

**Exercise Manual**  
*for*  
**Quartus II Software Design Series:  
Optimization**

**Software Requirements to Complete All Exercises**

**Software Requirements:** The Quartus II software version 8.0

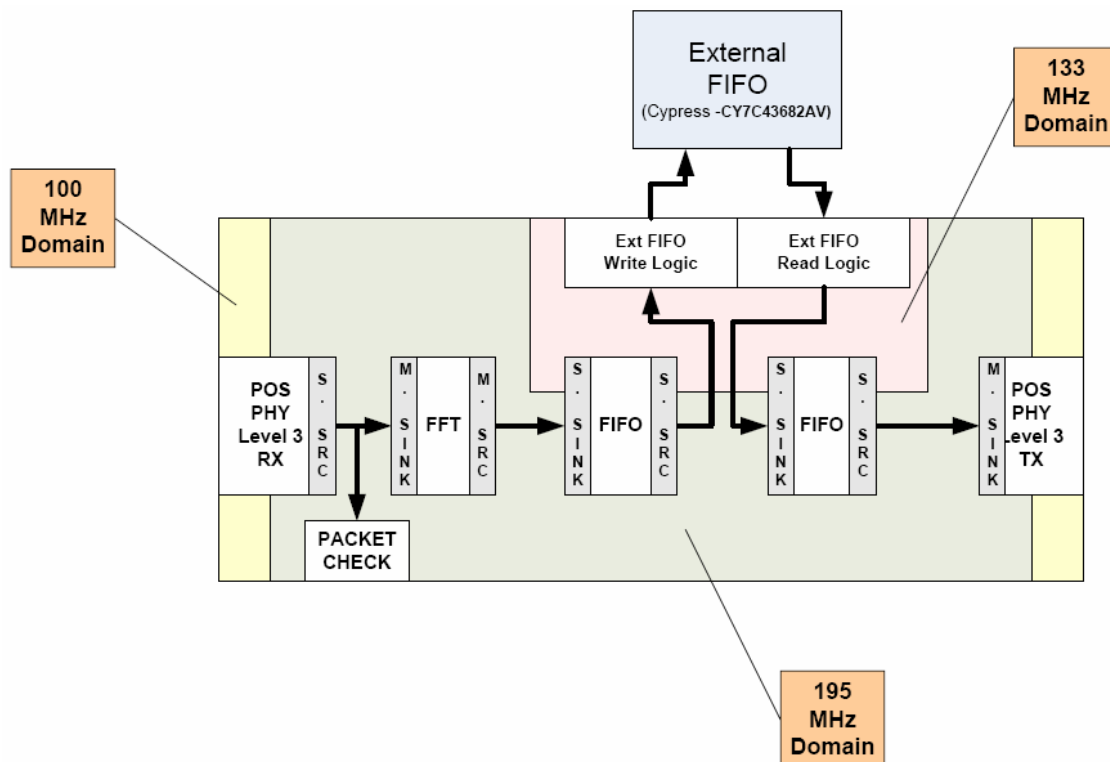
# **Exercise 1:**

## **General Timing Optimization**

## Objectives:

- *Practice optimization techniques*

## Top-Level Design



This networking design, named **lab\_design**, implements a POS-PHY level 3 link between two systems. After being received by the POS-PHY Level 3 receiver block, the “packet check” block analyzes a packet of data to ensure it is correct, e.g. making sure that a packet is 1K words long and that the ERR flag from the receiver is not asserted. The packet then passes through an FFT and a series of FIFOs, with an external FIFO being used for increased storage capability. The packet is finally transmitted by the POS-PHY level 3 transmitter to another device.

There are 3 clock domains in the design, a 100-MHz external domain, a 195-MHz internal clock domain, and a 133-MHz domain to drive the external FIFO.


You will use this design to practice determining and resolving timing issues.

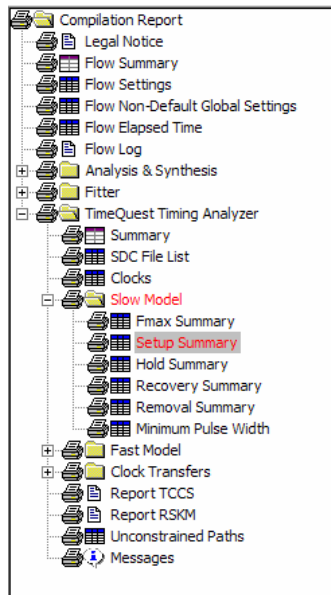
**Step 1 (Open project and compile)**


*First, you will open an existing project and compile. The design has already been fully constrained.*

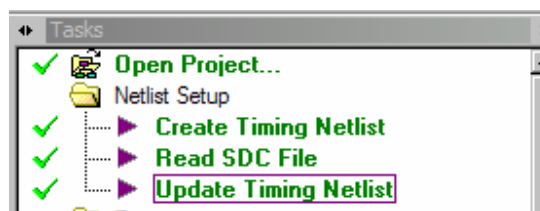
1. Open the project file. **File => Open Project...** and browse to directory `<lab_install_directory>\QII08_0\OPT\`. Select project **lab\_design.qpf** and click **Open**.

*The name of the top-level module in this design is **lab\_design**. The name of the current Quartus II revision is **lab\_design\_tq**.*

2. Compile the design. Click on  button or, from the **Processing** menu, click **Start Compilation**. Click **OK** when complete.
3. Check timing summaries. In the **Compilation Report**, expand the **TimeQuest Timing Analyzer** folder. Click on the **Setup Summary** (appearing in red to indicate there are failing paths) to see that the SCLK domain is failing.

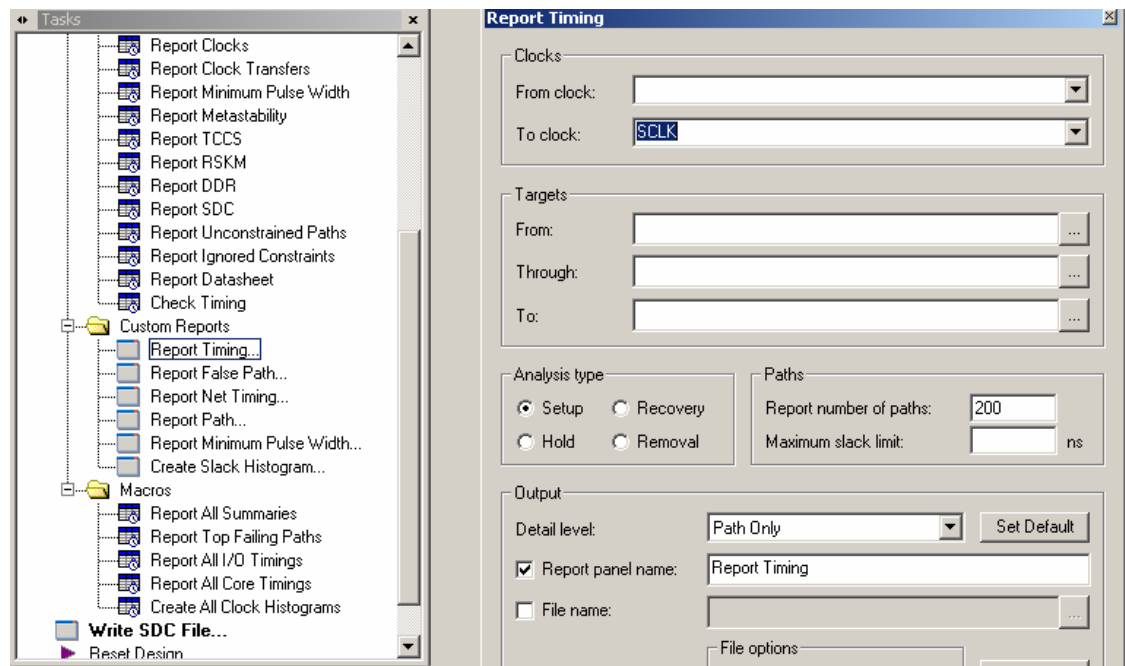
**Step 2 (Run report to Analyze failing paths in TimeQuest to determine cause)**

1. Open TimeQuest. From the **Quartus II Tools** menu, select **TimeQuest Timing Analyzer** or click on the  button in the **Quartus II** menu bar.
2. In timequest, perform commands as shown



3. Run a setup analysis report of the SCLK domain. From the **Tasks** pane in **TimeQuest**, choose **Report Timing**. Edit the following sections and choose **Report Timing**.

To clock	SCLK
Analysis type	Setup
Paths (Report number of paths)	200
Output (Detail level)	Summary



The **Report Timing Summary** report appears. What a sea of RED!!

Many times it is easy to get overwhelmed by a report such as this as it can be very difficult to figure out where you should start investigating. Some tips:

- Look for patterns between **From** and **To** nodes. Grouping the nodes may help understand what's failing, e.g. the interface between two modules or the nodes within a highly combinatorial module. It may also indicate that by fixing one or a few paths, you may be fixing related paths at the same time.
- Look for **From** and **To** nodes that you recognize. If you recognize the module/node names, it may be logic you've written (as opposed to Megafunction modules) which means it might be easier to understand what the logic is and how you may fix the timing failures.
- Fixing one or a few paths may inadvertently fix others. The compiler is trying to optimize many paths at once. If you are able to fix a path by editing the code or changing a constraint, it may free the compiler to work harder on other pieces of the design that were previously failing.

**Step 3 (Fix parity checking logic failures)**

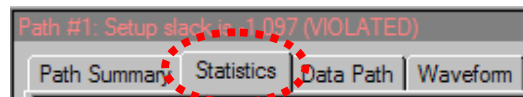
*The first path to fix will be the logic driving the error detection circuit of the parity checker. Click on the "from" column to sort the names. The first four entries are as shown*

Report Timing					
Command Info		Summary of Paths			
	Slack	From Node	To Node	Launch Clock	Latch Clock
1	-1.406	PACKET_CHECK:IPACKET_CHECK last_data[0]	PACKET_CHECK:IPACKET_CHECK parity_error	SCLK	SCLK
2	-1.932	PACKET_CHECK:IPACKET_CHECK last_data[1]	PACKET_CHECK:IPACKET_CHECK parity_error	SCLK	SCLK
3	-1.581	PACKET_CHECK:IPACKET_CHECK last_data[2]	PACKET_CHECK:IPACKET_CHECK parity_error	SCLK	SCLK
4	-1.500	PACKET_CHECK:IPACKET_CHECK last_data[3]	PACKET_CHECK:IPACKET_CHECK parity_error	SCLK	SCLK

1. View a detailed analysis of the failing paths. Right-click on the first line in the **Report Timing Summary** report. Select **Report Worst-Cast Path**.

*A detailed report indicating this path is failing appears.*

2. Check the number of logic levels in the data arrival path. In the **Data Arrival Path** section of the detailed timing report, click on statistics to examine the number of layers



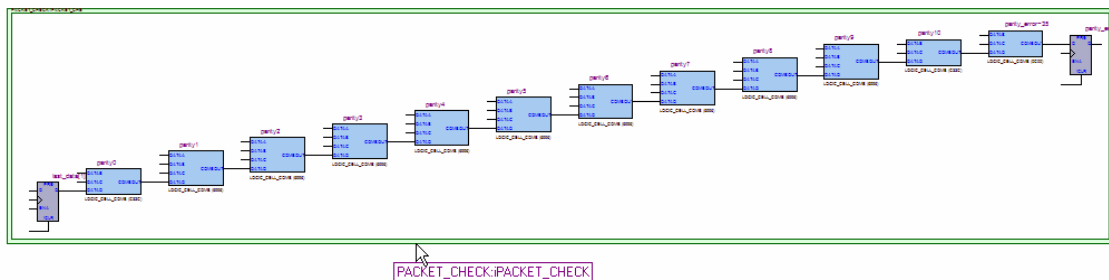
*Is this a lot? It could be. Let's check another way.*

3. View the path in the Technology Map Viewer. Right-click in the **Data Arrival Path** section and choose **Locate Path**. Select **Technology Map Viewer** and click **OK**.

	Total	Incr	RF	Type	Fanout	Lo
1	0.000	0.000				
2	2.813	2.813	R			
3	3.090	0.277		uTco	1	LC
4	3.090	0.000	RR	CELL	1	LC
5	3.090	0.000	RR	IC	1	LC
6	3.448	0.358	RR	CELL	1	LC
7	3.739	0.291	RR	IC	1	LC
8	3.917	0.178	RR	CELL	1	LC
9	4.391	0.474	RR	IC	1	LC

*In the **Technology Map Viewer** graphical display, you can see that there are **12** logic levels between the **last\_data** and **parity\_error** registers. This is good evidence that this is a **Case 1** timing failure: Too many logic levels.*

*Before fixing, let's verify that the fitter did try to perform good placement.*

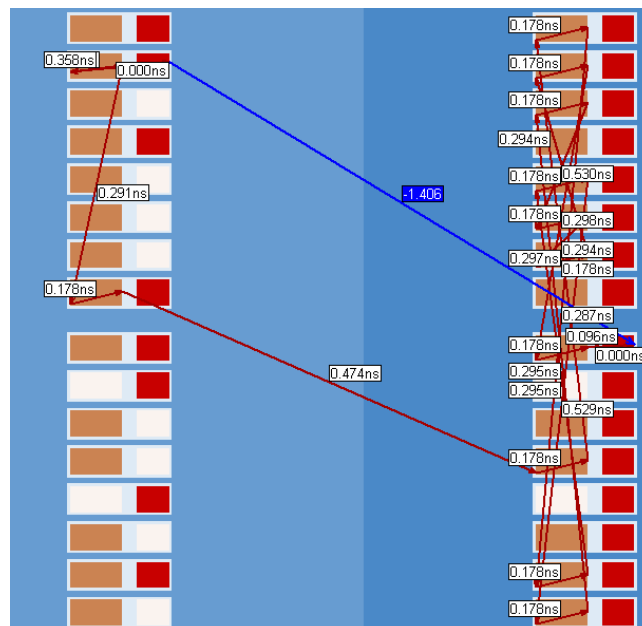


4. Verify the placement in the Chip Planner. In **TimeQuest**, right-click on the **Data Arrival Path** again. Choose **Locate Path**. Choose the default setting of **Chip Planner** and click **OK**.


*You will see that all of the nodes in this path are within two Logic Array Blocks (LABs) of each other, with only a few lines crossing between them. This is very good placement, so this is a good indication that the number of logic levels are the problem.*

*To fix this Case 1 timing path, you will add pipeline registers. This is a valid solution because, understanding the design, you know that when a parity error occurs, it is not critical that this error be caught immediately. A lag of a few cycles can easily be tolerated by the logic.*

*The code for the corrected design has already been written for you. This code involves adding multiple stages through the parity checking logic*

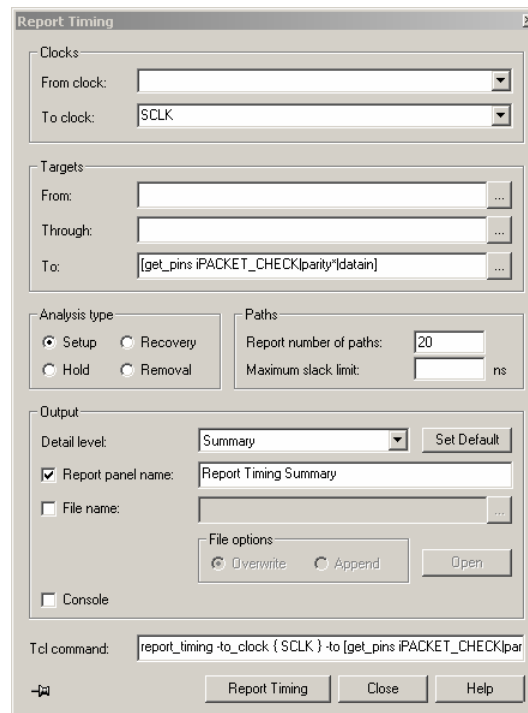


5. Replace the packet\_check.v file with an updated solution. In **Quartus II**, go to the **Project** menu and select **Add/Remove Files in Project**. Highlight and delete **src/packet\_check.v**. Click on the browse button and choose **src/packet\_check\_soln.v**. Click **Open** and then **Add**. Click **OK** to close the **Settings** dialog box.

6. Compile the design. Click on  button
7. Open TimeQuest.
8. Run your setup report to verify the parity checking logic. From the **Tasks** pane in **TimeQuest**, choose **Report Timing**. Edit the following sections and choose **Report Timing** (type the **To** target in the field directly instead of using the Name Finder; case does matter. Just copy and paste from below).

To clock	SCLK
To	[get_pins iPACKET_CHECK parity* datain]
Analysis type	Setup
Paths (Report number of paths)	20
Output (Detail level)	Summary

*From this report you can see that all parity checking registers are now meeting timing.*



### Step 3 (Fix remaining timing failures)

*Now, you will analyze and fix the remaining timing failures in the **SCLK** domain.*

1. Re-run a setup analysis report of the SCLK domain. From the **Tasks** pane in **TimeQuest**, choose **Report Timing**. Edit the following sections and choose **Report Timing**.

To clock	SCLK
----------	------



Analysis type	Setup
Paths (Report number of paths)	200
Output (Detail level)	Summary

Again, this is done so that you can recognize patterns between the failing nodes. In the **Report Timing Summary** table. Click on the **To Node** column to sort out the names. Many of the first failing paths are all to the same **To node**, the **sop\_error** (start of packet error flag) signal, coming from the FIFO address registers in the receiver block. This means that there is a chance they could all be fixed at once.

- Analyze the first failing **sop\_error** path in more detail. Right-click on the first path in the **Report Timing Summary** and select **Report Worst-Case Path**.
- Check the number of logic levels in the data arrival path. In the **Data Arrival Path** section of the detailed timing report, click on statistics and examine the number of cell count

*This time, there are few levels, so it appears the timing failures are probably not related to the number of logic levels. But, let's verify this graphically.*

- View the path in the Technology Map Viewer. Right-click in the **Data Arrival Path** section and choose **Locate Path**. Select **Technology Map Viewer** and click **OK**.

*Unlike before, this path only shows a RAM block and 3 levels of logic, so it appears NOT to be a **Case 1** timing failure. The path out of the RAM block is combinatorial. This means the total register-to-register delay includes the RAM block AND 3 logic cells. Does this make a difference?*

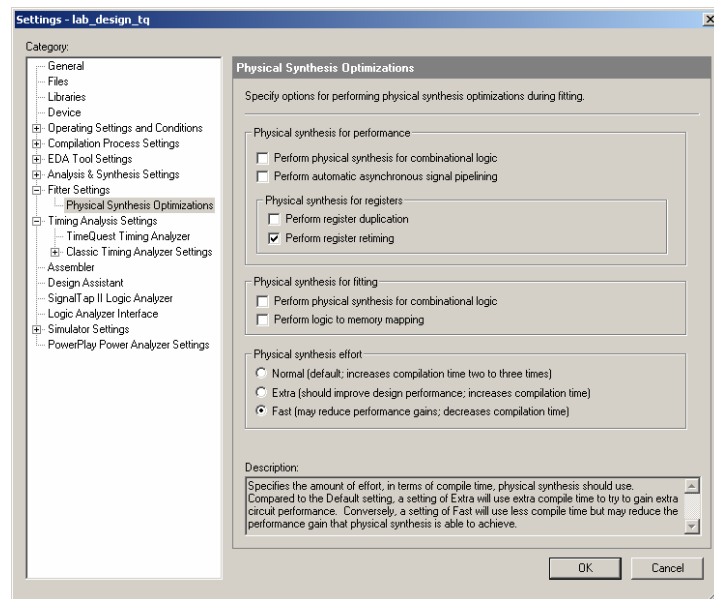
- Review the detailed path report again. Go back to the detailed slack report in **TimeQuest**. In the **Data Arrival Path** section, go to **line #4**.


*Notice in the **Type** column that this is cell delay. That means that the incremental value incurred by this line is the delay that passes through a cell of the device. The **Location** and **Element** columns indicate that the cell is actually an **M4K memory block**. How much delay does the M4K add? \_\_\_\_\_ (Look at the **Incr** column.)*

*Thus, even though there are only a few cells in the path, this is still a **Case 1** timing failure. So how can you fix? Well, manually pipelining the block (e.g. by using the M4K output registers) might be impossible since this RAM is part of the POS-PHY MegaCore function. A multicycle assignment would fix the analysis, but should be used if the logic was designed to work in this way.*

*In this case, you can use register re-timing, under physical synthesis. Though this WILL increase compile time, it might also serve to fix the other timing errors being incurred by the design.*

- Turn on physical synthesis. From the **Assignments** menu, choose **Settings**. Choose the **Physical Synthesis Optimizations** category under **Fitter Settings**. Enable **Perform register retiming**. Set the **Physical synthesis effort** to **Fast**. Click **OK**.



7. Compile the design. Click on  button or, from the **Processing** menu, click **Start Compilation**. Click **OK** when complete.

*From the **TimeQuest Timing Analyzer** section of the **Compilation Report**, you can see that the **Setup Summary** is finally clean.*

### Exercise Summary

- Use Quartus II tools to analyze failing timing paths to determine their causes
- Use techniques discussed in presentation to eliminate failing paths

## End of exercise 1

## **Exercise 2:**

# **Timing Optimization using PLLs**

**Objectives:**


- *Practice optimization techniques*
- *Use PLLs to improve timing optimization results*

**Top-Level Design**

In this exercise, you will take a design implementing a two-dimensional discrete cosine transform (DCT) and review why it is failing timing. You will then use optimization techniques including incorporating a PLL to improve design performance.


**Step 1 (Open project and compile)**

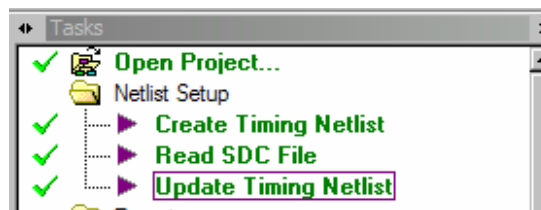
*First, you will open an existing project and compile. This project has already been set up to use TimeQuest and an SDC file to control fitting. The device I/O have already been locked down so their moving around does not affect device timing.*

1. Open the project file. **File => Open Project...** and browse to directory `<lab_install_directory>\QII08_0\PLL_OPT\`. Select project **two\_d\_dct.qpf** and click **Open**.
2. Compile the design. Click on  button
3. Check timing summaries. In the **Compilation Report**, expand the **TimeQuest Timing Analyzer** folder. You should see only the **Setup Summary** appearing in red to indicate it is failing. What is your failing setup slack value? \_\_\_\_\_

**Step 2 (Analyze failing path in TimeQuest)**

*Use the reporting capabilities of TimeQuest to analyze why you are having setup slack failures.*

1. Open TimeQuest. From the **Quartus II Tools** menu, select **TimeQuest Timing Analyzer** or choose the  toolbar button.
2. In timequest, perform commands as shown



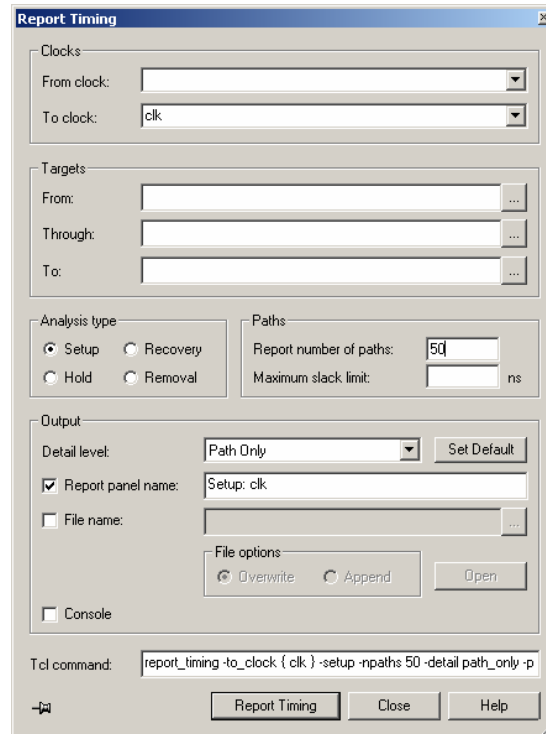
3. Run the setup slack summary report. In the **Tasks** pane, double-click on **Report Setup Summary**.

*As in the Quartus II Compilation Report, the Summary (Setup) report should show that clk is failing by -1.424 ns.*

Summary (Setup)			
	Clock	Slack	End Point TNS
1	clk	-1.424	-28.375

4. Review more detailed information on the failing clock domain. In the **Summary**

(**Setup**) report, right-click on **row 1** and choose **Report Timing**. In the **Report Timing** dialog box, change the **Report number of paths (Paths section)** to **50** and use the **Detail level:** drop-down menu (**Output section**) to select **Summary**. Click on the **Report Timing** button.



*Why do this? This report is generated in order to determine how many paths are actually failing (e.g. a few paths versus hundreds of paths) and if any relationships can be found amongst the failing paths. Can you notice any such relationships?*

- *You should notice that the top 23 failing paths are all between the design outputs (**dct\_out** and **data\_valid**) and the registers driving them, and that they are all failing by over 1 ns. You should analyze more closely to see if we are getting the best placement for the output registers.*

5. Analyze the detailed timing on a single output failing path. In the **Setup: clk Summary** table, right-click on the first failing path (**data\_valid**). Choose **Report Worst-Case Path**.

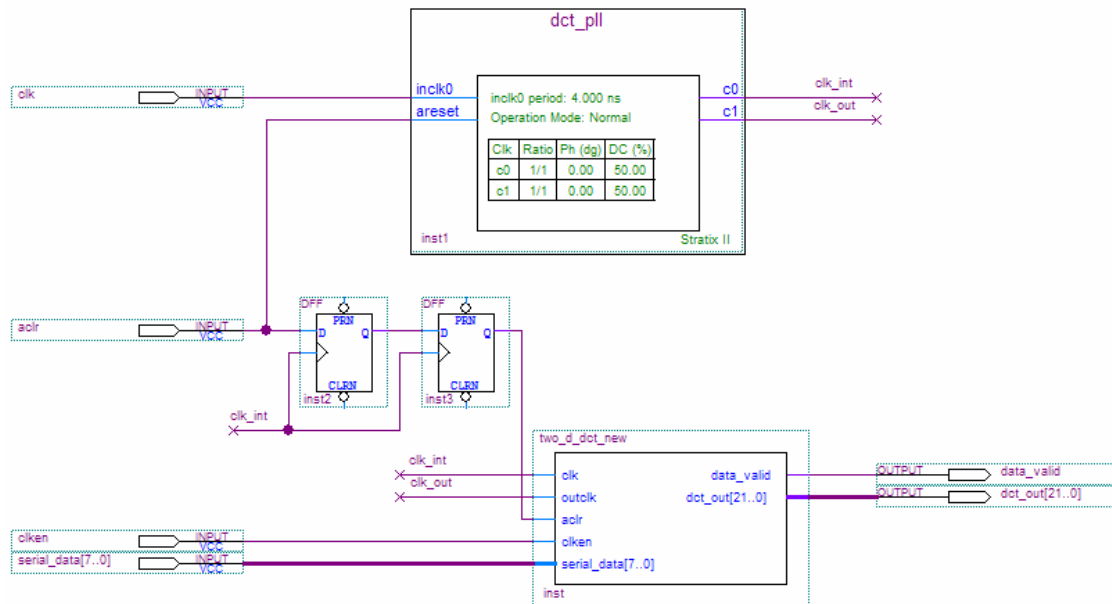
*A detailed slack analysis report appears for Path #1. You will now try to discover which of the typical timing failures you are dealing with.*

6. Is this a **Case 1** timing failure, too many logic levels? Look at the data arrival path for Path #1. How many target nodes does this path go through? So no, this is not a Case 1 timing failure.

7. Is this a **Case 2** timing failure, high fan-out signals? Look at the **Fanout** column in the data arrival path for Path #1. Are there any numbers that might signify fan-out issues? Since there are none, this is not a Case 2 timing failure.
8. Is this a **Case 3** timing failure, conflicting physical assignments? The only physical constraints that have been applied to this design are device I/O assignments, so that already is a hint that this might not be the issue. To further verify, look at **row 3** of the data arrival path (**Location** column). You should see that the data\_valid register (**data\_valid~reg0**) has been placed in **IOC\_X0\_Y20\_N1** (an I/O cell block). This means that this register has already been placed as close to the device I/O as possible (for the best output setup timing). Thus, this is not a Case 3 timing failure.
9. Is this a **Case 4** timing failure, conflicting timing assignments? Since you didn't create this design or build its SDC file, this one is a little more difficult. But, if you open the SDC file for the project, you will see that the design simply has the clock and I/O constrained. Plus, if there was an internal constraint interfering with this I/O register placement (like a clock setting), it would more than likely be trying to pull the register OUT of the I/O cell.
10. Finally, is this a **Case 5** timing failure, tight timing requirements? Since you have ruled out all the other failures and you verified that the output register has already been placed in the best possible location for output setup timing, Case 5 is probably the problem.

*So what are the solutions? According to the possible solutions from the presentation material, you could add multicycle assignments or go back and re-evaluate your timing constraints to verify if maybe you are over-constraining the I/O (i.e. your output maximum delay is too large). The solution you are going to employ in this exercise is to use a PLL to shift the output clocks.*

### Step 3 (Add PLL and synchronization register and analyzer timing)



A new top-level design file (**top\_dct.bdf**) has been created for you. It incorporates a PLL named **dct\_pll** as well as adds some synchronizations registers to your asynchronous input path.

1. Create a new revision for the design changes. From the **Project** menu of **Quartus II**, choose **Revisions**. In the **Revisions** dialog box, click on the **Create** button. Name the new revision **top\_dct**, based it on the **two\_d\_dct** revision. Click **OK** to close the **Create Revision** dialog box and **OK** again to close the **Revisions** dialog box.
2. Open the file **top\_dct.bdf**. From the **Project** menu, choose **Set as Top-Level Entity** (near the bottom).


Notice the top-level entity name shown in the **Project Navigator** is now **top\_dct**. Take a second and look over the design file. Notice that **two\_d\_dct** (now called **two\_d\_dct\_new**) is now a module in the design and that a new input has been added to it called **outclk**. This outclk signal fed by the PLL allows you to adjust the phase of the clock driving the output registers independently of the internal clock.

All of the top-level I/O names remained the same.

A new SDC file, named **top\_dct.sdc**, has also been created for you that incorporates the PLL settings.

3. Specify **top\_dct.sdc** as the SDC file for this revision. Go the **Assignments** menu and choose **Timing Analysis Settings**. Select the **TimeQuest Timing Analyzer** category. Remove **two\_d\_dct.sdc** as the SDC and replace with **top\_dct.sdc**. Click **OK** to close the **Settings** dialog box.

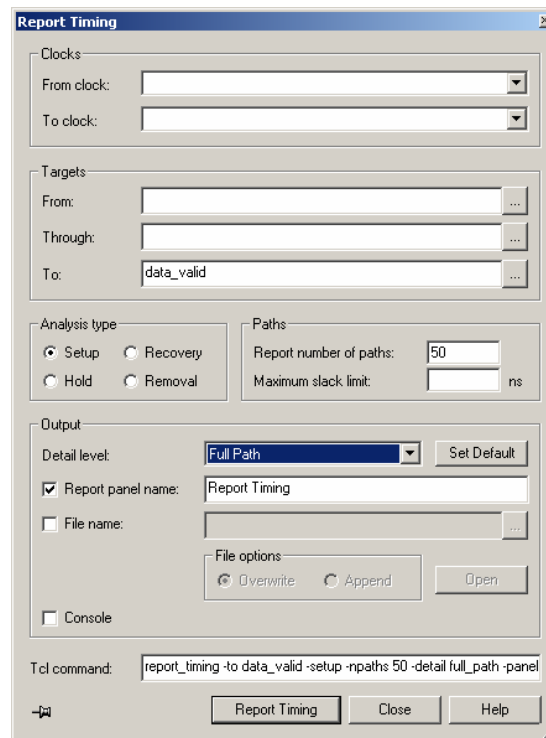


4. Compile the design. Click on  button, or from the **Processing** menu, click **Start Compilation**. Click **OK** when complete.
5. Check timing summaries. In the **Compilation Report**, expand the **TimeQuest Timing Analyzer** folder. The **Setup Summary** is no longer in red indicating that the problems have been fixed with the introduction of the PLL

*What happened to the failing paths on all of the output pins (now driven by **clk\_out**? One of the functions of the PLL is to effectively remove clock tree delay, thus your clock-to-output times decreased and now you can meet your output delay requirements..*

6. Verify the above information regarding the output registers. Open **TimeQuest**. In the **Tasks** pane, double click on **Report Timing**. In the **To:** field (**Targets** section), type **data\_valid**. In the **Detail level:** field (**Output** section), use the drop-down menu to select **Full Path**. Click **Report Timing**.

*This time, you will see full path for the clock expanded. On line 6 of the **Data Arrival Path**, you can see that the PLL inserts a **-5.049 ns** phase compensation offset to account for the global clock tree delay. This reduces the data arrival time enough for you to meet your setup time requirement for the outputs.*



*Remember that using the PLLs is one of many ways to optimize timing on a design. Whether you can use this method or not depends on other factors like the availability of PLLs, jitter tolerances, and even power consumption. For each design you try to optimize, you have to choose the best methods based on your knowledge of the design*

*and what is easiest to implement.*

### Exercise Summary

- Practiced reviewing a design to determine possible causes for failing paths
- Used PLLs to optimize timing in a design

## End of Exercise 2