

# COMBINATIONAL DATAPATHS

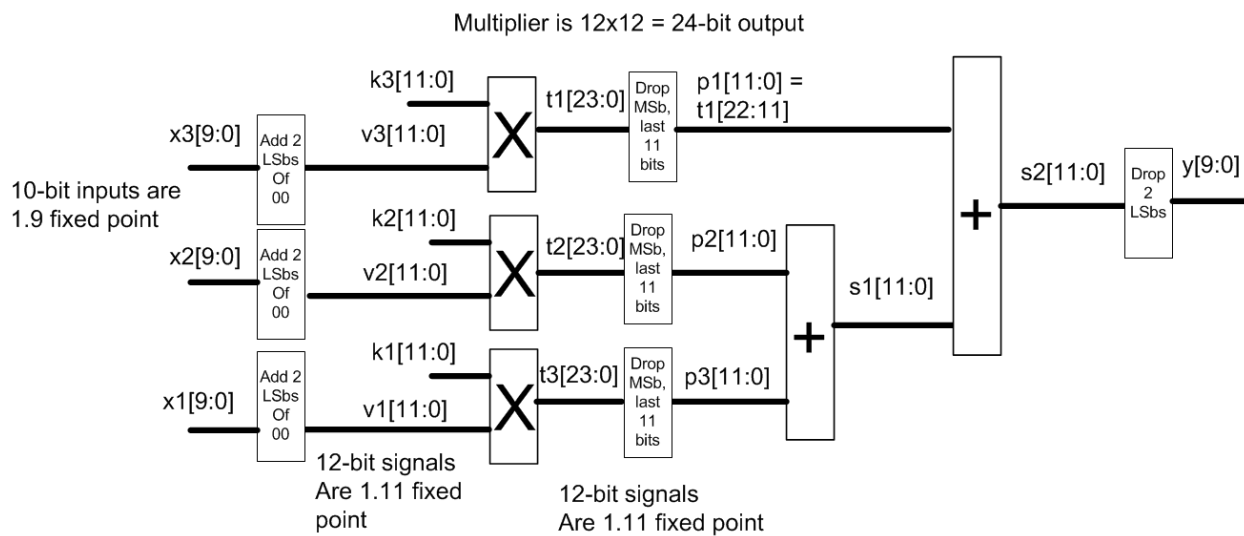
This lab has you implement a combinational datapath containing adders and multipliers. You will run both behavioral and post place & route simulations, run the timing analyzer, and view the placed/routed design in the FPGA viewer.

## BACKGROUND

Review the notes on multiplication and fixed-point representation.

## TASK DESCRIPTION

You are to create a Verilog module named *lab4dpath* that implements the following schematic:



Busses k1, k2, k3 are constants, and their values are specified in the *lab4\_computations.xlsx* file that is in the lab4 zip archive.

You will implement two different versions of this datapath, one will use LUT-based multipliers and the other will use hardmacro multipliers from the DSP blocks.

### Implementation Hints:

- You may only use assignment statements or component instantiations, you may not use an always block (sequential statements).
- You will need to expand the input values by adding two LSbs with values zero. This can be done using concatenation as follows:

```
wire [11:0] v1;
```

```
assign v1 = {x1,2'b00};    //concatenate 2 LSbs of zero
```

- When you have to drop bits, just choose which bits you want to keep by the bus indices:

```
assign y = s2[11:2];    //drop two LSbs of s2 to form y
```

## Procedure

1. Download the zip archive associated with the lab.
2. This contains the following files:
  - *lab4dpath.v* -- complete this module
  - *tb\_lab4dpath.v* – test bench
  - *multadd\_vectors.txt* – input vector file for testbench.
  - *lab4\_computations.xlsx* – spreadsheet file that contains the test vectors in decimal form, shows the intermediate calculations in hex and decimal. You can use this to help debug your design.
  - *report.doc* – report file that needs to be filled out. You should open this file, and fill out the requested information as you perform the remaining steps.
3. Create a project named *lab4\_part1* using the empty project template from the first lab and add the *lab4dpath.v*, *tb\_lab4dpath.v* files to it.
4. Copy the *multadd\_vectors.txt* file into the project directory. This is needed before you simulate your design.
5. Copy the *Basys3\_Master.xdc* file from the first lab to your project directory, and then add it to your project. Comment out all of the lines that have the 'set\_property' command for pins. This is easy to do by opening the file in Vivado, selecting a group of lines, and then use the "Toggle Line Comment" from the right-click menu. Then add the following lines to the file (does not matter where):

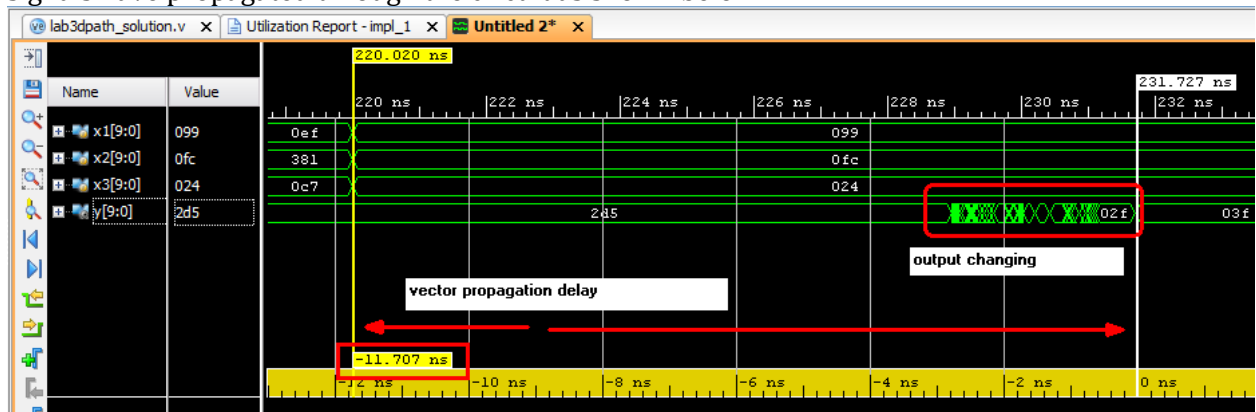
```
set_max_delay -from [get_ports x1] -to [get_ports y] 20
set_max_delay -from [get_ports x2] -to [get_ports y] 20
set_max_delay -from [get_ports x3] -to [get_ports y] 20
```

This will set a timing constraint that says the maximum propagation delay from the any input port to any output port should be less than 20 ns.

6. This lab requires you to use the Xilinx IP Catalog tool to implement the multipliers used in the schematic. You should generate the multiplier component before you modify the Verilog file. **This implementation will use LUT-based multipliers.** See the section titled *Multiplier Implementation* to learn how to generate the multiplier using IP Catalog. NOTE: the IP Catalog tool only has to be run ONE TIME to generate the multiplier implementation. You will instantiate this module three times in your design since it uses three multipliers. **For this lab, you must name your three multiplier instances as *netid\_a*, *netid\_b*, *netid\_c* (i.e., *bob3\_a*, *bob3\_b*, *bob3\_c*).** The Verilog module file (*modulename\_stub.v*) is generated in the *projectname.srscs/sources\_1/ip/modulename* folder – open this with a text editor so that you can discover the pin names that IP Catalog used for this module. You need this information when you write the statement that instantiates this module in your top-level module.

7. Implement your Verilog code in the *lab4dpath.v* file and test it with the *tb\_lab4dpath.v* testbench using a behavioral simulation. You are finished when all of the vectors pass as indicated by messages printed to the simulation console pane.
8. Once the behavioral simulation is working, do 'Run Implementation' and verify that the 'Post Implementation Timing Simulation' passes. You do not have to download it into your board.

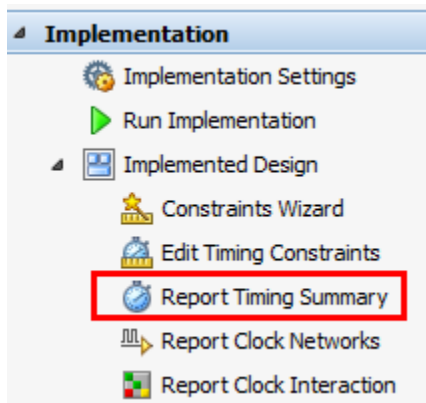
This simulation uses real timing delays. When a new vector is applied, different paths through the logic have different delays, so the output will change many times as different signals propagate at different rates. The output will finally settle to its correct value after all signals have propagated through the circuit as shown below.



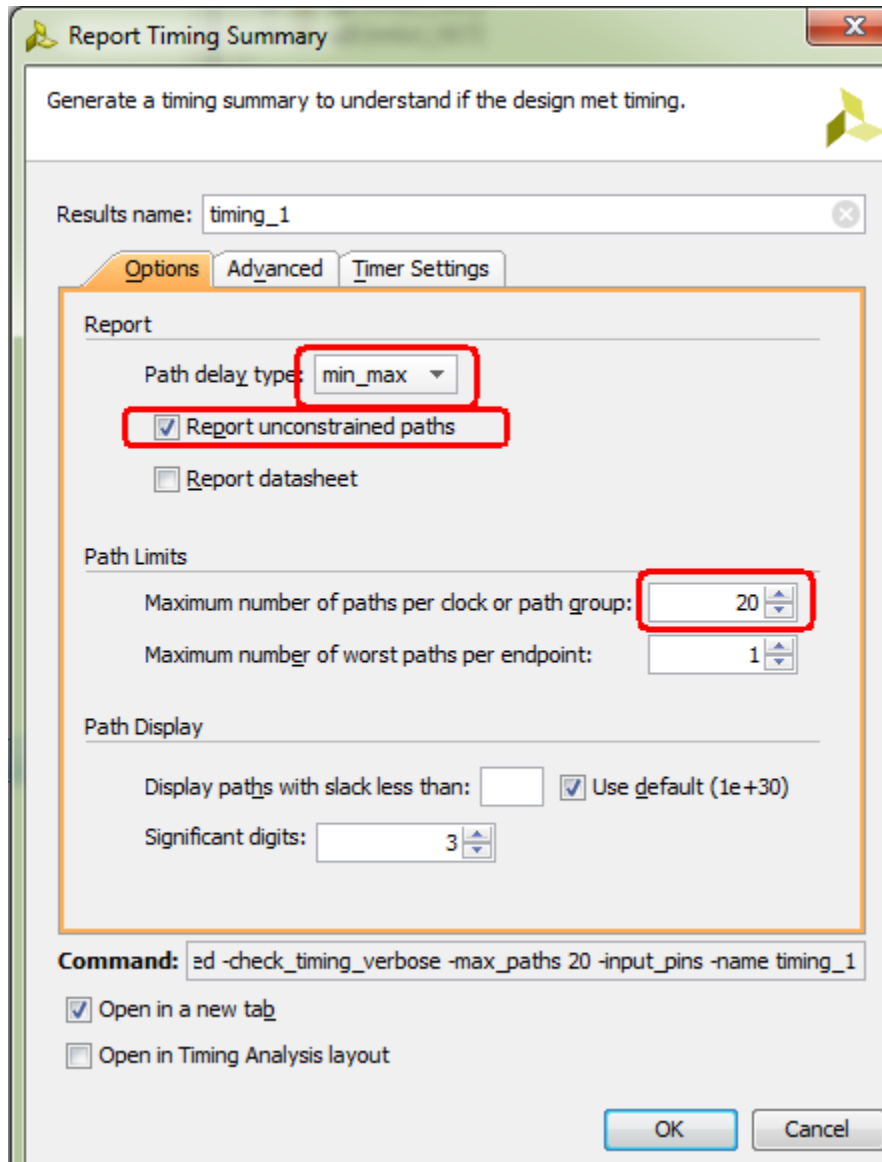
The above screen shot uses a 'Marker' to measure the time from the cursor to a marker (use right click menu to add a Marker).

Pick any vector in your design and capture a screenshot similar that above showing the output bouncing around and then settling to its final value after an input change. Include this in your report.

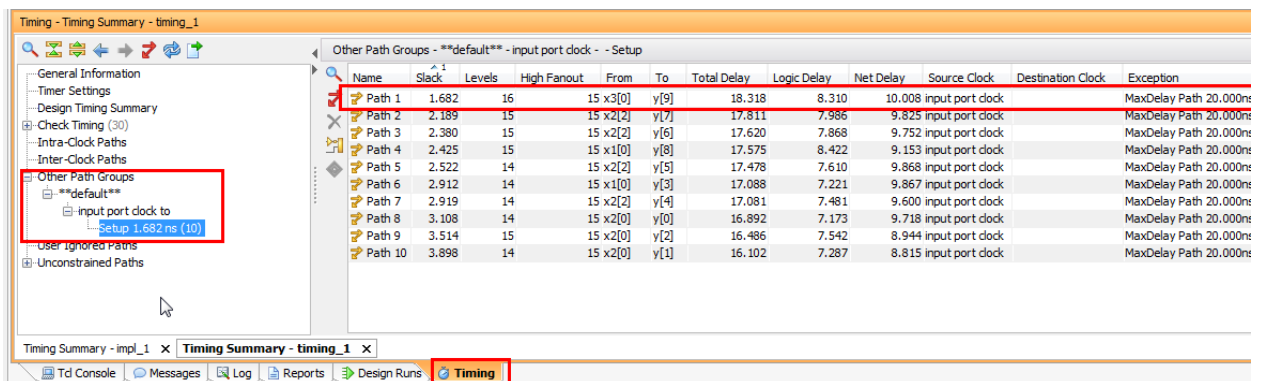
9. Open the implemented design. In the right-hand menu, click on the 'Report Timing Summary'.



In the "Report Timing Summary" panel, set the following options:



A new tab called 'Timing' will be created in Vivado output window. Click on the timing tab, and expand out the 'Other Path Groups' as shown below:

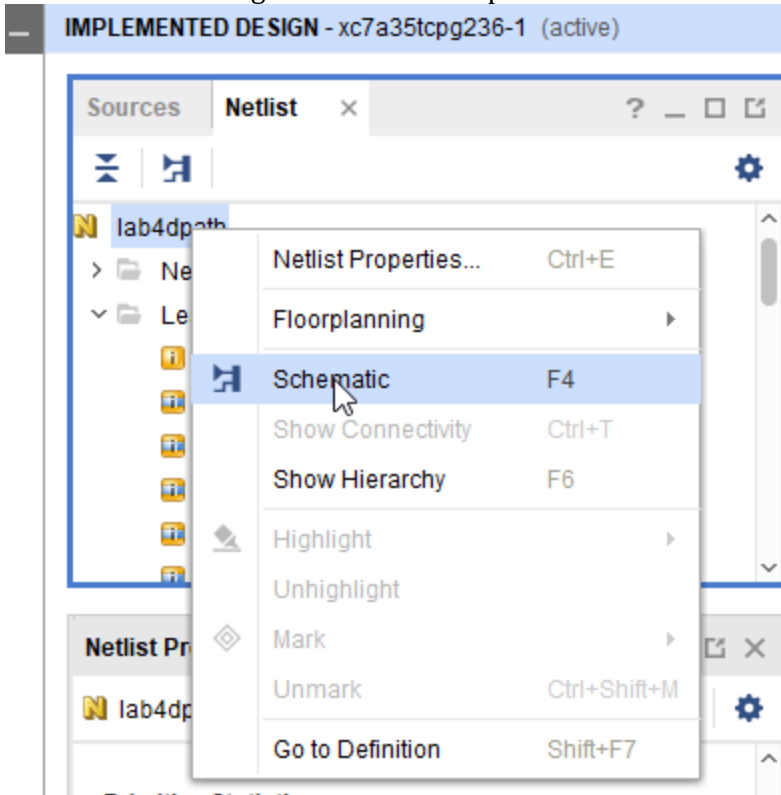


This reports the longest input pin to output pin delays in your design. Include this screenshot in your report.

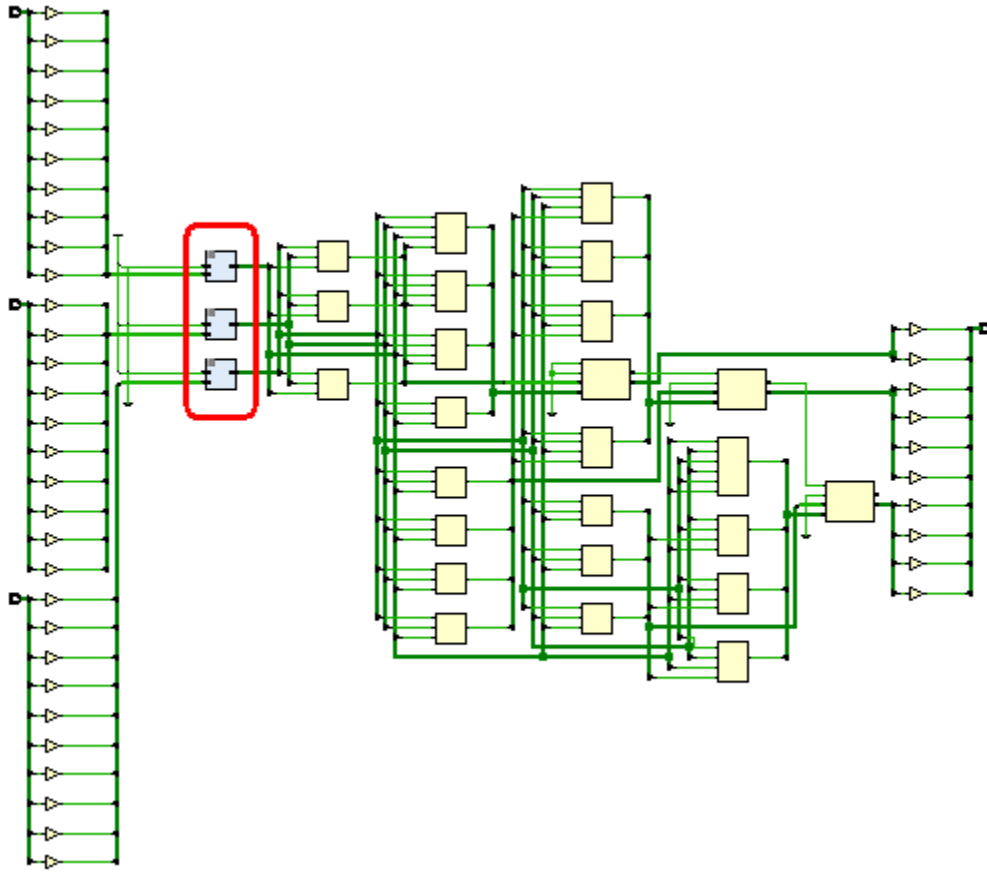
10. From the 'Reports' Tab, open the 'Place Design/place\_report\_utilization' Report'. Copy the tables that give 'Slice Logic' and 'Summary of Registers by Type' into your report. This design should have no registers reported.

Tcl Console Messages Log Reports x Design Runs Power DRC Timing	
<div> <div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div> <div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> </div> </div>	
Report	Report Type
impl_1_opt_report_timing_summary_0	Report timing summary (report_timing_summary)
Power Opt Design (power_opt_design)	
impl_1_power_opt_report_timing_summary_0	Report timing summary (report_timing_summary)
Place Design (place_design)	
impl_1_place_report_io_0	Report information about all the IO sites on the device (report_io)
impl_1_place_report_utilization_0	Report on utilization of resources on the targeted device (report_utilization)

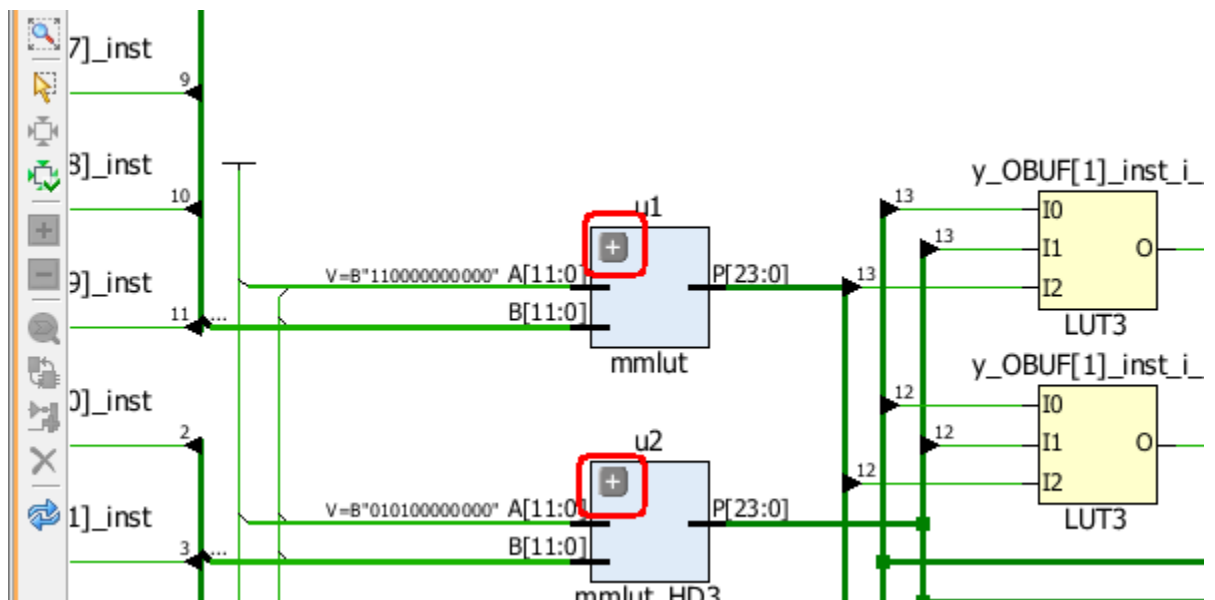
11. Ensure that the implemented design is open. In the 'Netlist' tab,, click on the top module name and use the right click menu to open the schematic of the generated logic.



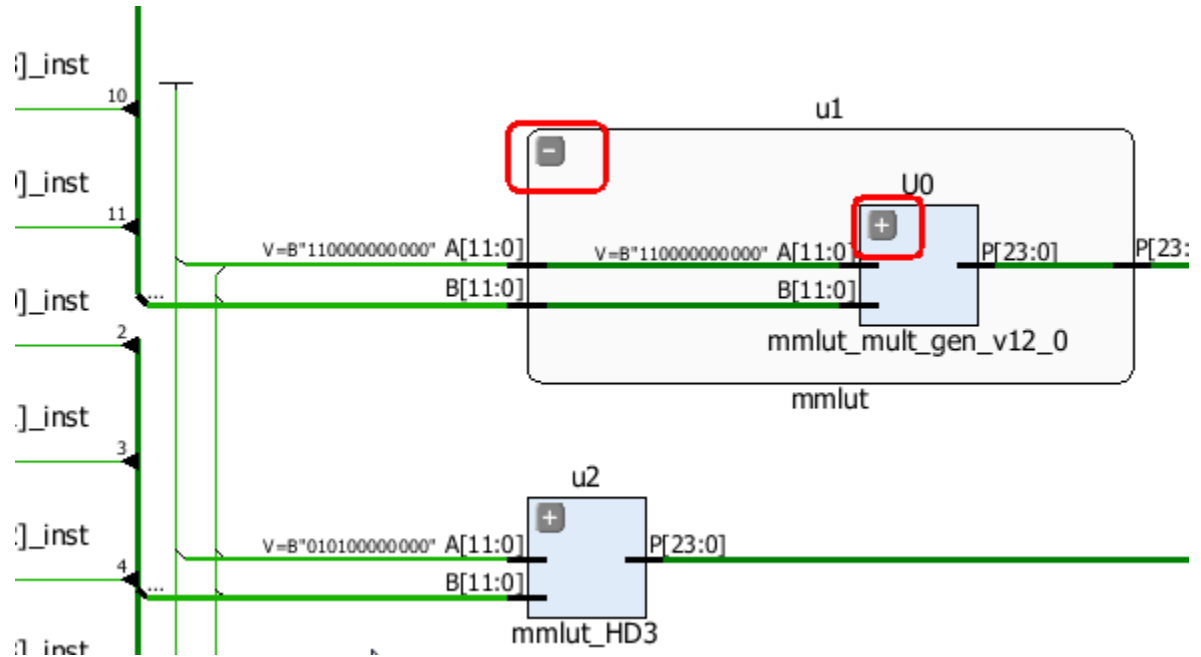
This will open a schematic that looks like:



The 3 blue boxes are my multipliers. If you zoom in, you will see a '+' sign, click on this expands this component.

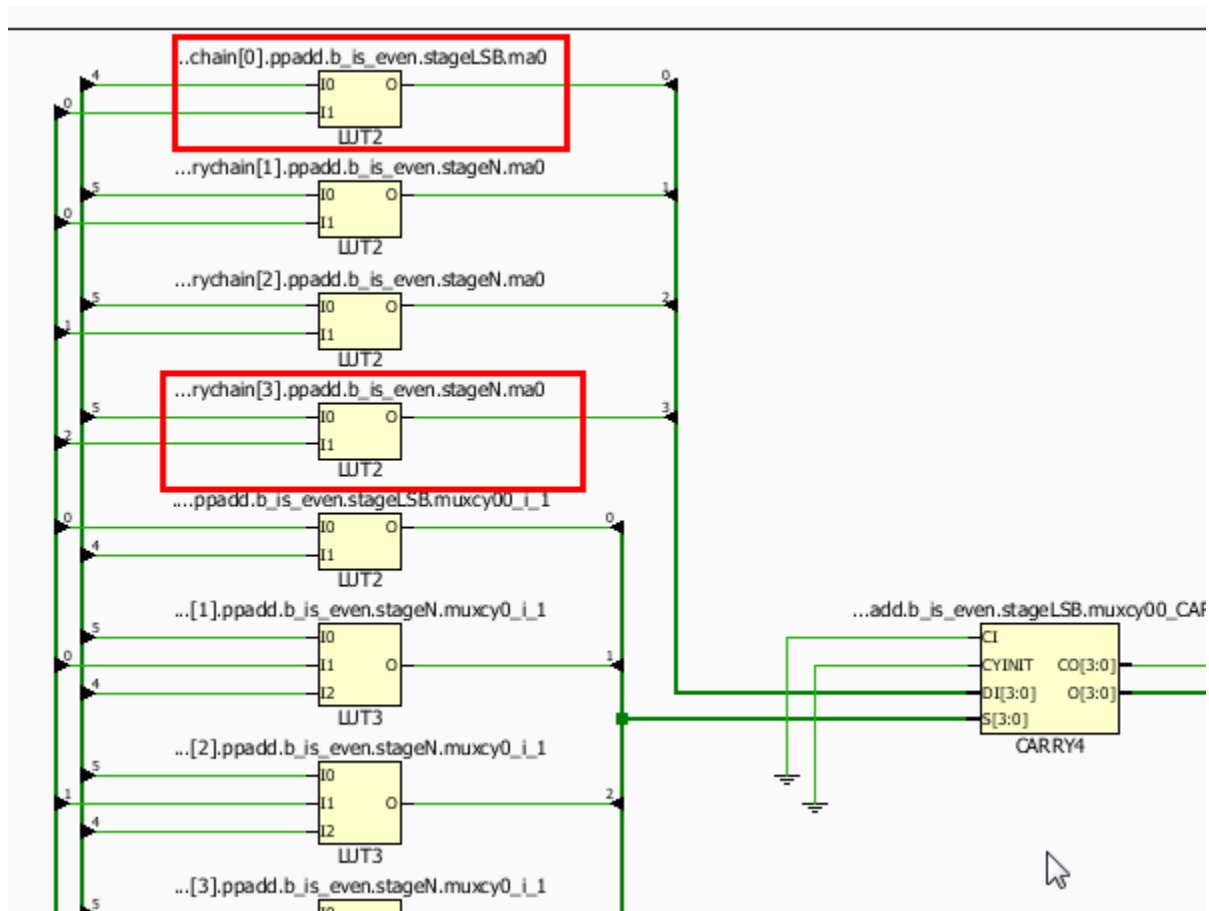


After expanding the top multiplier:



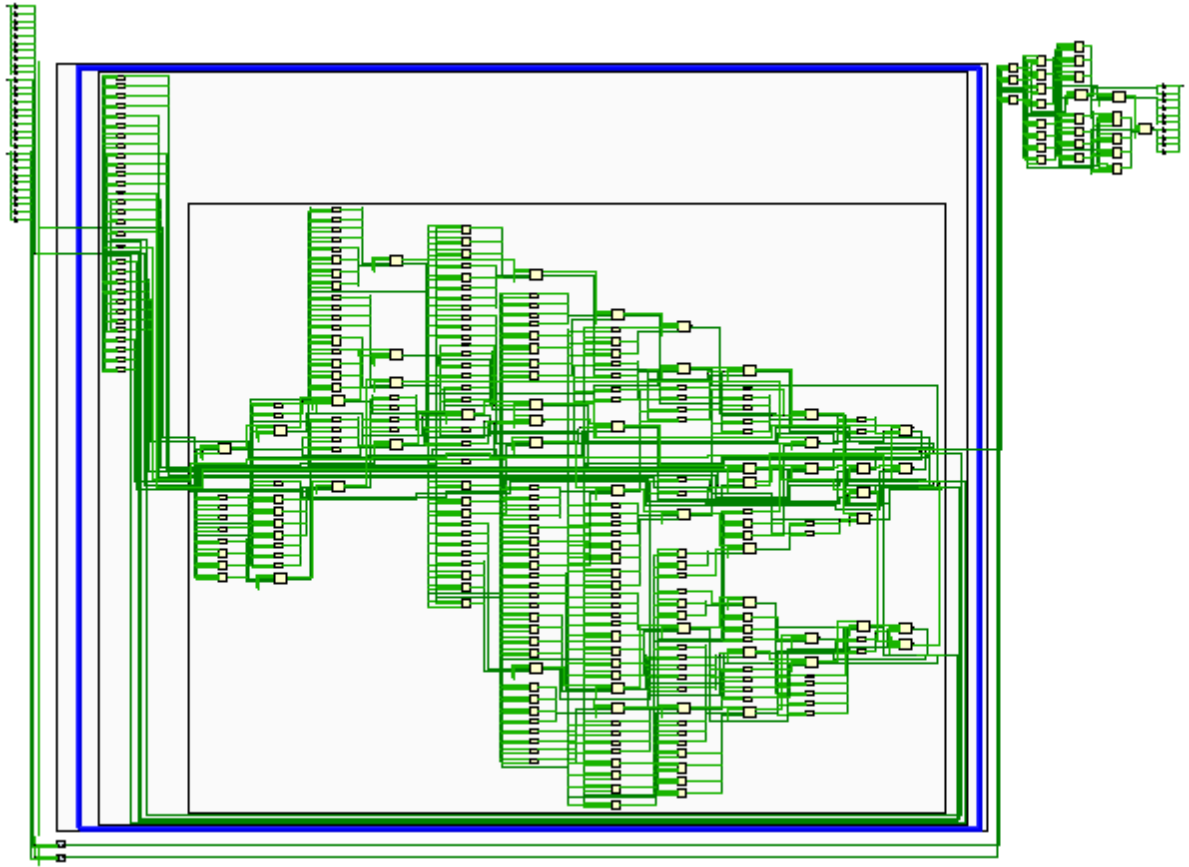
If you keep expanding, you will eventually reach a point where you can see individual lookup tables that cannot be expanded.

(zoomed in)



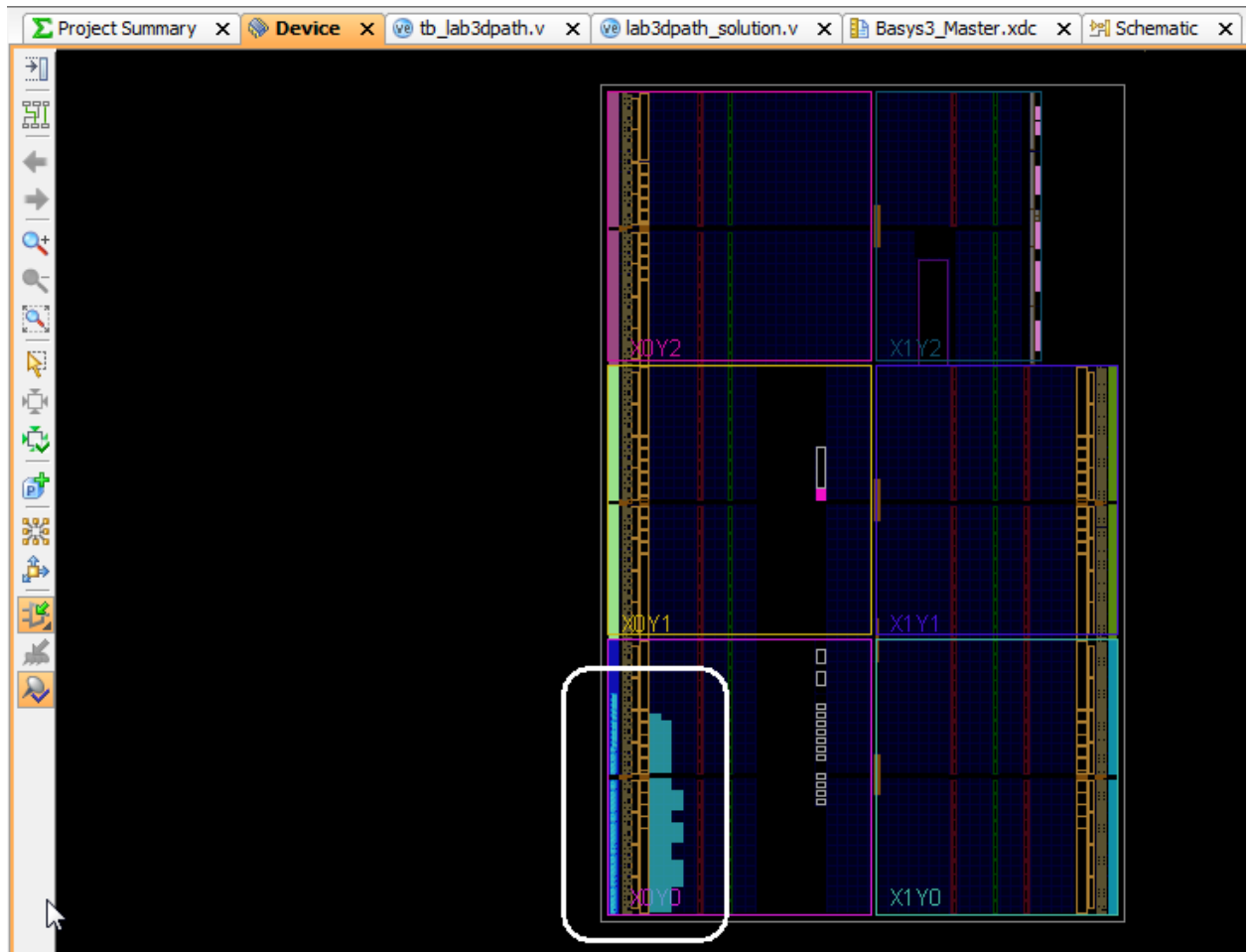
(zoomed out)



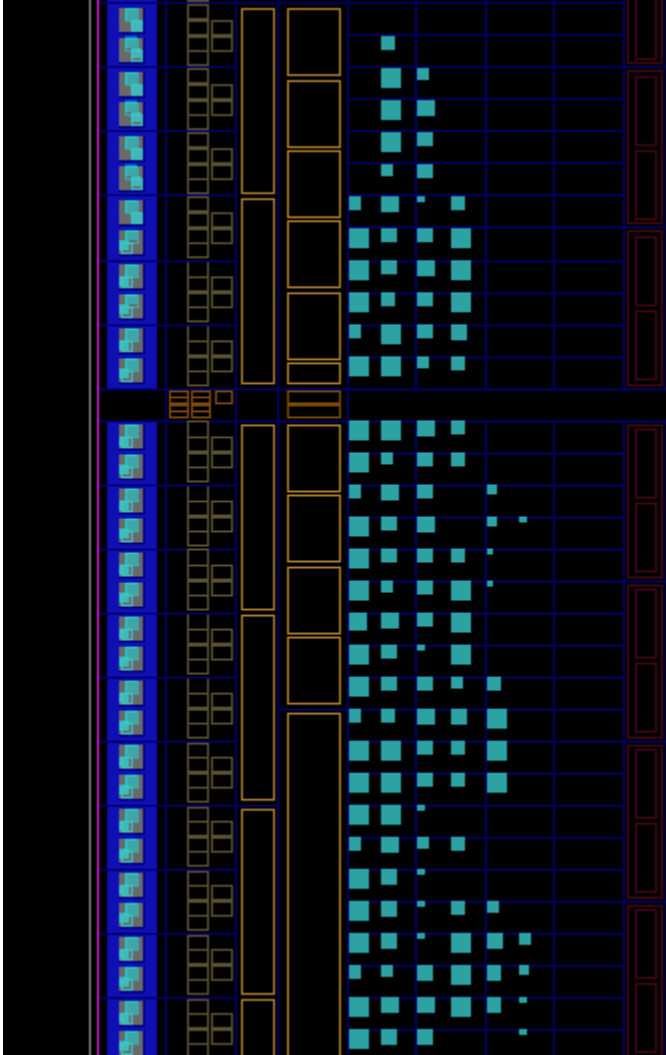


Expand all three multipliers and include your version of the above schematic in your report.

12. Click on the 'Device' tab of the Implemented design. You will see something like what is shown below; this is the 'floorplan' of the logic mapped onto the resources of the actual device. The circled blue part shows the components used by our design.



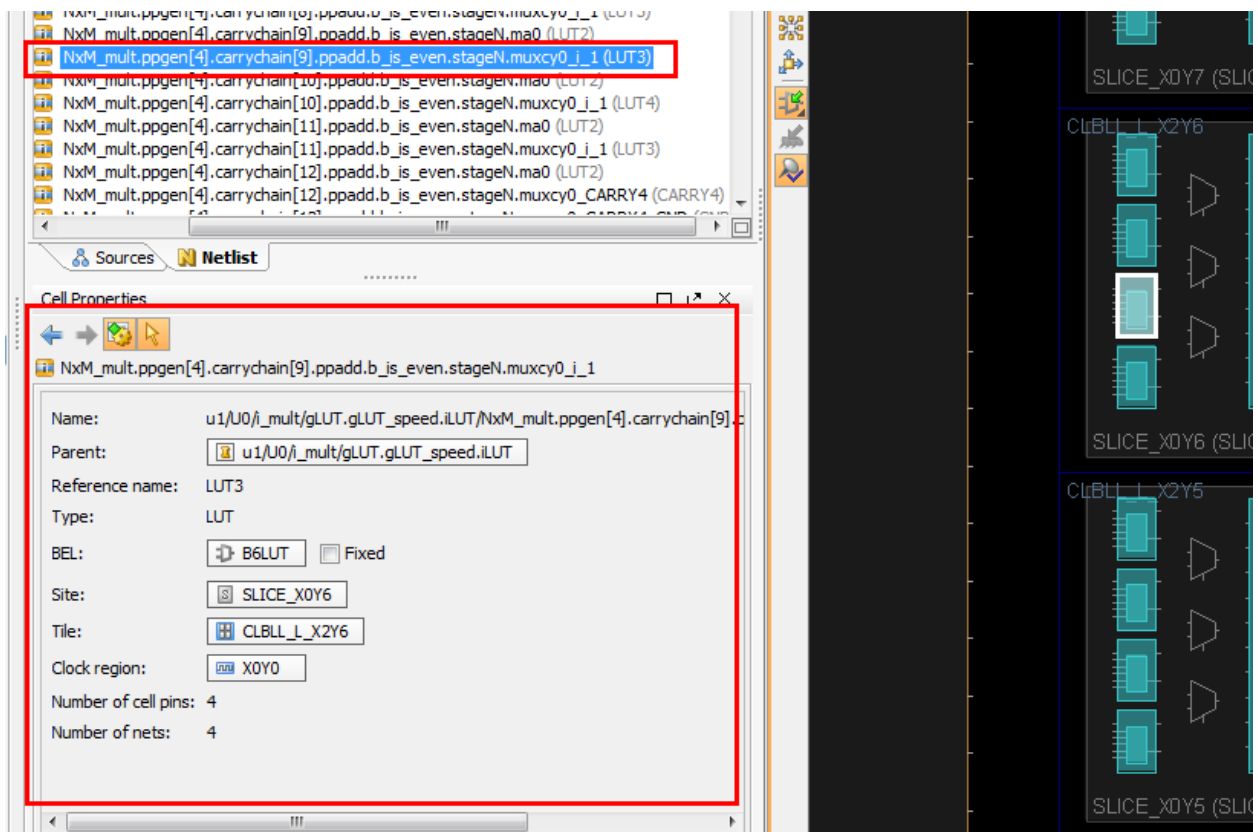
Zoomed in some:



Zoomed in a lot where you can see individual slices and the elements used within those slices.

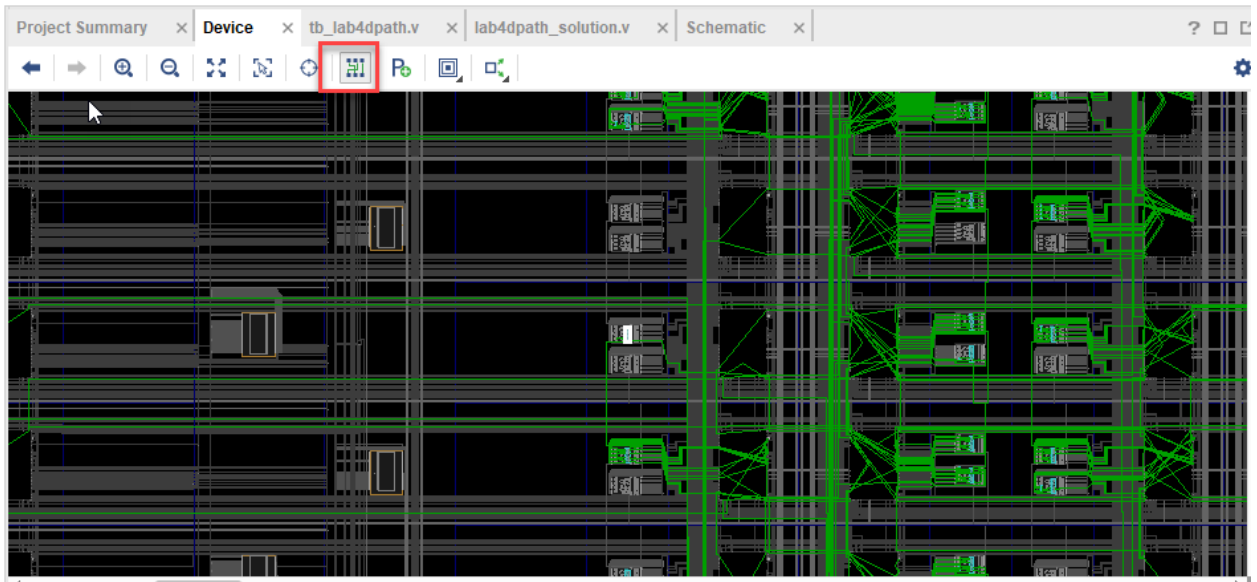


If you click on a lookup table, you can see the information about the LUT in the left hand window:

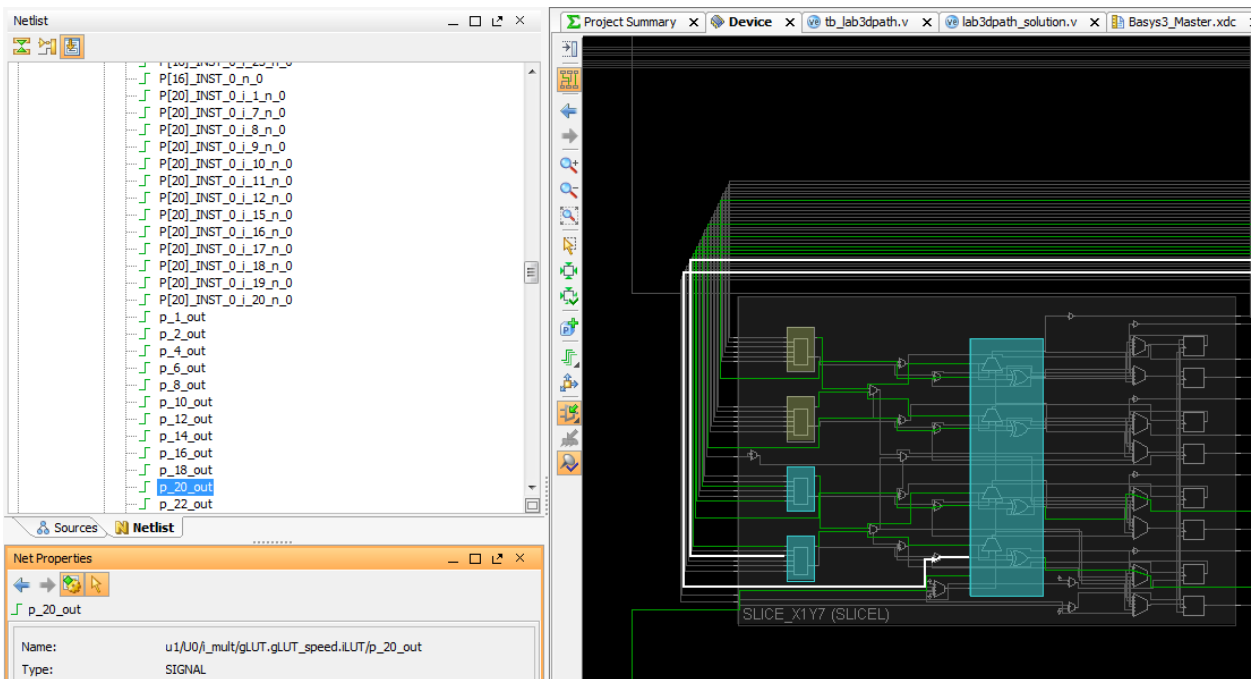


Capture a screenshot similar to the above and put it in your report.

13. You can show routing by clicking on the Show/Hide routing button along the top:



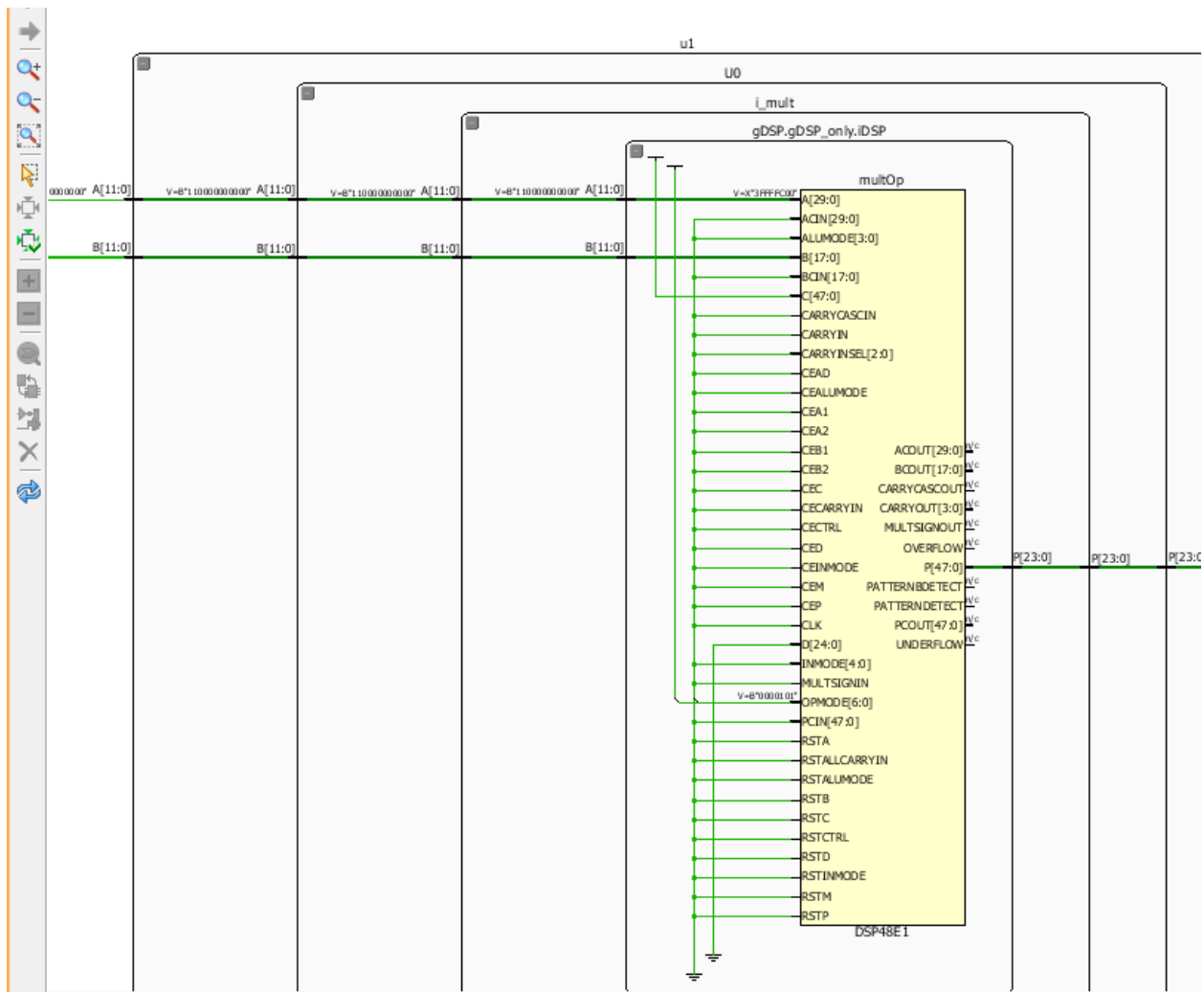
Zooming in, you can see the exact routes:



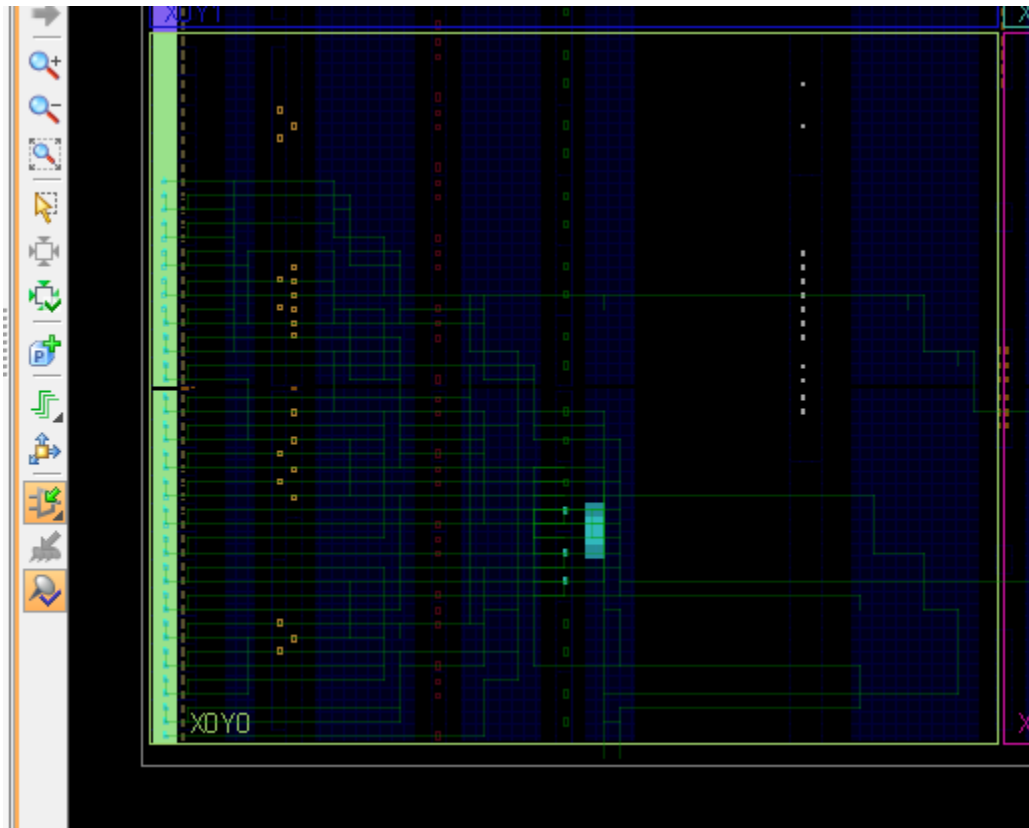
The white net is a selected net – it shows the path through the slice and left-hand pane gives information about the net.

Capture a screenshot similar to the above and put it in your report.

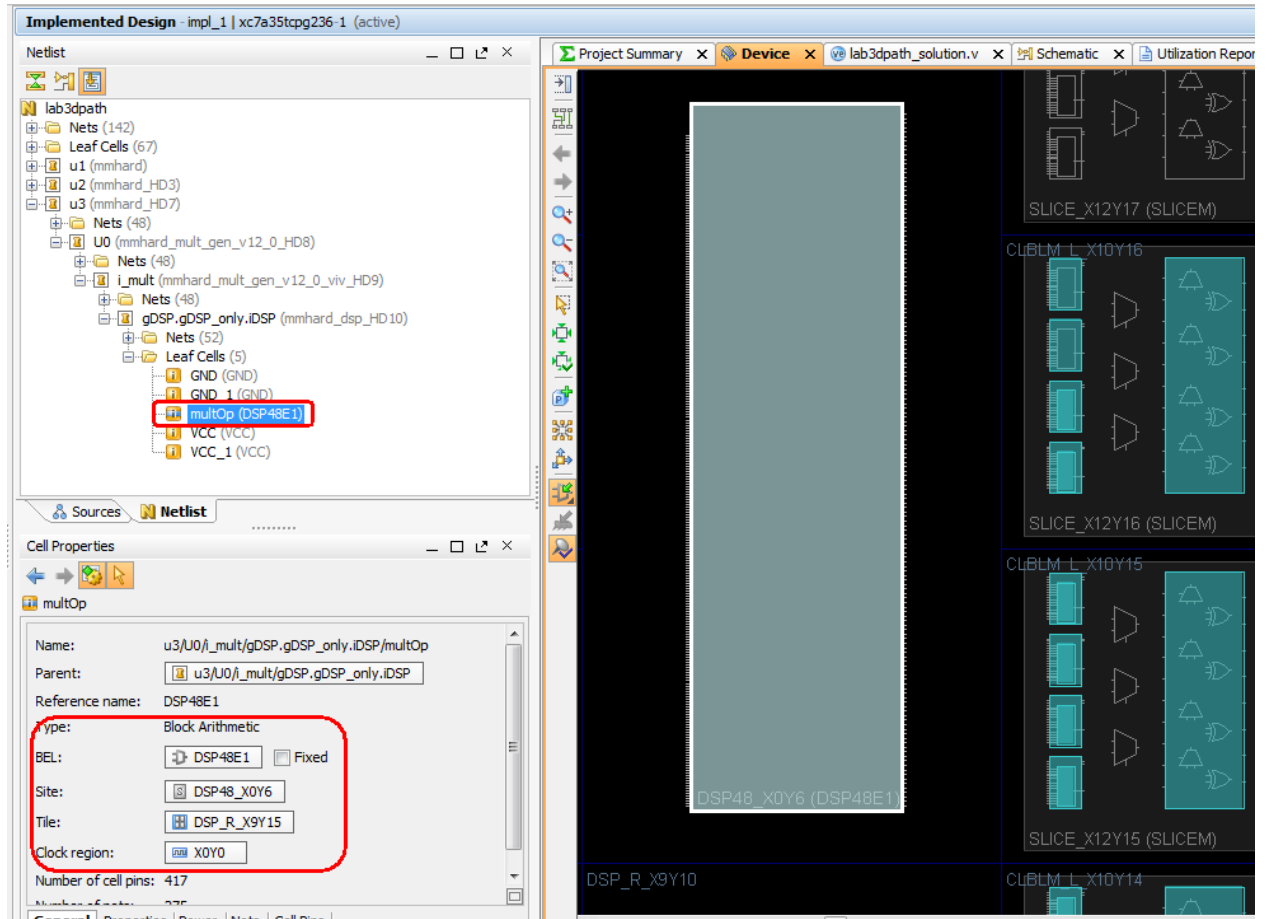
14. Create new project and call it *lab4\_part2*. The only difference between this project and the previous project is that you will use a hardmacro multiplier instead of a LUT-based multiplier (you can copy your solution for *lab4dpath.v* and just modify it to use the hardmacro multiplier module after you generate it). Verify that it passes post implementation timing simulation.
15. Do the same timing analysis, and record the resource usage tables as before. In the Reports, Implementation/Place Design/Utilization Report, also copy/screenshot the table labeled as DSP, as this design should use some DSP blocks (the LUT usage in this implementation will be much lower as the DSP blocks replaces most of the LUTs).
16. Examine the schematic, and expand a multiplier as far you can to show that this was implemented using a DSP block, not LUTs. Include a screenshot similar to that below:



17. In the Device Floorplan, you will find that it looks very different from the previous implementation as DSP blocks are now used.

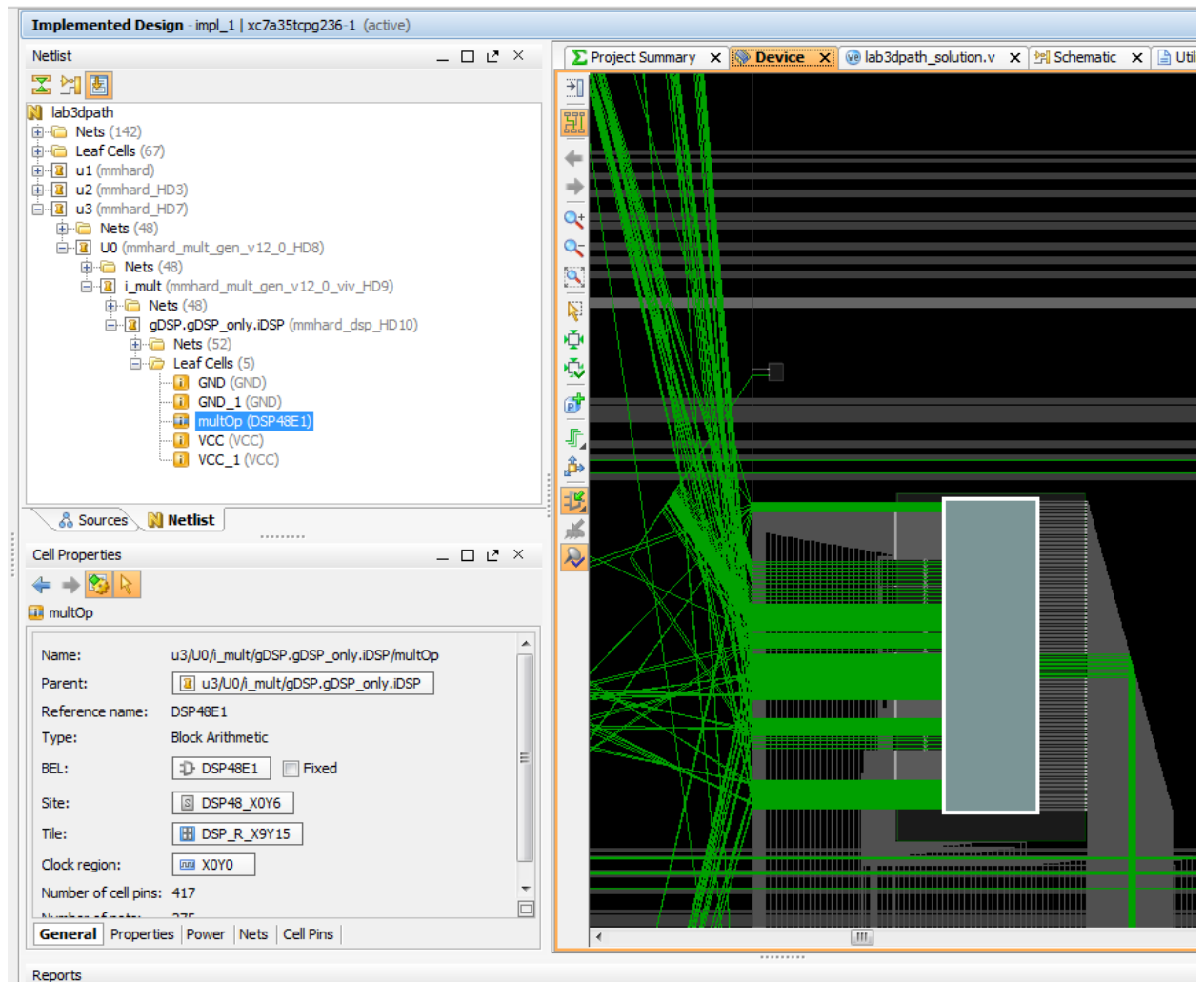


Zoomed in, show DSP block, routing turned off:



Routing turned on:

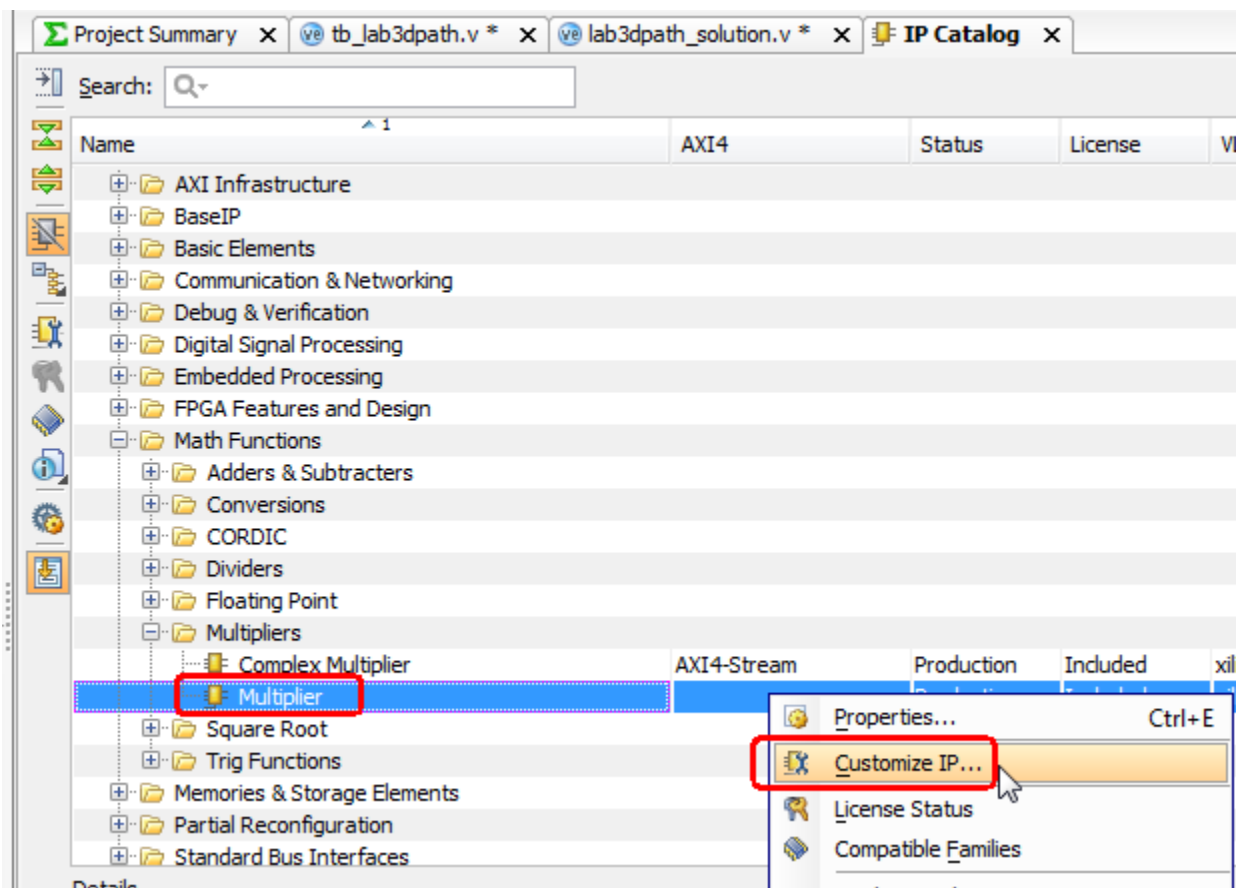




Capture a screenshot similar to that above for the report.

## Multiplier Implementation

The Xilinx IP Catalog tool will be used to generate the multiplier module. See the Lab 2 writeup about how to start up the Xilinx IP Catalog tool. Generate a LUT-based signed 12x12=24 bit multiplier. See the screenshots below for the correct options to generate this.



### Multiplier (12.0)

Documentation IP Location Switch to Defaults

**IP Symbol** Information

☒ Show disabled ports

Component Name: mmlut

**Basic** Output and Control

Multiplier Type

☒ Parallel Multiplier ☐ Constant Coefficient Multiplier

Input Options

$P = A * B$

Data Type: Signed Signed

Width: 12 12

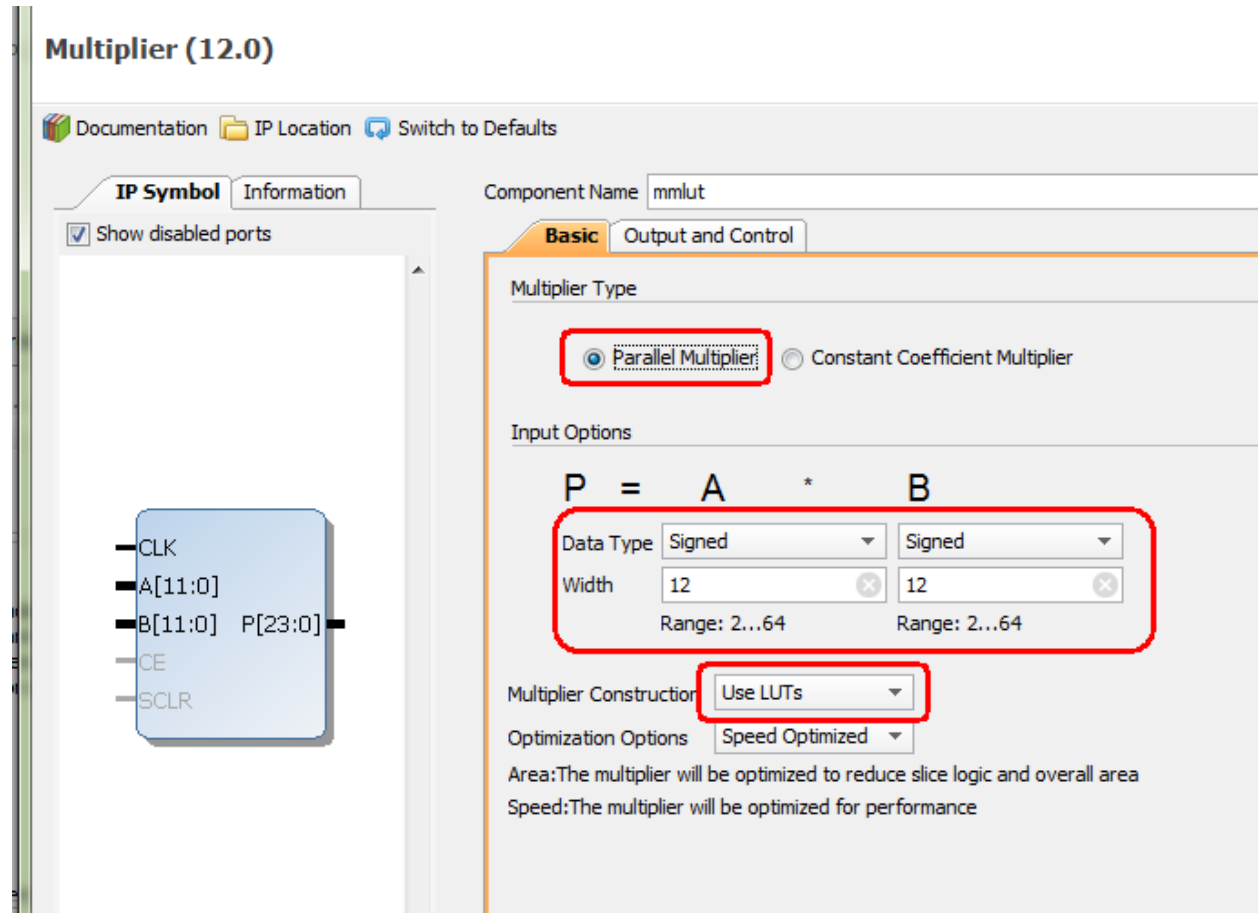
Range: 2...64 Range: 2...64

Multiplier Construction: Use LUTs

Optimization Options: Speed Optimized

Area: The multiplier will be optimized to reduce slice logic and overall area

Speed: The multiplier will be optimized for performance



**Multiplier (12.0)**

Documentation IP Location Switch to Defaults

**IP Symbol** Information

☒ Show disabled ports

CLK  
A[11:0]  
B[11:0] P[23:0]  
CE  
SCLR

Component Name mmlut

**Basic** **Output and Control**

Output Product Range

☐ Use Custom Output Width

Output MSB 23 [0 - 127]  
Output LSB 0 [0 - 23]

Output product width (max, min) = (23,0)

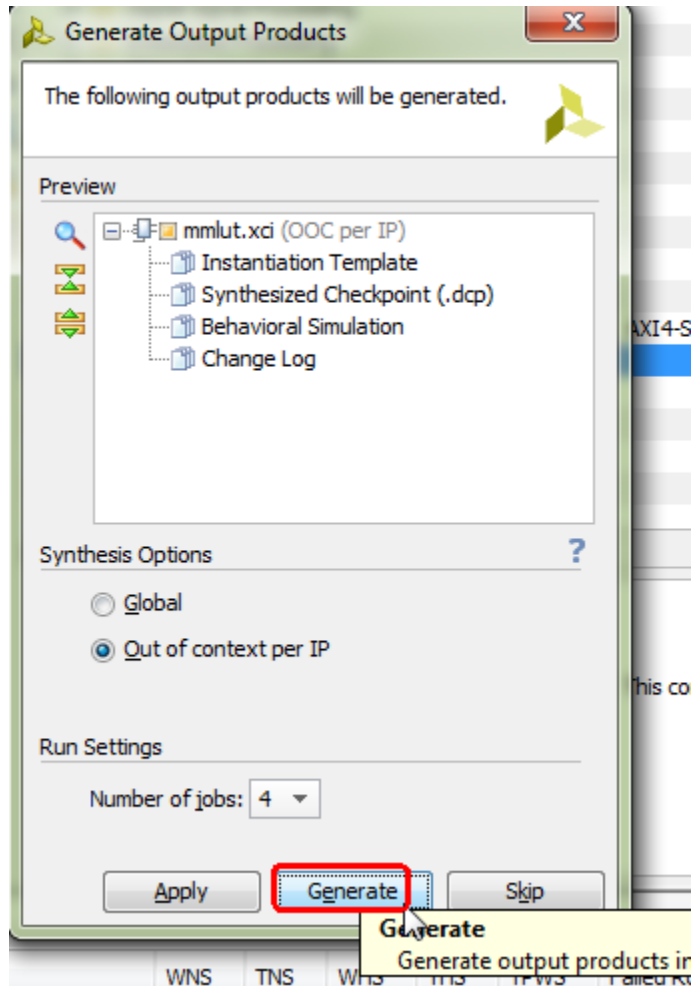
☐ Use Symmetric Rounding

Pipelining and Control Signals

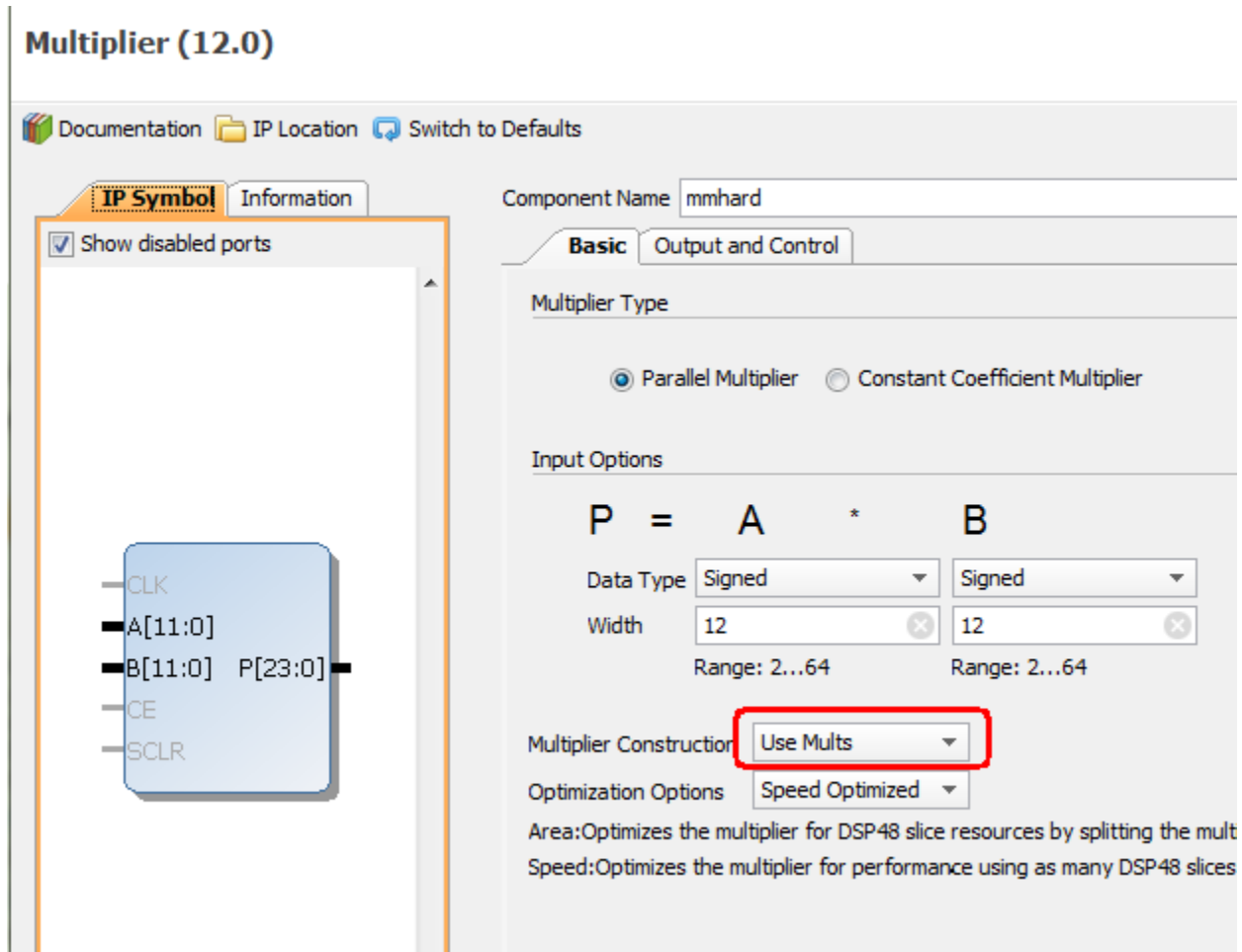
Pipeline Stages 0 Optimum pipeline stages: 4

☐ Clock Enable ☐ Synchronous Clear

Synchronous Controls and Clock Enable(CE) Priority SCLR Overrides CE



To generate Multipliers using DSP blocks, change the 'Multiplier Construction' option from LUTs to MULTs. You will also need to change the name of this IP block.



## GRADING

The grading point distribution is specified in the report document.

## SUBMISSION INSTRUCTIONS

Create a directory named 'lab4\_netid', i.e., (lab4\_rbr5).

Copy the lab4\_part1, lab4\_part2 directories to this directory.

Copy your completed report.doc to this directory.

From Windows, select the 'lab4\_netid' folder, and from the right-click menu, use 'Send To Compressed (zipped) Folder' command to produce a ZIP archive named 'lab4\_netid.zip'.

Upload your 'lab4\_netid.zip' file to Canvas using the Lab4 assignment link.