits to **26**, click Next.


MegaWizard Plug-In Manager [page 1 of 5]

LPM_COUNTER

Currently selected device family: Cyclone V

[✓] Match project/default

How wide should the 'q' output bus be? 26 bits

What should the counter direction be?

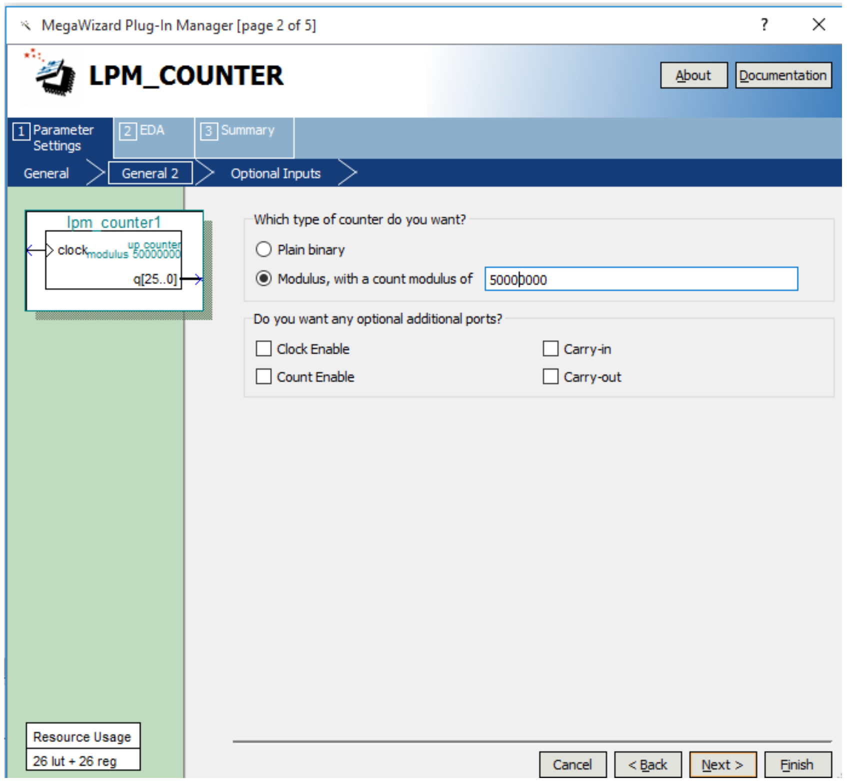
[●] Up only

[○] Down only

[○] Create an 'updown' input port to allow me to do both (1 counts up; 0 counts down)

Resource Usage: 26 lut + 26 reg

2. Insert **50,000,000** as modulus with a count modulus, click Next.


MegaWizard Plug-In Manager [page 2 of 5]

LPM_COUNTER

Which type of counter do you want?

[○] Plain binary

[●] Modulus, with a count modulus of 50000000

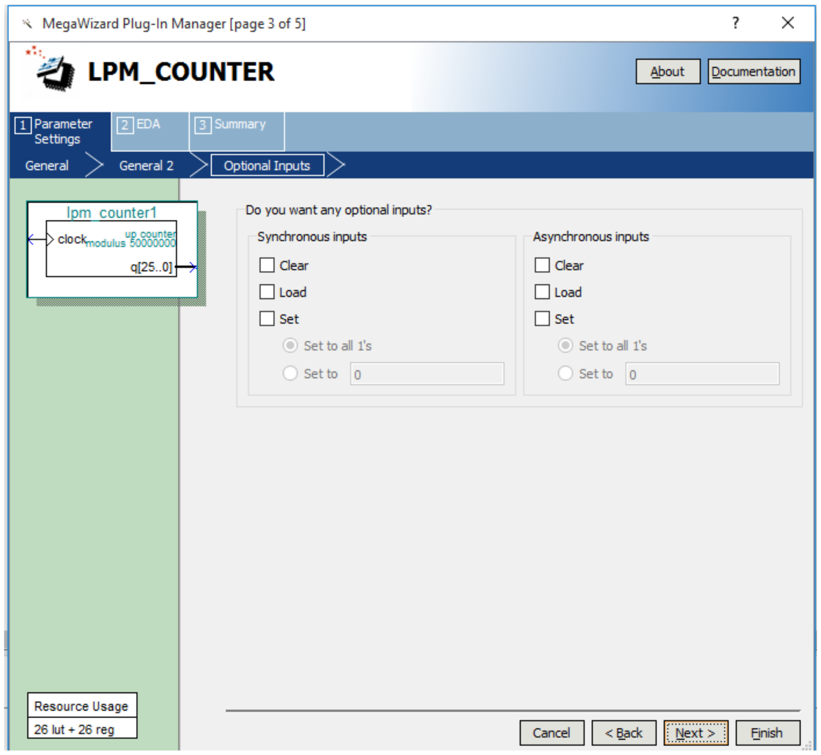
Do you want any optional additional ports?

[ ] Clock Enable [ ] Carry-in

[ ] Count Enable [ ] Carry-out

Resource Usage: 26 lut + 26 reg

3. Click Next.


MegaWizard Plug-In Manager [page 3 of 5]

LPM_COUNTER

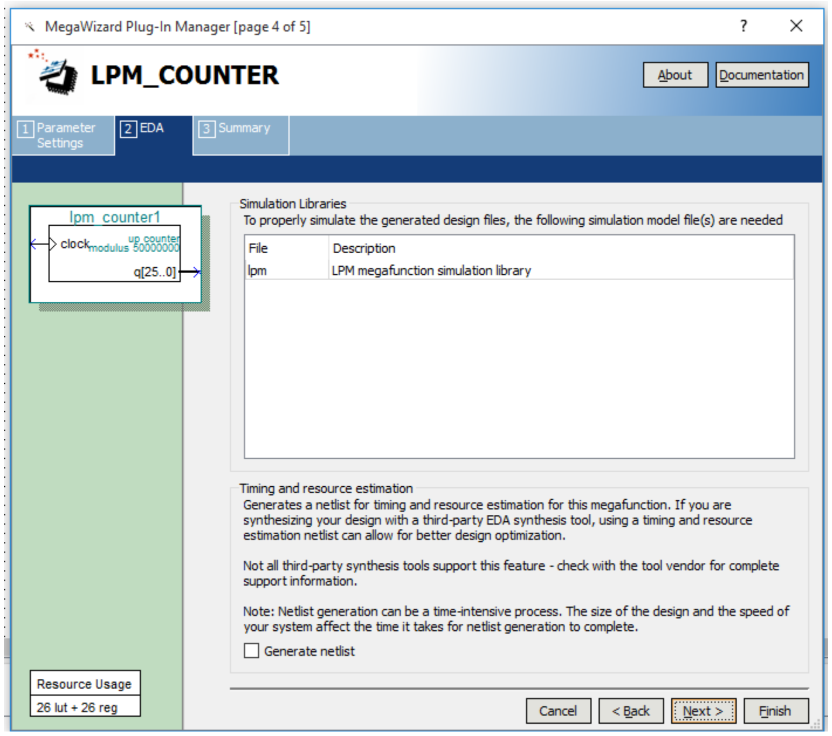
Do you want any optional inputs?

Synchronous inputs
- [ ] Clear
- [ ] Load
- [ ] Set
  - Set to all 1's
  - Set to 0

Asynchronous inputs
- [ ] Clear
- [ ] Load
- [ ] Set
  - Set to all 1's
  - Set to 0

Resource Usage
26 lut + 26 reg

4. Click Next.


MegaWizard Plug-In Manager [page 4 of 5]

LPM_COUNTER

Simulation Libraries
To properly simulate the generated design files, the following simulation model file(s) are needed

File	Description
lpm	LPM megafunction simulation library

Timing and resource estimation
Generates a netlist for timing and resource estimation for this megafunction. If you are synthesizing your design with a third-party EDA synthesis tool, using a timing and resource estimation netlist can allow for better design optimization.

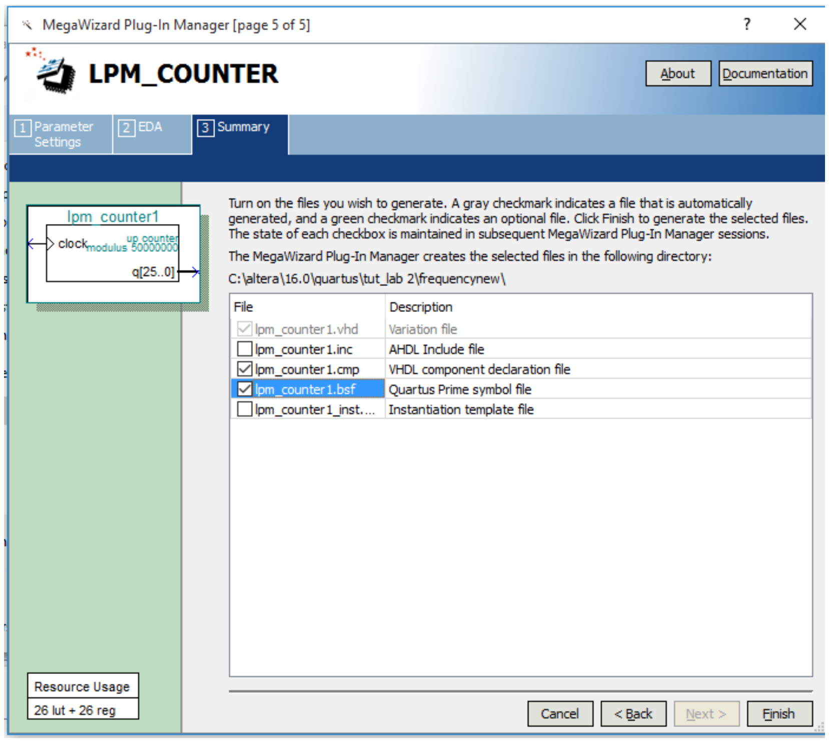
Not all third-party synthesis tools support this feature - check with the tool vendor for complete support information.

Note: Netlist generation can be a time-intensive process. The size of the design and the speed of your system affect the time it takes for netlist generation to complete.

- [ ] Generate netlist

Resource Usage
26 lut + 26 reg

5. Select the type of files you want Quartus II to generate. Select “VHDL component declaration file”, “symbol file” and then click Finish.


MegaWizard Plug-In Manager [page 5 of 5]

LPM_COUNTER

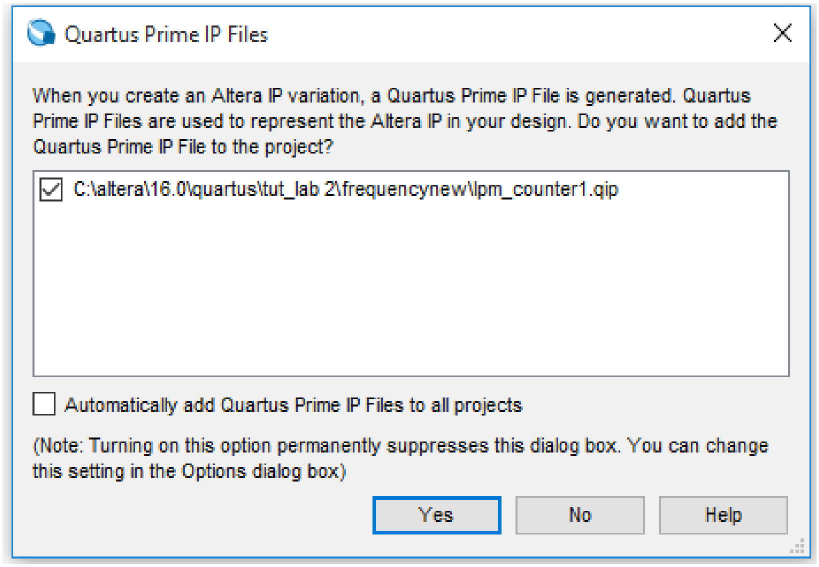
Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

The MegaWizard Plug-In Manager creates the selected files in the following directory:

C:\altera\16.0\quartus\tut_lab 2\frequencynew\

File	Description
✓ lpm_counter1.vhd	Variation file
☐ lpm_counter1.inc	AHDL Include file
✓ lpm_counter1.cmp	VHDL component declaration file
✓ lpm_counter1.bsf	Quartus Prime symbol file
☐ lpm_counter1_inst....	Instantiation template file

6. After that, click Yes to add lpm_counter1 file to the existing projects.


Quartus Prime IP Files

When you create an Altera IP variation, a Quartus Prime IP File is generated. Quartus Prime IP Files are used to represent the Altera IP in your design. Do you want to add the Quartus Prime IP File to the project?

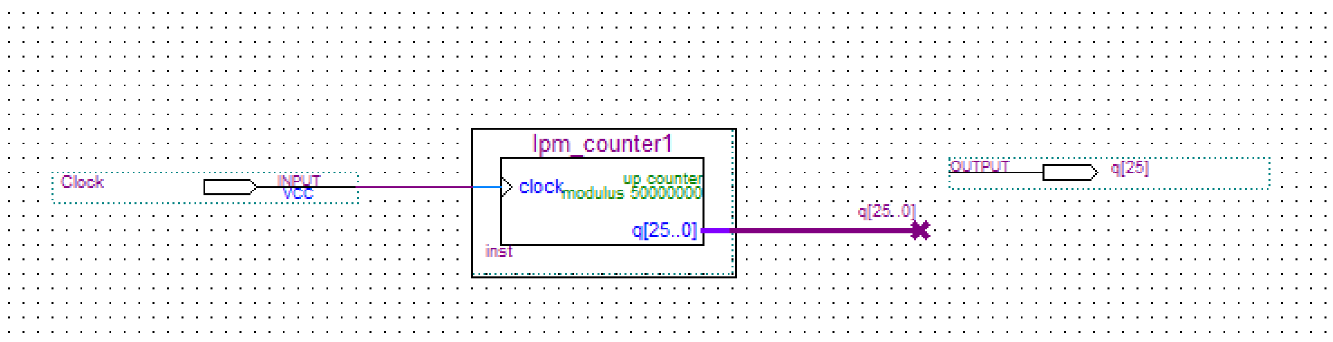
☑ C:\altera\16.0\quartus\tut_lab 2\frequencynew\lpm_counter1.qip

☐ Automatically add Quartus Prime IP Files to all projects

(Note: Turning on this option permanently suppresses this dialog box. You can change this setting in the Options dialog box)

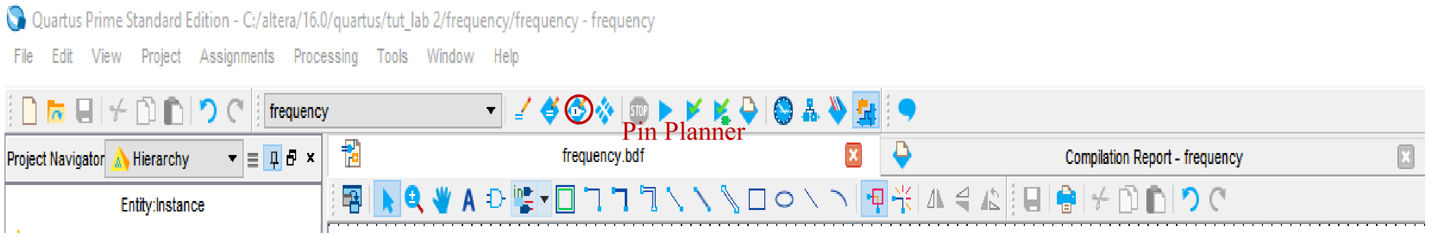
Once added, lpm_counter1 can be found in the symbol tool and can be dragged on the block editor.

Step3: With lpm_counter1 in the block editor, fetch an Input pin, connect it with lpm_conter1, and name the pin as Clock. Connect the wire on the output side and double click the wire to highlight it into blue. The highlighted blue indicates that the wire can be named. Name it as q[25..0]. Further, name the output pin as q[25].

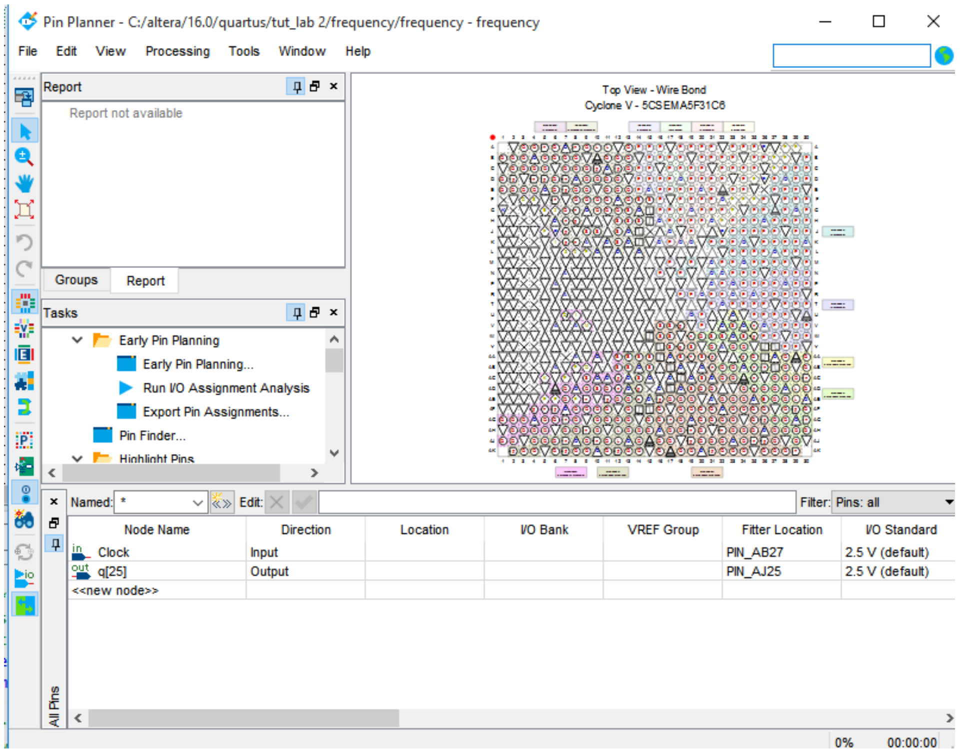


Step 3: Save and compile the project.

Step 4: Pin assignment. After the project is successfully compiled, assign the DE1 pins to both input and output of your design.



Click on pin planner and the following image appears.

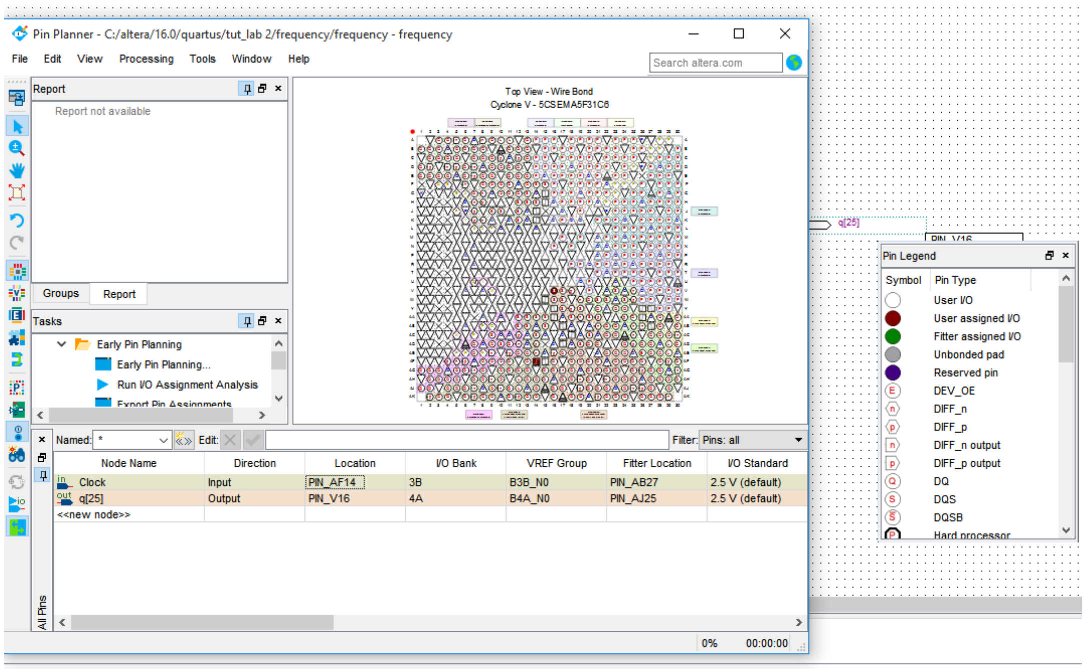


Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
Clock	Input				PIN_AB27	2.5 V (default)
q[25]	Output				PIN_AJ25	2.5 V (default)
<<new node>>						

View of DE 1 board before assigning pins

Click on location and type in PIN_AF14 to assign pin to clock signal (input) and type in PIN_V16 to assign q[25] (output).

The details of PIN numbers of clock input signal (Page 20) and output (Page 24) can be found in DE1 board manual (distributed along this lab document).

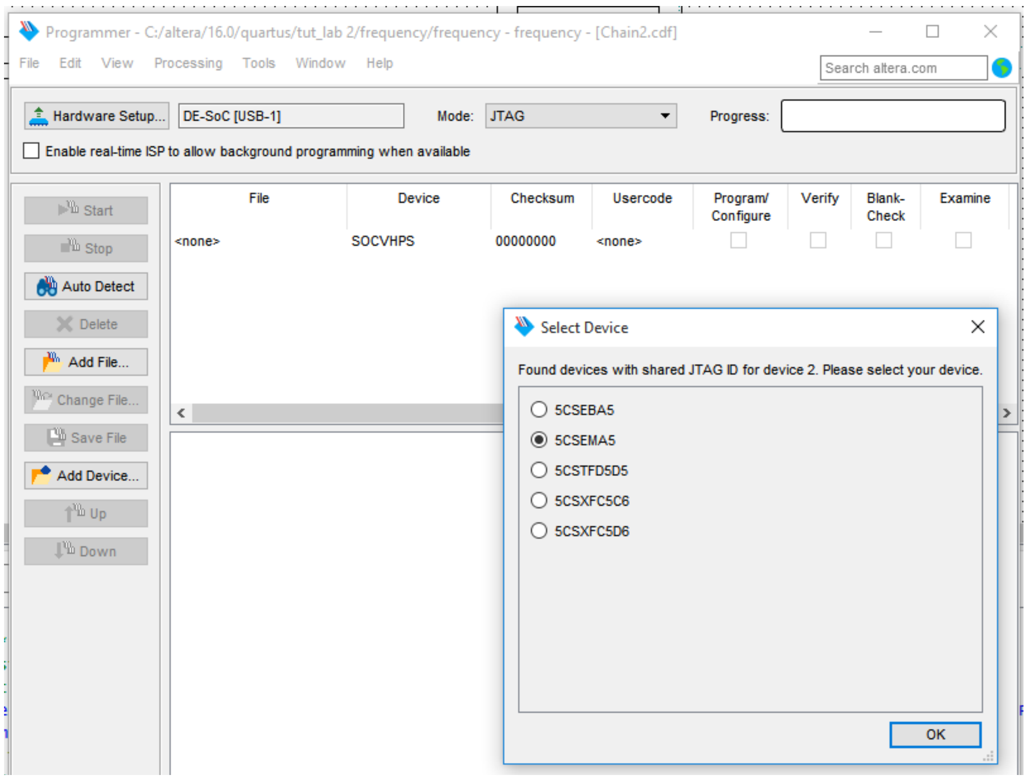


Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
Clock	Input	PIN_AF14	3B	B3B_N0	PIN_AB27	2.5 V (default)
q[25]	Output	PIN_V16	4A	B4A_N0	PIN_AJ25	2.5 V (default)
<<new node>>						

View of DE 1 board after assigning pins

Step 5: Compile again and upload the program to DE 1 board

- Switch ON the DE1 Board using the red button
- In Quartus II, go to Tools-→ Programmer→ Hardware Setup-→select 'DE-SoC'
- Make sure Mode is 'JTAG'
- Click auto-detect
- Select 5CSEMA



Programmer - C:/altera/16.0/quartus/tut_lab 2/frequency/frequency - frequency - [Chain2.cdf]
File Edit View Processing Tools Window Help
Search altera.com
Hardware Setup...
Start
Stop
Auto Detect
Delete
Add File...
Change File...
Save File
Add Device...
Up
Down
OK

DE-SoC [USB-1] Mode: JTAG Progress:

Enable real-time ISP to allow background programming when available

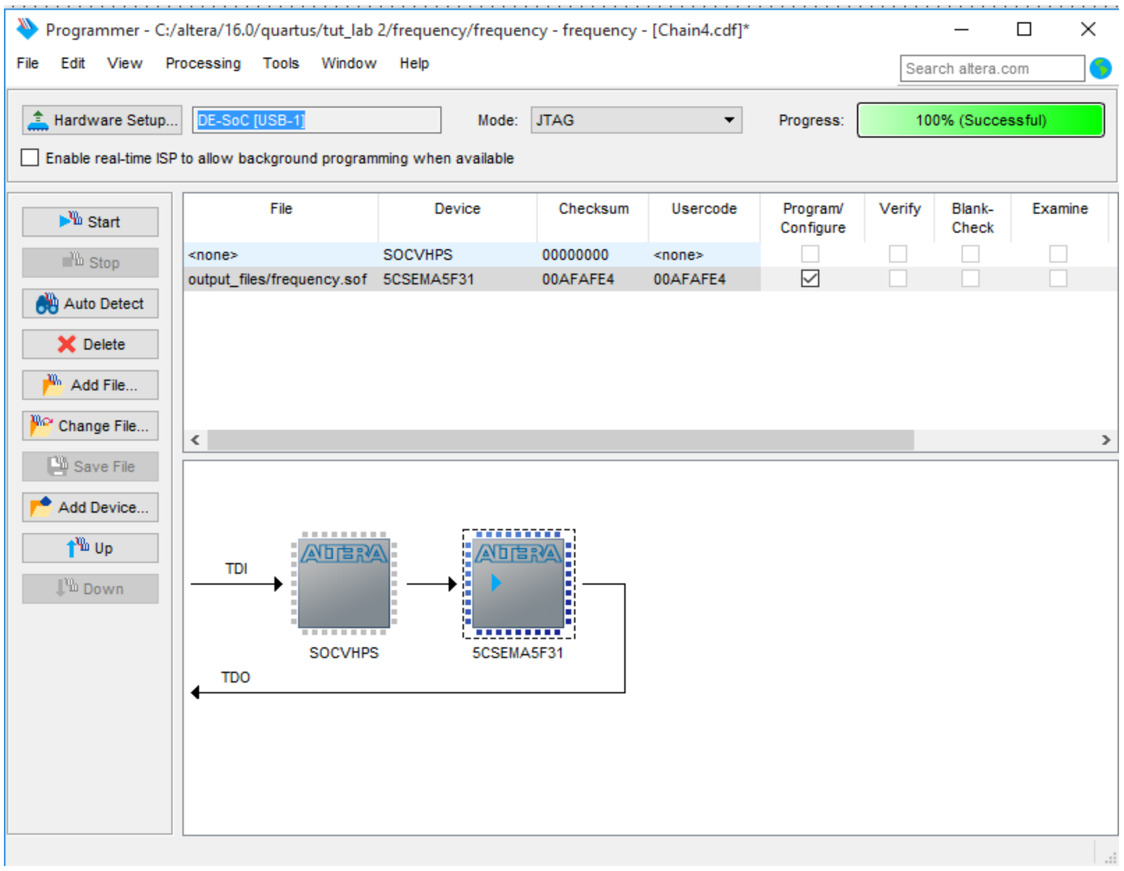
File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
<none>	SOCVHPS	00000000	<none>	☐	☐	☐	☐

Select Device

Found devices with shared JTAG ID for device 2. Please select your device.

- ○ 5CSEBA5
- ● 5CSEMA5
- ○ 5CSTFD5D5
- ○ 5CSXFC5C6
- ○ 5CSXFC5D6

- Right click on 5CSEMA5-→add file-→output files-→select the frequency.sof file (in output files of the directory)
- Click on the Program/Configure checkbox on the 5CSEMA5 line
- Click ‘Start’.
- At this point, your program has been transferred to the DE1 board. You should see the LED flashing on DE1 Board.


Programmer - C:/altera/16.0/quartus/tut_lab 2/frequency/frequency - frequency - [Chain4.cdf]*

Hardware Setup... DE-SoC [USB-1] Mode: JTAG Progress: 100% (Successful)

☐ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
<none>	SOCVHPS	00000000	<none>	☐	☐	☐	☐
output_files/frequency.sof	5CSEMA5F31	00AFAFE4	00AFAFE4	✓	☐	☐	☐

You can also follow the youtube link for the implementation:

https://www.youtube.com/watch?v=sbUAJByFb30

Note that our Quartus II version is different from the one from the video. You should adjust accordingly, based on the instructions provided above.

Write the Lab Report: Please follow the sample report format as in Lab 01 and do the following in the lab report:

- **Print the source codes/schematics (make sure it is well commented) for Part I and the schematics for Part II.**
- **Provide waveforms for all designs in Part I.**
- **Mark the printouts to show each test case for Part I.**
- **Explain your test procedures for Part I and Part II.**

Print this page, bring the hardcopy to your lab session, get it signed by the TA after the demo, and submit it along with your report

Lab demo (50 pts)	**Score**	**TA initial**
Part I (Designs A, B), 25 pts		
Report (50 pts)		
Lab 2 Total Grade (100 pts)		
Bonus: Quartus Prime Installation (25 pts) (Note: You only get one time 25 pts, not both times)		

HW #2 (100 points)

1) (30 pts) Use two types of methods, algebraic manipulation and truth tables, to prove the following Boolean statements.

a) $f(x_1, x_2) = \prod M(0, 3) = \sum m(1, 2)$ (Note these are two input functions)

b) $\bar{x}yz + x\bar{y}z + xy\bar{z} + xyz = xy + yz + xz$

c) $f(x_1, x_2) = \sum m(0,1,2) = M_3$ (Assume that the minterms and maxterms are for the two input functions).

2) (10 pts) Draw two types of the AND-OR schematic for the following function: $f(x, y, z) = \prod M(0, 3, 6)$; a) Product-Of-Sums (POSs) form, or the maxterm form. b) Sum-Of-Products (SOPs) form, or a minterm form.

3) (60 pts) For the logic function,

$$Y = (A + \bar{C})(A + \bar{B} + C)(\bar{A} + B + C)$$

a) Develop a truth table for the following standard POS expressions:
b) Based on the truth table developed, draw the schematic using NAND gates only
c) Based on the truth table developed, draw the schematic using NOR gates only