

**Exercise Manual**  
*for*  
**Quartus<sup>®</sup> II Software Design Series:**  
**Timing Analysis**

Software requirements to complete all exercises

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



# Exercise 1

## Introduction to the TimeQuest Tool




## Exercise 1

### Objective:

- *Given an existing SDC file, follow the TimeQuest flow to generate timing reports*
- *Learn about using the TimeQuest interface*


**Step 1: Open and synthesize a project**

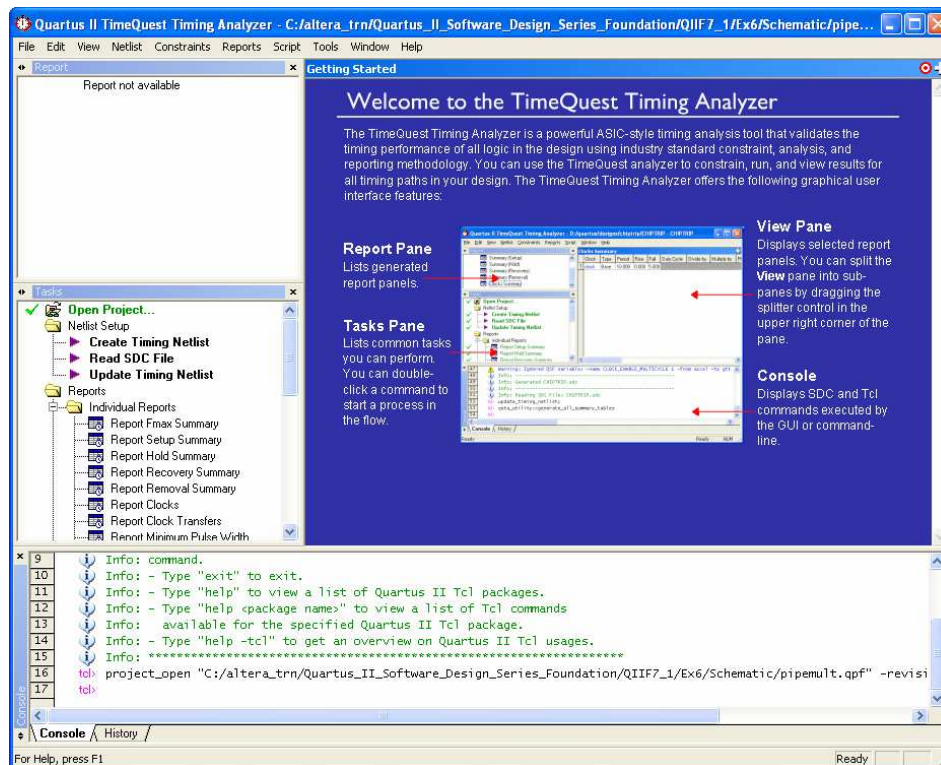
*To get started with using TimeQuest timing analysis, you will open a simple project and configure it to use an existing SDC file with the TimeQuest timing analyzer.*

1. Unzip the lab project files, if necessary. In an Explorer window, go to **C:\altera\_trn\Quartus\_II\_Software\_Design\_Series\_Timing**. The name of the directory may be shortened to **Quartus II Timing** or something similar on some machines. Double-click the executable file (**Quartus II Design Series\_Timing\_8\_0\_v1.exe**) found in that location, or on the CD included with this lab manual. If you cannot find this file, ask your instructor for assistance. In the WinZip dialog box, click **Unzip** to automatically extract the files to a folder named **QIIT8\_0** in the above directory. Close WinZip.
2. Start the Quartus II software version 8.0 from the Altera program folder in the Windows Start menu.
3. **Open** the project **pipemult.qpf** located in the **<lab\_install\_directory>\QIIV8\_0\Intro** directory. Remember to use the **Open Project** command from the **File** menu instead of the **Open** command (which is used for opening individual files instead of entire projects).
4. Click  to synthesize the design.

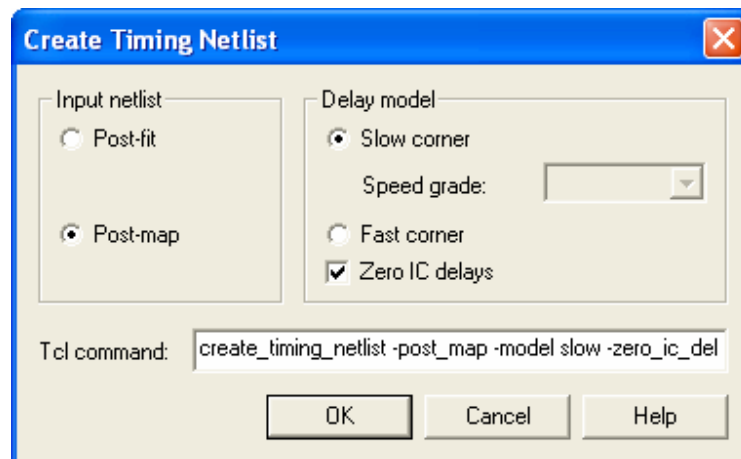
*Though you could also perform a full compilation since this design is complete, we're going to follow the TimeQuest flow as if we were working on a new design that requires a long place and route. Performing synthesis allows you to quickly generate a netlist in order to start constraining the design.*

**Step 2: Start the TimeQuest GUI and create timing netlist for analysis**

1. From the main Quartus II toolbar, click  or, from the **Tools** menu, select **TimeQuest Timing Analyzer**. You can also access the TimeQuest interface through the **Compile Design** category in the **Tasks** window. Click **No** that you do not want to generate a .SDC file from the .QSF file. If you accidentally click **Yes**, close the TimeQuest interface, open an Explorer window, and delete the **pipemult\_lc\_phys\_syn.sdc** file that got created in your project directory. Return to the Quartus II software and open the TimeQuest tool again.



The window above opens. You will now go through the steps to use **TimeQuest** timing analysis to constrain a design and verify timing.



2. Create a timing netlist. From the **Netlist** menu, select **Create Timing Netlist** and change the **Input netlist** type to **Post-map** OR in the **Console** pane, type `create_timing_netlist -post_map`. Click **OK**.

*You could have chosen to create a timing netlist for a fast corner device in this dialog box or by using the **Set Operating Conditions** command from the **Netlist** menu (after creating a default slow or fast corner model). This project uses only a standard Cyclone® II device, so there is no third timing model (slow, 0°C model normally needed for 65 nm and smaller devices) and no industrial or military grade models.*

*A green checkmark appears next to **Create Timing Netlist** in the **Tasks** pane to indicate the command was successful. Notice there is a message in the **Console** pane (in blue) indicating that the TimeQuest timing analyzer is not the default timing analysis tool. You will correct that later in the exercise. You could have double-clicked **Create Timing Netlist** in the **Tasks** pane. However, that manner of creating the netlist would use the default setting of creating a post-fit netlist, which would not work since the Fitter has not run yet.*

3. Read in an SDC file. Simply type `read_sdc` at the `tcl>` prompt in the **Console** pane or double-click **Read SDC** in the **Tasks** pane.

*A message appears in the Console pane indicating that an SDC file for the current revision of the project has not been found. This is correct. Since you did not specify a filename, the tool automatically looked for any SDC files that were manually added to the project and then looked for an SDC file in the project directory sharing the same name as the current revision `pipemult_lc_phys_syn`, neither of which exists. There is an SDC file in the project directory that we'll use in the next step, but its name does not match the current revision.*

4. Examine and read in an existing SDC file. From the TimeQuest **File** menu, select **Open SDC File**. Select and open the `pipemult.sdc` file found in the project directory.

*The SDC file opens in an SDC file editor window. Remember that the SDC file editor is the same as the Quartus II text editor, so you can create and edit SDC files without opening the TimeQuest interface.*



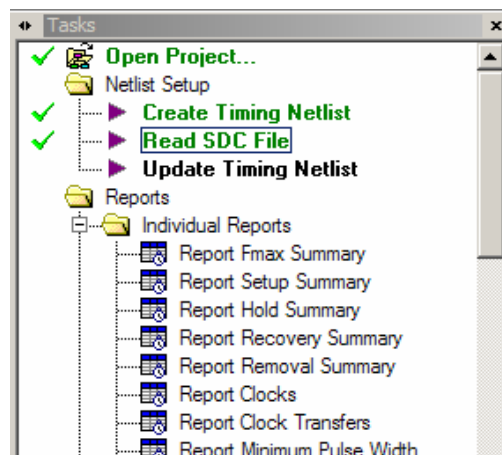
Examine the constraints found in the **pipemult.sdc** file. These constraints constrain the input clock and the I/O signals in the design. These constraints fully constrain this simple design. We'll talk much more about constraints and creating them later.

5. Close the SDC file. From the **Netlist** menu in the main TimeQuest window, select **Delete Timing Netlist**.

Since we tried to read in a non-existent SDC file, we have to create a new timing netlist by first deleting the old one.

6. Again from the **Netlist** menu, create a new, post-map netlist.
7. From the **Constraints** menu, select **Read SDC File** and select the **pipemult.sdc** file. The file is read in as the file to use for constraining the design.

There should now be a green checkmark next to **Read SDC File** indicating you read in an SDC file.



Your **Tasks** pane should look like the screenshot above at this point.

8. Update the timing netlist. In the **Tasks** pane of the TimeQuest GUI, double-click **Update Timing Netlist**.

Once the SDC file is added to the project, you can skip **Read SDC File** and just update the timing netlist to automatically read in the file and update the netlist in one step. Or you can even skip both steps and start creating timing reports. The TimeQuest tool will perform all the intermediate actions required. Of course, you could also create a simple Tcl script with all the appropriate commands and run it from the TimeQuest **Script** menu.

### Step 3: Use TimeQuest reports to verify design meeting timing

Now that the netlist has been updated, you can begin generating various reports. We'll look at reports in more detail later. For now, experiment and explore the different reports you can create.

1. In the **Tasks** pane, double-click **Report SDC**.

In the **Report** pane, a new folder called **SDC Assignments** appears containing three reports called **Create Clock**, **Set Input Delay**, and **Set Output Delay**. This report lists all the SDC constraints entered by the SDC file.

2. In the **Tasks** pane, double-click **Report Clocks**.

*This report verifies that clocks in the design are constrained correctly.*

3. In the **Tasks** pane, double-click **Report Setup Summary**.

*This design meets setup timing.*

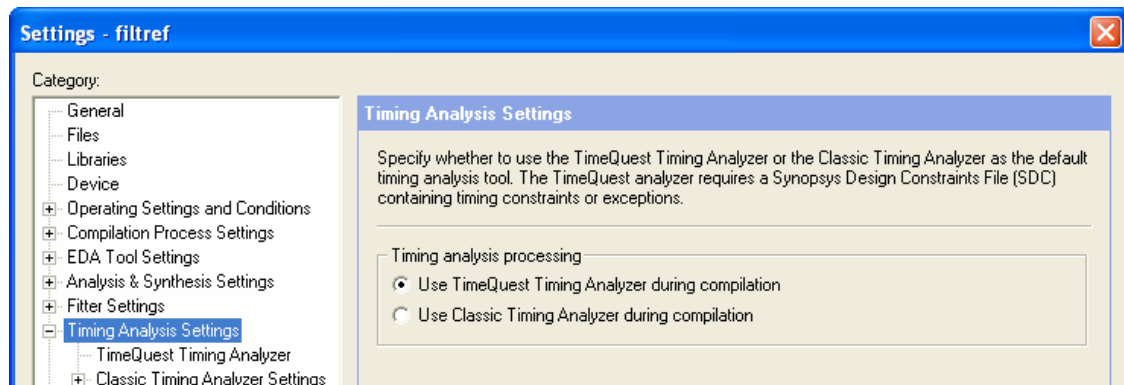
4. In the **Tasks** pane, double-click **Report Hold Summary**.

*Uh oh. It looks like the design, even with the timing constraints added, just fails hold timing, indicated by the clock domain in red and a negative slack value. This is not too big a deal since we're looking at the slow corner model (remember that hold timing must be met in the fast corner model), but it still needs to be fixed. This failure will get fixed by using physical synthesis in the project to optimize the design's placement and routing. We'll see it fixed later when we create the same report using the post-fit netlist. For now, ignore the failure.*

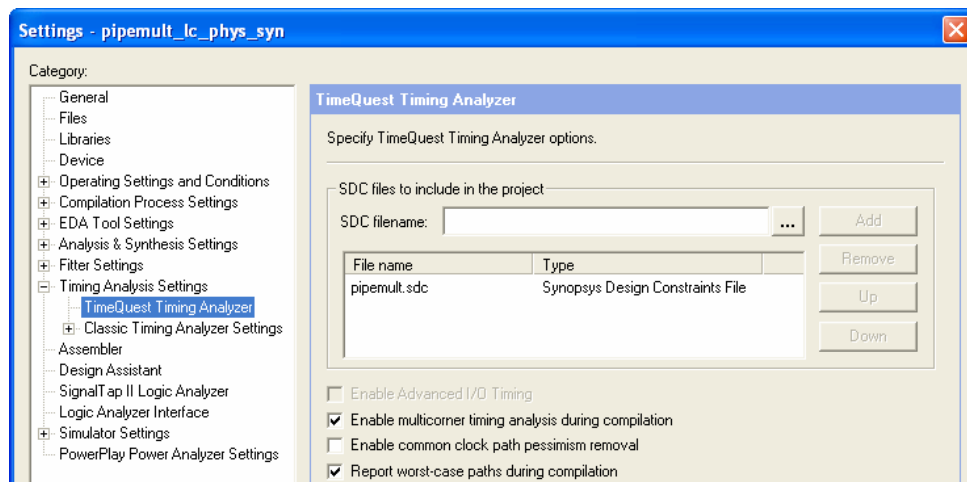
#### Step 4: Use the SDC file to guide the Quartus II fitter


1. Bring the **Quartus II** software to the foreground.
2. From the **Assignments** menu, choose **Timing Analysis Settings**.

*The **Settings** dialog box opens with the **Timing Analysis Settings** category selected.*




3. Enable **Use TimeQuest Timing Analyzer during compilation** as shown above.



4. Add the **pipemult.sdc** file to the project. In the **Settings** dialog box, click on the **TimeQuest Timing Analyzer** category (under **Timing Analysis Settings**). Use the browse button  to locate the file **pipemult.sdc**, click **Open**, and then click **Add**.

*Don't forget to click **Add**!*

5. If they are not already enabled, turn on multicornert timing analysis and the reporting of worst-case paths.
6. Click **OK** to close the **Settings** dialog box.
7. Click on  or select **Start Compilation** from the **Processing** menu.
8. When compilation is complete, open the **TimeQuest Timing Analyzer** folder in the **Compilation Report**.

*You'll see setup and hold summary reports generated by the timing analysis based on the post-fit netlist. The design now meets timing.*

### Step 5: Use the TimeQuest interface to verify timing

*To finish the TimeQuest flow, you can return to the TimeQuest tool and generate reports based off the post-fit netlist to verify that the Fitter met your timing requirements.*

1. Open the TimeQuest interface again from the Quartus II toolbar or **Tools** menu.
2. Create the timing netlist again, but this time, select a **Post-fit** netlist.
3. From the **Tasks** pane, generate the setup and hold summary reports.

*These reports should match the results seen in the Quartus II compilation report. You don't have to manually read in the SDC file or update the netlist anymore since you specified an SDC file in the Quartus II timing settings.*

*At this point, you could generate more advanced reports and get more detail about specific paths in the design. We'll talk about how to generate these reports in the next section.*

**Exercise Summary**

- Practiced basic steps for using the TimeQuest timing analyzer with the Quartus II software

**END OF EXERCISE 1**



# Exercise 2

## Timing Analysis: Clock Constraints





## Exercise 2

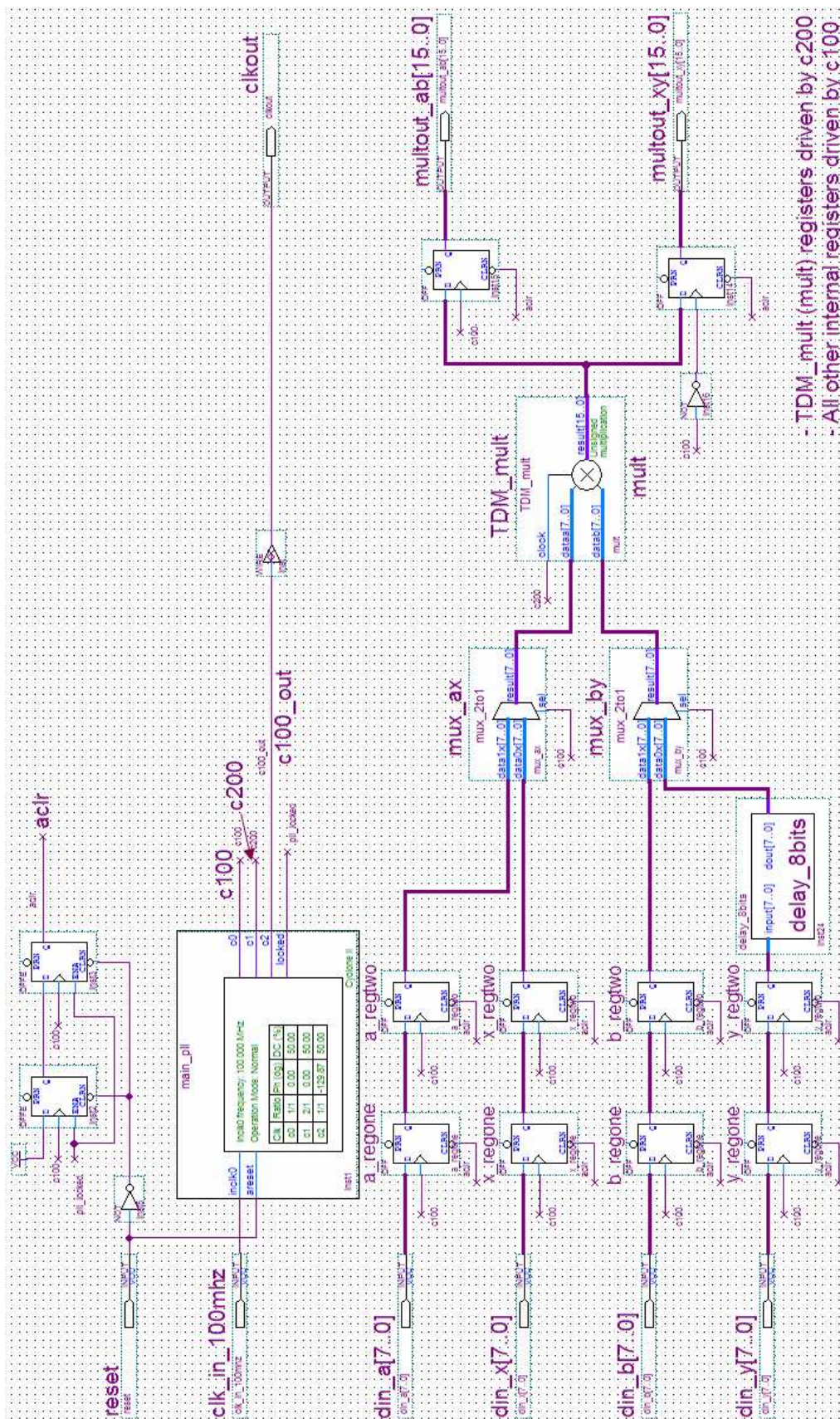
### Objectives:

- Create a new SDC file
- Use SDC to constrain the clocks in a design containing PLLs

### Top-Level Design:

The design used for the rest of the exercises in this training (shown on the following page) multiplies two sets of 8-bit data inputs: *din\_a* \* *din\_b* and *din\_x* \* *din\_y*. Along with this data input, the design also receives a board clock named *clk\_in\_100mhz* running at 100 MHz and an asynchronous reset named *reset*. To save on multiplier space, the data is time-domain multiplexed through a single multiplier running at twice the clock speed (200 MHz). All clocks for the design are generated by a PLL called *main\_pll* with 3 output clocks. A 100 MHz PLL output called *c100* is used to reduce clock tree delay to internal registers. A 200 MHz PLL output called *c200* drives the multiplier at twice the input frequency. A second 100 MHz output from the PLL called *c100\_out* drives the output port, *clkout*. The resulting data output named *multout\_ab* and *multout\_xy* is center-aligned with *clkout* (by means of the PLL) and then sent off-chip to another device on the board.

**NOTE:** Throughout the remaining exercises, you will be asked to create SDC commands without being directly guided by the instructions. It is up to you to figure out the correct commands using the training presentation material as well as the GUI tools and help information available in the TimeQuest timing analyzer. However, all SDC answers can be found in the file **top\_sdc.txt**, located in the **Solutions\Timing** subdirectory of the exercise installation directory. The solutions there are only examples, and the file itself cannot be used as the final .sdc file for the exercises. The format of your SDC commands may be different yet equivalent to the commands found in that file. If you are stuck or do not understand a solution command, please ask your instructor for assistance. The answer values for other questions in the exercises can be found in the file: **Quartus II Design Series\_Verification\_8\_0\_Exercise\_Solutions.doc** located in the same subdirectory.



- TDM\_mult (mult) registers driven by c200
- All other internal registers driven by c100

**Step 1: Create SDC file for the design**


*In this step, you will open the project and locate and constrain all the design clocks.*

1. If it's not already open, start the Quartus II software version 8.0 from the Altera program folder in the Windows Start menu. **Open** the project **top.qpf** located in the **<lab\_install\_directory>\QIIV8\_0\Timing** directory.
2. Open the file **top.bdf** by double-clicking **top** in the Project Navigator or through the **Open** command in the **File** menu.

*You should see the schematic from the previous page of this exercise manual.*

3. Synthesize the design. Click the  button or, from the **Processing** menu, choose **Start ⇒ Start Analysis & Synthesis**. Click **OK** when finished.

*Remember, we're only at the point of adding constraints, so there is no need to perform a full compilation. So you can synthesize, generate a timing netlist, and begin constraining the design with the node names from that database.*

4. Open the TimeQuest timing analyzer. Click on the  button or from the **Tools** menu, select **TimeQuest Timing Analyzer**. Remember to click **No** to indicate that you do not want to generate an SDC file from a QSF.

*Remember, you would select Yes here if (and only if) you wanted to use the TimeQuest timing analyzer with a design that previously used the Classic Timing Analyzer and had timing constraints contained in the project's .QSF file. If you do accidentally click Yes, either delete the created .SDC file or edit it for use as the main .SDC file as you proceed through the exercises.*

5. Create a slow model timing netlist. In TimeQuest, from the **Netlist** menu, select **Create Timing Netlist**. Change the **Input netlist** type to **Post-map** OR in the **Console** pane, type `create_timing_netlist -post_map -model slow`.

*Remember, you must use the **-post\_map** netlist type because you only performed synthesis (no fitting) and you cannot access the **-post\_map** option from the **Create Timing Netlist** task in the **Tasks** pane.*

6. If one hasn't already been created, create a new file called **top.sdc**. Select **File ⇒ New SDC File**. Select **File ⇒ Save As** and save the file as **<lab\_install\_directory>\QIIV8\_0\Timing\top.sdc**.

*Notice that an option, **Add file to current project**, is turned on in the **Save As** dialog box.*

**Step 2: Create base and generated clock constraints**

*Next, as with many other steps in this exercise, you will be adding a constraint to your **top.sdc** file. You will only be told what to constrain and given values. It will be up to you to use the proper SDC command.*

*Feel free to type the SDC command directly into the **SDC File Editor** or use the **Insert Constraint** submenu of the SDC File Editor's **Edit** menu, whichever you feel more comfortable with. It is recommended that you enter all constraints in the SDC File Editor instead of trying to enter constraints directly into the **Console** pane. This makes it much easier to create, edit, and manage your constraints.*

1. In the **top.sdc** file, assign a **100 MHz** clock to the input port **clk\_in\_100mhz**. Use the default name **clk\_in\_100mhz** as the name of the clock.

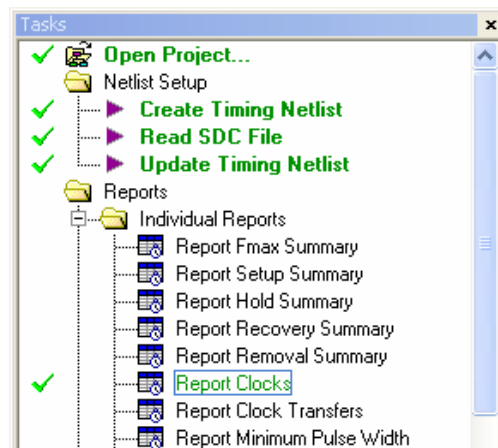
*Access the constraint GUI dialog boxes from the **Insert Constraint** submenu of the **Edit** menu. The GUI will place the constraint at the current location of the cursor, so remember to create new lines before accessing the dialog boxes. The GUI constraint dialog boxes can be used for almost all the constraints you'll be creating. However, remember from the presentation that there are some constraint options and arguments that are not in the GUI. You can always refer back to the presentation slides to find a constraint option that you may need.*

2. **(Optional)** On the line above the **create\_clock** command you've just created, insert a comment using **#** to indicate what the succeeding SDC command is doing. This is just good coding practice, and it may be helpful if you need to go back and review the constraints you've entered.
3. In **top.sdc**, add the command to automatically create generated clocks on all of the PLL outputs based on your previously specified clock input. **Hint:** this is an SDC extension command that does not require **any** optional arguments (and is **not** accessible through the GUI).

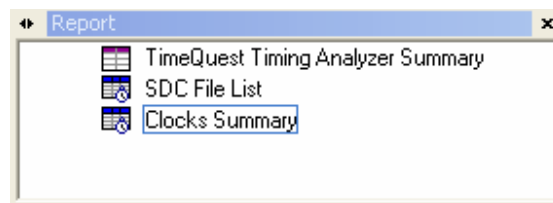
*A **hint** for an even shorter shortcut: Instead of using the **create\_clock** command (as instructed in #1 above) to create your base clock, you could use an optional argument with the command used in this step.*


4. Save and close **top.sdc**.
5. In **TimeQuest**, double-click **Report Clocks** (Tasks pane ⇒ **Reports** ⇒ **Individual Reports**).





Typically, you would read your SDC file and then update the timing netlist. The TimeQuest timing analyzer will automatically perform these tasks if you request a report via the **Tasks** pane and if your SDC file is the same name as your **Quartus II** revision or you have added your SDC file to your **Quartus II** project. Unlike the previous exercise, the **top.sdc** file name matches the project revision name.



| Clocks Summary                   |           |        |           |       |       |            |           |             |       |        |           |            |          |               |                                |                                    |  |
|----------------------------------|-----------|--------|-----------|-------|-------|------------|-----------|-------------|-------|--------|-----------|------------|----------|---------------|--------------------------------|------------------------------------|---|
| Clock Name                       | Type      | Period | Frequency | Rise  | Fall  | Duty Cycle | Divide by | Multiply by | Phase | Offset | Edge List | Edge Shift | Inverted | Master        | Source                         | Targets                            |   |
| 1 clk_in_100mhz                  | Base      | 10.000 | 100.0 MHz | 0.000 | 5.000 |            |           |             |       |        |           |            |          |               |                                | { clk_in_100mhz }                  |   |
| 2 inst1altpll_componentpllclk[0] | Generated | 10.000 | 100.0 MHz | 0.000 | 5.000 | 50.00      | 1         | 1           |       |        |           |            | false    | clk_in_100mhz | inst1altpll_componentpllclk[0] | { inst1altpll_componentpllclk[0] } |   |
| 3 inst1altpll_componentpllclk[1] | Generated | 5.000  | 200.0 MHz | 0.000 | 2.500 | 50.00      | 1         | 2           |       |        |           |            | false    | clk_in_100mhz | inst1altpll_componentpllclk[1] | { inst1altpll_componentpllclk[1] } |   |
| 4 inst1altpll_componentpllclk[2] | Generated | 10.000 | 100.0 MHz | 0.000 | 5.000 | 50.00      | 1         | 1           |       |        |           |            | false    | clk_in_100mhz | inst1altpll_componentpllclk[2] | { inst1altpll_componentpllclk[2] } |   |

In the **Report** pane of TimeQuest, a new table called **SDC File List** appears. Use this table to verify which SDC files have been read in. Next is a report called **Clocks Summary**. Use this report to verify that your clocks have been created correctly. As shown above, you should see one base clock called **clk\_in\_100mhz** and three generated clocks, one for each output of the PLL. Notice their rise and fall times. All rise times are at time 0.000. Thus, these related clocks are all in phase.

- Write out an SDC file called **test.sdc** based on current SDC assignments applied to the analysis. To do this, in the **Console** pane, type:

```
write_sdc -expand test.sdc
```

Remember, you must enter this command in the console. The **-expand** option is not in the GUI.

7. Open the file **test.sdc**. In the TimeQuest interface, from the **File** menu, select **Open SDC File**. Select **test.sdc**.

*Notice in **test.sdc** that the individual generated clock commands have been written out, replacing the **derive\_pll\_clocks** command. You could have tried to write out this file from the **Tasks** pane or the **Constraints** menu. However, the **derive\_pll\_clocks** constraint would not have been expanded since the **-expand** option would not be used.*

8. Open the file **top.sdc**.
9. Copy the 3 `create_generated_clock` commands from **test.sdc** into **top.sdc**. In **top.sdc**, make sure you comment out the `derive_pll_clocks` command.

*Note that this is not necessary. You could simply keep the **derive\_pll\_clocks** command in your SDC file. The advantage of keeping the command is that whenever the PLL settings are changed, the generated clock commands are updated automatically. However, you can't rename the generated clocks to something more meaningful, which is what you will do next. Also, **derive\_pll\_clocks** is not a standard SDC command and will typically not be recognized by other tools that support SDC.*

10. Use the table below to change each of the PLL generated output clock names to more easily recognized names. For each generated clock command in **top.sdc**, change the name of the clock (the **-name** argument ONLY) from the name in the left column of the table to the name in the right column.

| Change -name argument from:       | To:      |
|-----------------------------------|----------|
| inst1 altpll_component pll clk[0] | c100     |
| inst1 altpll_component pll clk[1] | c200     |
| inst1 altpll_component pll clk[2] | c100_out |

*Again, this is not necessary. But, it does help with the readability of later SDC commands and timing reports.*

11. Save and close **top.sdc** and **test.sdc**.
12. Reset the design. To do this, do ONE of the following:
  - a. From the **Constraints** menu, select **Reset Design**.
  - b. Double-click **Reset Design** at the bottom of the **Tasks** pane.
  - c. In the **Console** pane, type `reset_design`.

*Resetting the design tells the timing analyzer to flush all timing constraints from the netlist, thus allowing you to “start over” with new constraints on the same netlist. It is analogous to deleting the timing netlist and creating a new netlist of the same type.*

*You should now see that all of your reports have been deleted.*

### 13. Run **Report Clocks** again.

| Clocks Summary |               |           |        |           |       |       |            |           |             |       |        |           |            |          |               |                                     |                                       |
|----------------|---------------|-----------|--------|-----------|-------|-------|------------|-----------|-------------|-------|--------|-----------|------------|----------|---------------|-------------------------------------|---------------------------------------|
|                | Clock Name    | Type      | Period | Frequency | Rise  | Fall  | Duty Cycle | Divide by | Multiply by | Phase | Offset | Edge List | Edge Shift | Inverted | Master        | Source                              | Targets                               |
| 1              | c100          | Generated | 10.000 | 100.0 MHz | 0.000 | 5.000 | 50.00      | 1         | 1           |       |        |           |            | false    | clk_in_100mhz | inst1 altpll_component pll inclk[0] | { inst1 altpll_component pll clk[0] } |
| 2              | c100_out      | Generated | 10.000 | 100.0 MHz | 0.000 | 5.000 | 50.00      | 1         | 1           |       |        |           |            | false    | clk_in_100mhz | inst1 altpll_component pll inclk[0] | { inst1 altpll_component pll clk[2] } |
| 3              | c200          | Generated | 5.000  | 200.0 MHz | 0.000 | 2.500 | 50.00      | 1         | 2           |       |        |           |            | false    | clk_in_100mhz | inst1 altpll_component pll inclk[0] | { inst1 altpll_component pll clk[1] } |
| 4              | clk_in_100mhz | Base      | 10.000 | 100.0 MHz | 0.000 | 5.000 |            |           |             |       |        |           |            |          |               |                                     | { clk_in_100mhz }                     |

*Your clocks should appear as above with all the clock names you entered in the SDC file.*

### 14. Check to see if all clocks are constrained. What report should you run?

| Unconstrained Paths Summary |                                 |       |      |
|-----------------------------|---------------------------------|-------|------|
|                             | Property                        | Setup | Hold |
| 1                           | Illegal Clocks                  | 0     | 0    |
| 2                           | Unconstrained Clocks            | 0     | 0    |
| 3                           | Unconstrained Input Ports       | 33    | 33   |
| 4                           | Unconstrained Input Port Paths  | 34    | 34   |
| 5                           | Unconstrained Output Ports      | 33    | 33   |
| 6                           | Unconstrained Output Port Paths | 33    | 33   |

| Clock Status Summary |               |           |             |
|----------------------|---------------|-----------|-------------|
|                      | Clock         | Type      | Status      |
| 1                    | c100          | Generated | Constrained |
| 2                    | c100_out      | Generated | Constrained |
| 3                    | c200          | Generated | Constrained |
| 4                    | clk_in_100mhz | Base      | Constrained |

*You should find no unconstrained clocks or illegal clocks.*

### Exercise Summary

- Created a new SDC file
- Constrained the input and PLL clocks in the design

**END OF EXERCISE 2**



# Exercise 3

## Timing Analysis: Synchronous I/O Constraints



## Exercise 3

### Objective:

- *Constrain the synchronous I/O paths in the design*

**Step 1: Constrain I/O paths using SDC**

Now we need to define the FPGA's I/O timing relative to external devices.

1. Open **top.sdc**.

The upstream devices (sending data to **din\_a**, **din\_b**, **din\_x** and **din\_y**) have timing numbers as shown in the following table. Let's assume the board is being laid out with a star topology clocking scheme. Thus, all devices using the master clock (**clk\_in\_100mhz**) should be clocked at (relatively) the same time.

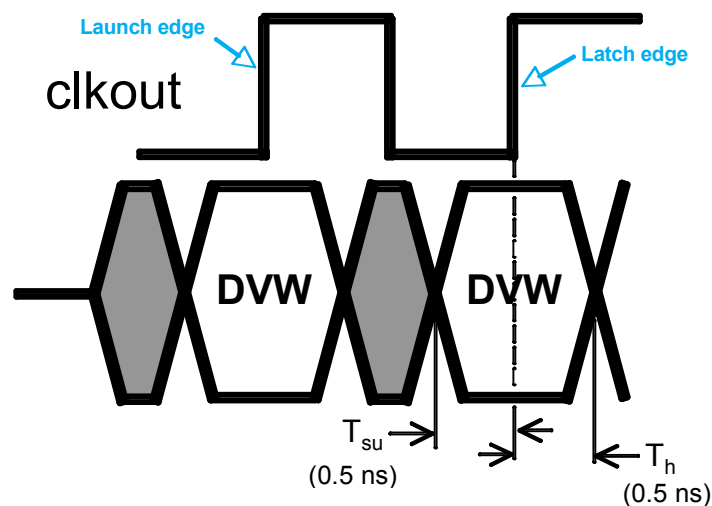
|   | <i>Minimum</i> | <i>Maximum</i> |
|---|----------------|----------------|
| <b><i>Clock-to-output Delay (ns)</i></b>                            | <b>1</b>       | <b>3</b>       |
| <b><i>Estimated PCB Data Trace Delay (between devices) (ns)</i></b> | <b>0.5</b>     | <b>1</b>       |
| <b><i>Board Clock Skew (ns)</i></b>                                 | <b>-0.5</b>    | <b>0.5</b>     |

2. Use the table and diagram above to fully constrain all input data ports with respect to **clk\_in\_100mhz**. To refresh your memory, the equations are below:

|   |
|---|
| $\text{set\_input\_delay (max)} = \text{board delay (max)} - \text{clock skew (min)} + \text{ext. } t_{co} \text{ (max)}$ $\text{set\_input\_delay (min)} = \text{board delay (min)} - \text{clock skew (max)} + \text{ext. } t_{co} \text{ (min)}$ |
|---|

For the downstream device (receiving data from **multout\_ab** and **multout\_xy**), a source synchronous interface has been implemented. The data is transmitted **center-aligned** to the clock.

The downstream device in this example requires a **1-ns data valid window**. That's  $\pm 0.5$  ns with respect to the device output port, **clkout**, which is driven by the PLL generated clock, **c100\_out**.



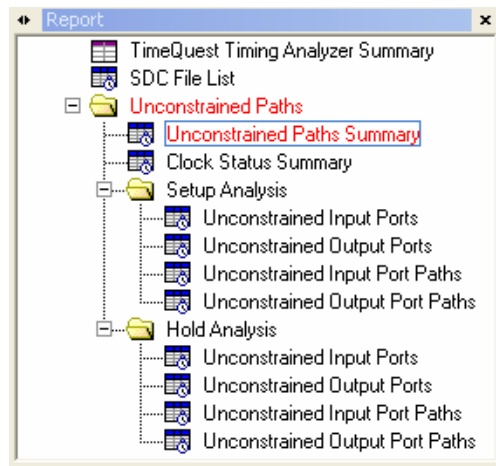
3. In **top.sdc**, constrain the data output ports. Create a generated clock targeted to the **clkout** output port sourced by the output of the PLL, and then add **set\_output\_delay** constraints to all of the data output ports.

*Hints:* Remember that the source for the generated clock needs to be a physical location in the netlist, not a clock name. For the source for this new generated clock, you could copy the PLL output pin name from an earlier constraint or you can try using a **Hierarchical** search in the Name Finder to find the pin name. Make sure to place the generated clock early in your SDC file (group it with other clock constraints) to make sure it is recognized by any new constraints you add. With wildcards, you should only need two **set\_output\_delay** constraint commands.

Remember that you can derive your setup and hold times (and in turn maximum and minimum output delays) from the data valid window with source synchronous interfaces. We'll assume that the board trace delays for the clock and data are evenly matched. In reality, you should include their maximum and minimum delays in your constraint, even though they would be small.

4. Save **top.sdc**.
5. Reset the design and check your newly entered constraints. Use the **Tasks** pane to run reports to check the new clock, for ignored constraints, and for unconstrained paths.

Use the **Ignored Constraints** report to make sure you didn't type anything incorrectly. Use the **Unconstrained Paths** report to see what hasn't been constrained.



| Unconstrained Paths Summary |                                 |       |      |
|-----------------------------|---------------------------------|-------|------|
|                             | Property                        | Setup | Hold |
| 1                           | Illegal Clocks                  | 0     | 0    |
| 2                           | Unconstrained Clocks            | 0     | 0    |
| 3                           | Unconstrained Input Ports       | 1     | 1    |
| 4                           | Unconstrained Input Port Paths  | 2     | 2    |
| 5                           | Unconstrained Output Ports      | 1     | 1    |
| 6                           | Unconstrained Output Port Paths | 1     | 1    |

*At this point, you should have one remaining unconstrained input and 2 unconstrained input paths along with one unconstrained output and output path. If you look at the **Setup Analysis** and **Hold Analysis** reports (under **Unconstrained Paths**), you will see that these are caused by the reset and the clock output. You will fix these in exercise 4.*

*You could now take this SDC file into the Quartus II software and use it to compile your design. However, as you can see, there are still some remaining paths to constrain before that would be truly useful. Remember, the idea with constraining is that you know how the design should work. Thus, you need to pass all timing information on to the compiler so it can adjust the fit of your design based on that information.*

**Exercise Summary**

- Constrained a synchronous input interface
- Constrained a source synchronous output interface

**END OF EXERCISE 3**





## Exercise 4

# Timing Analysis: Timing Exceptions & Analysis



## Exercise 4

### Objectives:

- *Constrain asynchronous input signals*
- *Eliminate timing violations by using timing exceptions*

**Step 1: Constrain asynchronous path**

*In this step, you will constrain the asynchronous input path of this design.*

*Look at the reset circuit in the design schematic. You can see that the design has a reset driven by an external source. Let's assume the input path is truly asynchronous (no timing on the external path), so the solution to synchronize the reset internally is the correct one. The circuit shown is a common method of synchronizing a reset to your internal clock domain. Since this is a truly asynchronous input and no external timing is known, a false path exception is needed to constrain.*

1. In **top.sdc**, use an SDC command to constrain all paths from the asynchronous reset input.
2. Save **top.sdc**.
3. Reset the design and check your ignored constraints and unconstrained paths reports.

| Unconstrained Paths Summary |                                 |       |      |
|-----------------------------|---------------------------------|-------|------|
|                             | Property                        | Setup | Hold |
| 1                           | Illegal Clocks                  | 0     | 0    |
| 2                           | Unconstrained Clocks            | 0     | 0    |
| 3                           | Unconstrained Input Ports       | 0     | 0    |
| 4                           | Unconstrained Input Port Paths  | 0     | 0    |
| 5                           | Unconstrained Output Ports      | 1     | 1    |
| 6                           | Unconstrained Output Port Paths | 1     | 1    |

*You should have only one output port and one output path unconstrained now.*

**Step 2: Constrain clock output port**

1. In the **Unconstrained Paths** report folder, examine the **Unconstrained Output Ports** and **Unconstrained Output Port Paths** reports (expand the **Setup Analysis** or **Hold Analysis** folders).

*You should see that the port **clkout** is the only unconstrained output. It appears as unconstrained because the timing analyzer has determined the path from the clock output of the PLL to the **clkout** port is an unconstrained combinatorial path. This path is considered as both a clock path (for the source synchronous interface) and a data path (any path that goes to an output port) thanks to the TimeQuest timing analyzer's "clock as data" analysis feature. Remember that a "clock" in SDC is not a node. It refers to the behavior of a node in the design. Without TimeQuest's clock-as-data analysis, besides **clkout**, **clk\_in\_100mhz** would be considered an unconstrained input port, even with the **create\_clock** constraint we created at the beginning of Exercise 2.*

*While the path from the PLL to **clkout** is actually a true combinatorial path in the design, it is a timing-based false path and does not need to be analyzed for timing. By design, you are only concerned with the relationship of the output clock to the output data and not with the absolute time between **c100\_out** and **clkout**.*

2. In **top.sdc**, use an SDC command to tell TimeQuest not to analyze the path from the PLL output pin to the clock output port **clkout**.

*Hint: Again, you could copy the pin name from an earlier constraint or you can try using a **Hierarchical** search in the Name Finder to find the pin name.*

3. Save **top.sdc**.
4. Reset the design and check your ignored constraints and unconstrained paths reports.

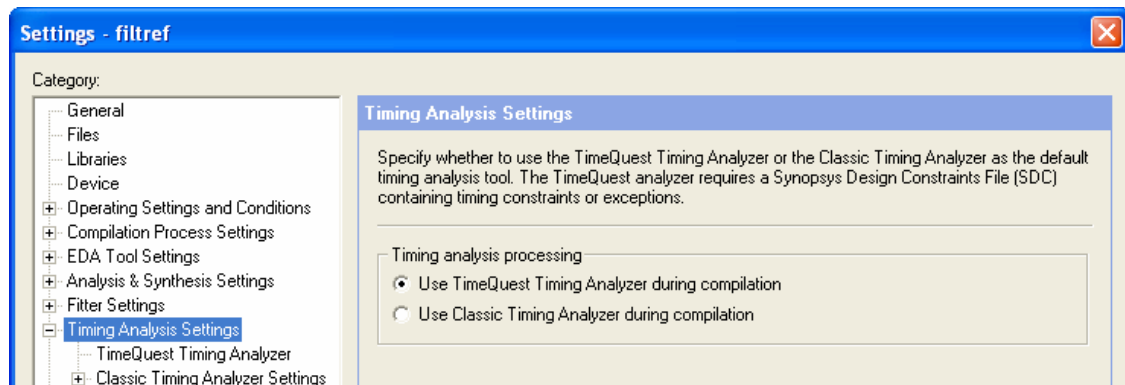
*All paths should now be constrained.*

**Step 3: Run compilation using SDC file**

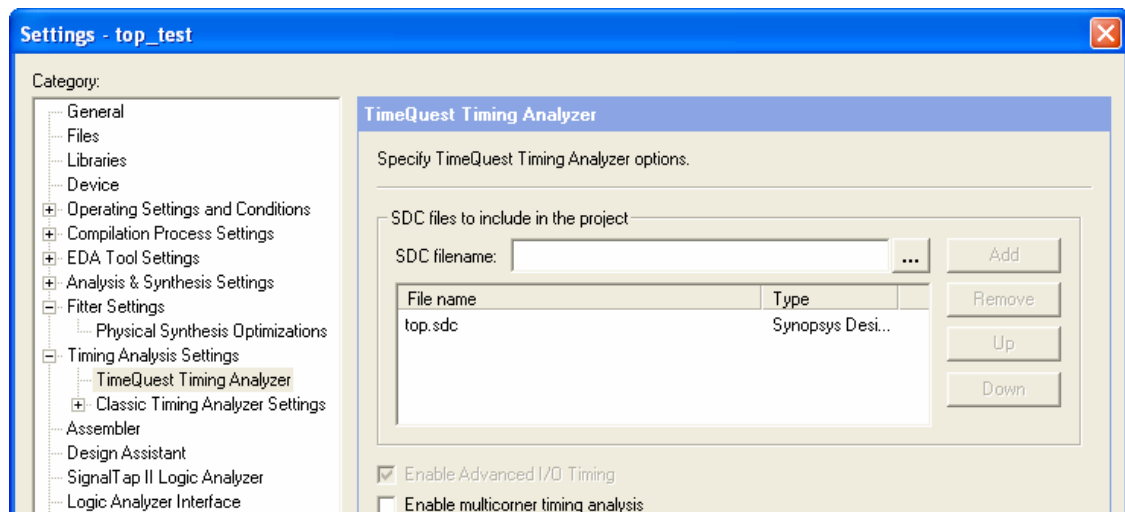
*Now we'll run the Quartus II fitter using the constraints stored in the SDC file.*


1. Bring the Quartus II software to the foreground.
2. From the **Assignments** menu, choose **Timing Analysis Settings**.

*The **Settings** dialog box opens with the **Timing Analysis Settings** category selected.*




3. If it's not already selected, enable **Use TimeQuest Timing Analyzer during compilation** as shown above.



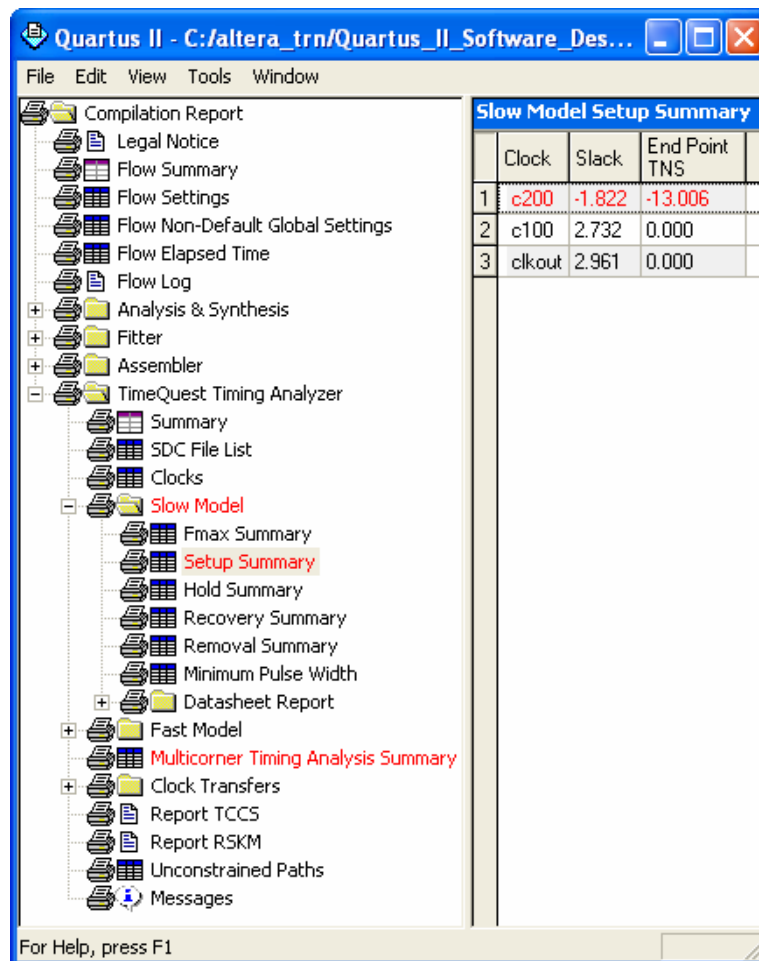
4. Add your **top.sdc** file to the project if it has not been added already. The file should have been added here when it was saved initially. If it wasn't, in the **Settings** dialog box, click on the **TimeQuest Timing Analyzer** category (under **Timing Analysis Settings**). Use the browse button  to locate the file **top.sdc**. Click **Open**, and then click **Add** to add the file to the list.
5. If they're not already enabled, turn on multicorner timing analysis and the reporting of worst-case paths. Leave common clock path pessimism (CCPP) removal disabled.

*CCPP removal is not supported by the Cyclone II device used in this lab.*

6. Click **OK** to close the **Settings** dialog box.
7. Click on the  button or select **Start Compilation** from the **Processing** menu.
8. When compilation is complete, open the **TimeQuest Timing Analyzer** folder in the **Compilation Report**.

*You should see the **Slow Model** folder and the **Multicorner Timing Analysis Summary** report appear in red indicating that you are failing timing. Typically, the slow model would fail setup timing because slow signals do not reach their destinations early enough to meet setup requirements. While not the case here, the fast model for a design might fail hold timing because fast signals may change their values before meeting hold timing requirements. Remember that if we can meet setup timing in the slow model and hold timing in the fast model, the design will meet timing across all PVT.*

9. Expand the **Slow Model** folder and select the **Setup Summary** table to see which clock is failing. Look at the **Multicorner Timing Analysis Summary** to see timing information for all analyses (setup, hold, recovery, and removal) for both the slow and fast models.



The screenshot shows the Quartus II software interface. The left pane displays a tree view of the compilation report, with the 'TimeQuest Timing Analyzer' folder expanded. Under 'TimeQuest Timing Analyzer', the 'Slow Model' folder is expanded, showing various summary reports. The 'Setup Summary' report is highlighted in red, indicating a timing failure. The right pane displays the 'Slow Model Setup Summary' table, which shows the following data:

|   | Clock  | Slack  | End Point TNS |
|---|--------|--------|---------------|
| 1 | c200   | -1.822 | -13.006       |
| 2 | c100   | 2.732  | 0.000         |
| 3 | clkout | 2.961  | 0.000         |

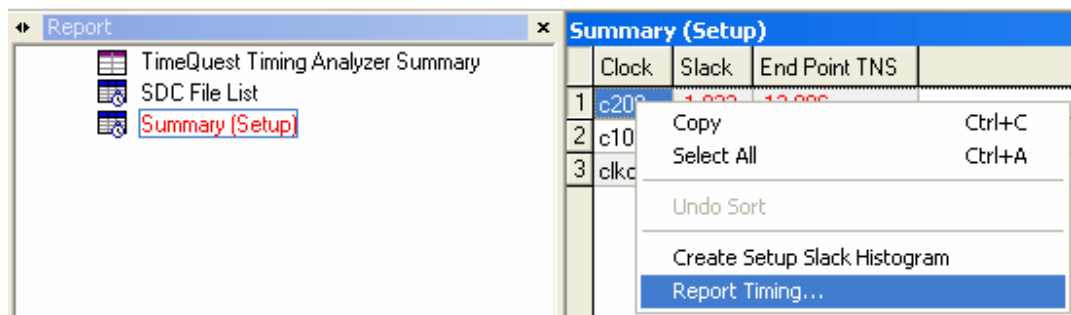
The **Setup Summary** table reveals that clock **c200** is failing timing by **1.822 ns** in the setup analysis, but that's about all the information the TimeQuest reports in the Compilation Report give us. There is no indication in any of the reports here what path(s) is/are failing. We'll use the TimeQuest timing analyzer's more extensive reporting capabilities to analyze where the failure is and why it is occurring.

#### Step 4: Analyze c200 clock domain using TimeQuest reports

1. Open the TimeQuest interface again if it is not already open.
2. Double-click **Report Setup Summary** in the **Tasks** pane. Remember this automatically runs **Create Timing Netlist** (now with the **Post-fit** netlist), **Read SDC File**, and **Update Timing Netlist**.

Again, you will see a table similar to the **Setup Summary** table in the **Compilation Report** indicating the **c200** clock domain is failing.

One way to run a detailed analysis of the **c200** clock domain is to use **Report Timing** (**Tasks** pane, **Reports** menu, or `report_timing` command). You will do this in an even easier way by means of the TimeQuest GUI.



3. In the **Summary Setup** table, select the **c200** clock, then right-click **c200** and select **Report Timing**.



**Report Timing**

Clocks:  
 From clock:   
 To clock: c200

Targets:  
 From:   
 Through:   
 To:

Analysis type:  
☒ Setup ☐ Recovery  
☐ Hold ☐ Removal

Paths:  
 Report number of paths: 10  
 Maximum slack limit:  ns

Output:  
 Detail level: **Summary**   
☒ Report panel name: Setup: c200 Summary  
☐ File name:   
 File options:  
☒ Overwrite ☐ Append   
☐ Console

Tcl command: report\_timing -to\_clock { c200 } -setup -npaths 10 -detail summary -

4. In the **Report Timing** dialog box, select **Summary** in the **Output** ⇒ **Detail Level** drop-down menu. Click the **Report Timing** button.

| Setup: c200 Summary |        |             |  |              |             |
|---------------------|--------|-------------|--|--------------|-------------|
|                     | Slack  | From Node   | To Node  | Launch Clock | Latch Clock |
| 1                   | -1.822 | y_regtwo[7] | ...m_mult:lpn_mult_component(mult_dcq:auto_generated)mac_mult1~OBSERVABLEDATAB_REGOUT7 | c100         | c200        |
| 2                   | -1.806 | y_regtwo[6] | ...m_mult:lpn_mult_component(mult_dcq:auto_generated)mac_mult1~OBSERVABLEDATAB_REGOUT6 | c100         | c200        |
| 3                   | -1.775 | y_regtwo[2] | ...m_mult:lpn_mult_component(mult_dcq:auto_generated)mac_mult1~OBSERVABLEDATAB_REGOUT2 | c100         | c200        |
| 4                   | -1.755 | y_regtwo[1] | ...m_mult:lpn_mult_component(mult_dcq:auto_generated)mac_mult1~OBSERVABLEDATAB_REGOUT1 | c100         | c200        |
| 5                   | -1.577 | y_regtwo[3] | ...m_mult:lpn_mult_component(mult_dcq:auto_generated)mac_mult1~OBSERVABLEDATAB_REGOUT3 | c100         | c200        |
| 6                   | -1.538 | y_regtwo[5] | ...m_mult:lpn_mult_component(mult_dcq:auto_generated)mac_mult1~OBSERVABLEDATAB_REGOUT5 | c100         | c200        |
| 7                   | -1.376 | y_regtwo[4] | ...m_mult:lpn_mult_component(mult_dcq:auto_generated)mac_mult1~OBSERVABLEDATAB_REGOUT4 | c100         | c200        |
| 8                   | -1.357 | y_regtwo[0] | ...m_mult:lpn_mult_component(mult_dcq:auto_generated)mac_mult1~OBSERVABLEDATAB_REGOUT0 | c100         | c200        |
| 9                   | 1.931  | a_regtwo[6] | ...m_mult:lpn_mult_component(mult_dcq:auto_generated)mac_mult1~OBSERVABLEDATAA_REGOUT6 | c100         | c200        |
| 10                  | 1.940  | a_regtwo[5] | ...m_mult:lpn_mult_component(mult_dcq:auto_generated)mac_mult1~OBSERVABLEDATAA_REGOUT5 | c100         | c200        |

The **Setup: c200 Summary** table appears. This table displays the 10 worst paths for the **c200** clock domain. This is only a summary table, so no detail on these paths is provided, just slack, source register/clock, and destination register/clock. But you should see eight delay paths that are missing timing and shown in red. All of these paths are launched by the **c100** clock and latched by the **c200** clock.

5. Using the **Setup: c200 Summary** report, analyze the timing violation paths in the **c200** domain. What is the common source and destination of all 8 paths? Record below.

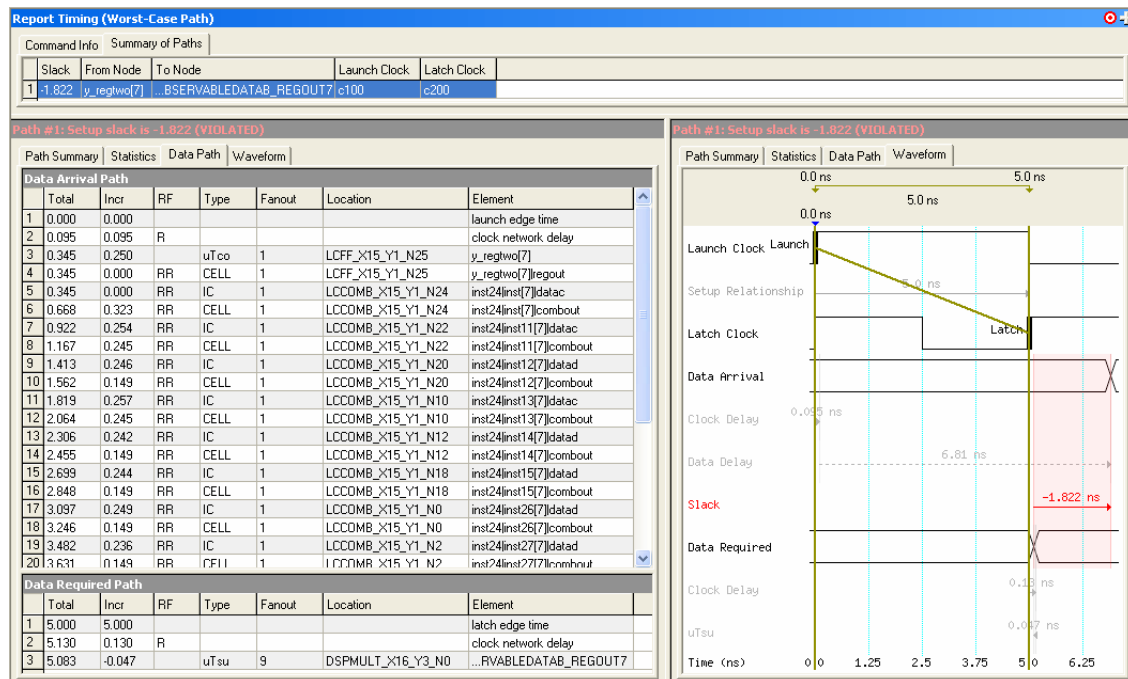
| c200 Timing Violation Paths |  |
|-----------------------------|--|
| Source                      |  |
| Destination                 |  |

6. Examine detail on the worst slack timing path for **c200**. Select the worst slack in the list (first row), right-click, and select **Report Worst-Case Path**.

*A new report appears called **Report Timing (Worst-Case Path)**. The top of this new report shows the same information as in the summary report. The lower panels are called **Path #1: Setup Slack is -1.822 (VIOLATED)**.*

7. Click through the tabs in the lower panels of the reports to see different information about the selected failing path.

*The **Path Summary** tab slightly expands on the summary report by displaying the data arrival and required times calculated by the timing analyzer. The **Statistics** tab displays statistical information that breaks down the amount of time the signal spends traveling through path interconnect (IC) and logic cells. The **Data Path** tab provides detailed information about the actual device logic and interconnect the signal passes through as part of the data arrival and data required paths. Finally, the **Waveform** tab displays waveforms that show exactly how the timing analyzer arrived at its slack calculation. Click and drag in the waveform to add cursors that “snap” to events such as the launch and latch edges.*



You should see the timing report above, which gives all the detailed timing information about the requested path.

- Analyze the design to determine how to fix the violations. Bring the **top.bdf** schematic to the foreground.

Notice there is a delay block called **delay\_8bits** in the output path of register **y\_regtwo**, which is the second bank of registers fed by the input bus **din\_y**. This delay block has been inserted purposely to represent some logical delay in the design. This delay block is causing the timing violations for clock **c200** driving the multiplier. Also notice that the **x\_regtwo** and **y\_regtwo** outputs are connected to the zero input of the multiplexers (**mux\_ax** & **mux\_by**), whose select lines are controlled by clock **c100**. This means that the multiplexers only select the output of registers **x\_regtwo** and **y\_regtwo** during the negative cycle of clock **c100**.

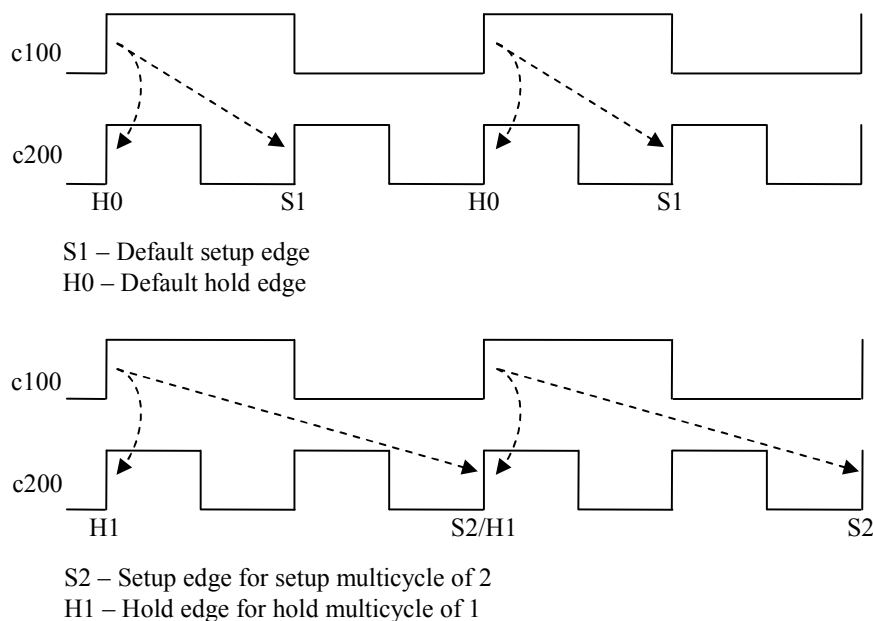
The TimeQuest timing analyzer automatically assumes data from all the registers must reach the multiplier within one cycle of the 200 MHz clock (the first 5 ns or when **c100** is high). But this is only true for the outputs of **a\_regtwo** and **b\_regtwo**. The **x\_regtwo** and **y\_regtwo** data is computed on the second cycle of **c200**, so this data has **10 ns** to reach the multiplier. So, in this case, a **multicycle** assignment can fix the analysis while accurately describing the way the design is supposed to work.

**Step 5: Use multi-cycle constraints to correct failing internal paths**

1. In **top.sdc**, add an SDC command to make the paths from the **x\_regtwo\*|clk** and **y\_regtwo\*|clk** registers have a setup multicycle path of **2** (no **–to** argument is required). The multicycle assignment should be based on the **destination clock edges**.

*Hint: This can be done with one or two lines in the SDC file. Remember that timing exceptions applied to registers have the source set to the clock pin of the source register.*

2. Add a **hold multicycle** of **1** for these same paths.



*These setup and hold multicycle constraints tell the timing analyzer that data from the **x\_regtwo** and **y\_regtwo** registers have the full two cycles of the 200 MHz clock to reach the multiplier. It is good to note again that a hold multicycle value in SDC would not be the same value used in the Classic Timing Analyzer. To convert to SDC on projects constrained using the Classic Timing Analyzer, you subtract 1 from the hold multicycle value.*

3. Save **top.sdc**.

- Reset the design (no need to recompile yet), update your timing netlist, and run **Report Setup Summary** and **Report Hold Summary**.

| Summary (Setup) |        |       |               |
|-----------------|--------|-------|---------------|
|                 | Clock  | Slack | End Point TNS |
| 1               | c200   | 1.931 | 0.000         |
| 2               | c100   | 2.732 | 0.000         |
| 3               | clkout | 2.961 | 0.000         |

| Summary (Hold) |        |       |               |
|----------------|--------|-------|---------------|
|                | Clock  | Slack | End Point TNS |
| 1              | c100   | 0.515 | 0.000         |
| 2              | clkout | 1.234 | 0.000         |
| 3              | c200   | 1.327 | 0.000         |

*All clock domains should now be shown in black to indicate they are meeting timing.*

### Step 6: Analyze asynchronous timing in the design

- From the **Tasks** pane, generate the recovery and removal summary reports.

*You should see only positive slack on all asynchronous signals in the design. The summary reports indicate that all asynchronous signals are within the **c100** clock domain. Let's examine the asynchronous paths in more detail.*

- In the **Summary (Recovery)** report, select and right-click **c100**, and select **Report Timing**.
- Set the report **Detail level** to **Summary**, and set the number of paths to analyze to **100**. Click **Report Timing**.
- Scroll to the bottom of the new report and note the number of paths reported. What is the common source for all these paths? Check the schematic to find this instance in the design.

*Notice that this is the second register of the reset synchronization circuit that generates the **aclr** signal. Since this signal connects to the asynchronous reset pins of 96 registers in the design, 96 paths are listed in the report. The input **reset** signal is also asynchronous, but since we set **reset** to be a false path, it does not get analyzed or included in the report.*

*Note that both the launch and latch clocks for all 96 paths is the **c100** clock. Remember that the recovery analysis is analogous to the setup analysis of synchronous signals and removal analysis is analogous to hold. Since this asynchronous signal is synchronized into the **c100** clock domain, **c100** is the clock indicated in the recovery and removal summary reports.*

### Step 7: Use SDC to fix source-synchronous output analysis

- Look at the slack times for all multiplier outputs. In the **Summary (Setup)** report, select and right-click on the clock **clkout** and choose **Report Timing**. In the **Report Timing** dialog box, change the **Detail level** to **Summary** and change the number of paths to **32**. Click **Report Timing**.

*Notice that the slack calculations on the **multout\_xy** bus are over 4 ns less than their **multout\_ab** counterparts. Highlight paths (rows) 1 through 16 to see this easily.*

- Look at the **Data Arrival Path** for any of the **multout\_xy** paths. Select and right-click any of the **multout\_xy** outputs in the **Setup: clkout Summary** table. Choose **Report Worse-Case Path**.

What is the launch edge time? \_\_\_\_\_

*Why does the launch edge time not start at 0? This is because the register **inst14** (the register driving **multout\_xy**) is clocked by a falling edge. So these paths only have a half-cycle (5 ns) to meet timing, not the full 10 ns of the destination clock cycle. Let's assume that we will fix this on our downstream device by using negative edge-triggered registers on that device's input bus. Now we need to tell the timing analyzer that this is the case.*

- In **top.sdc**, add the **–clock\_fall** argument to the output delay assignments for the **multout\_xy** bus. **DO NOT** add the **–clock\_fall** argument to the **multout\_ab** bus.

*Hint: This requires going back to the **set\_output\_delay** constraints you created earlier and editing them. You should end up with 4 of these constraints instead of the original 2 if you use wildcards.*

- Save **top.sdc**.
- Reset the design (again, no need to recompile), update your timing netlist, and regenerate the setup and hold summary reports as well as the **Setup: clkout Summary** report you created in #1 above. You can use the GUI again or use the commands as still listed in the **History** tab of the console.

*All the slack values found in the **Setup: clkout Summary** should be closer than before.*

### Step 8: Use the PLL to center the data valid window

*Though all timing constraints are being met, typically you want to center your actual data valid window around your output clock, in this case **clkout**. Notice from the setup and hold summary reports that the setup slack for **clkout** is much bigger than the hold slack. This means that the data valid window is skewed with respect to the data required window. To fix this, you will adjust the PLL to shift the **c100\_out** (**clkout**) clock.*

- In the table below, under **Slow Analysis**, record the slack values for **clkout** from the **Summary (Setup)** and **Summary (Hold)** tables.

| Slow Analysis |            | Fast Analysis |            |
|---------------|------------|---------------|------------|
| Setup Slack   | Hold Slack | Setup Slack   | Hold Slack |
|               |            |               |            |

2. Delete the current timing netlist. From the **Netlist** menu, select **Delete Timing Netlist** or type `delete_timing_netlist` in the **Console** pane.
3. Generate a netlist using the fast timing model. In the **Netlist** menu, click **Create Timing Netlist**. In the dialog box, leave the **Input netlist** type to **Post-fit** and enable **Fast-corner** (under **Delay model**). Click **OK**. Alternatively, you can type `create_timing_netlist -model fast` in the **Console** pane.

*Instead of deleting and recreating the fast model netlist, you could have just changed the operating conditions with the **Set Operating Conditions** command from the **Netlist** menu and selected the **MIN\_fast** operating condition. This performs the same operation without needing to first delete the netlist. There's also no need to recreate the reports from scratch. They'll just need to be updated based on the new netlist. Right-click an **Out of Date** report in the **Reports** pane and select **Regenerate All Out of Date**.*

4. Update your timing netlist and regenerate the setup and hold summary reports.
5. In the table above, under **Fast Analysis**, record the slack values from the **Summary (Setup)** and **Summary (Hold)** tables for **clkout**.

*To figure out how much the PLL output clock **c100\_out** needs to be shifted, we need to find the center of the data valid window. The fast and slow models give you the absolute smallest slack for both corner cases. Now, if you subtract the smallest setup slack value in the table (from either the fast or slow analyses) from the smallest hold slack value from either analysis and divide the result by 2, you can find this center. Round the calculated value to the nearest hundredth (2 decimal places).*

(SMALLEST HOLD VALUE – SMALLEST SETUP SLACK) ÷ 2 = \_\_\_\_\_

6. Add this time shift on the **c2** output of the PLL. Bring the **top.bdf** schematic to the foreground. Double-click the PLL block in the schematic. In the **MegaWizard Plug-In Manager**, go to **page 6 (clk c2 of the Output Clocks section)**. In the **Clock phase shift** field, enter the rounded time delay shift you calculated above (with the negative sign) and change the units to **ns**. The MegaWizard indicates what the actual shift will be, which may be slightly different from your entered value. Click **Finish** twice and then click **OK**. You will also have to update the symbol.

***NOTE:** This may resize the PLL symbol in the diagram so make sure that all connections to it are still correct.*

7. From the PLL symbol in the schematic, note the equivalent **phase shift** in degrees that the **MegaWizard Plug-In Manager** has determined will give you your shift on the **c2** output. Enter that value here: \_\_\_\_\_ °
8. In **top.sdc**, add the **-phase** argument to your generated clock statement for the **c100\_out** (not **clkout**) clock using the **phase shift value** from the PLL symbol (units are not required).
9. Save **top.sdc**.

*Before recompiling, you should lock down device I/O selected by the fitter during the last compilation. This ensures that the fitter does not select different locations for your I/O ports and thus possibly change your timing.*

10. Fix device I/O locations. From the Quartus II **Assignments** menu, select **Back-Annotate Assignments**. In the **Back-Annotate Assignments** dialog box, leave the defaults (**Pin & device assignments** along with **Delay chains**). Click **OK**.
11. **Recompile** the design, saving changes to the .bdf schematic file.
12. Check timing. After compilation, re-open the timing analyzer and recheck your **Summary (Setup)** and **Summary (Hold)** reports. Check both the slow and fast netlists by just changing the operating conditions (**Netlist** menu, **Set Operating Conditions**). Enter the new values for **clkout** here:

| Slow Analysis |            | Fast Analysis |            |
|---------------|------------|---------------|------------|
| Setup Slack   | Hold Slack | Setup Slack   | Hold Slack |
|               |            |               |            |

*Now you should notice more of a balance between the slack times. This indicates that the data valid window is now centered around the clock and the data required times.*

### Exercise Summary

- Constrained an asynchronous control input
- Adding timing exceptions (i.e. false path and multicycle) to match design functionality
- Adjusted PLL settings to improve timing of a source synchronous interface

## END OF EXERCISE 4