



CUSTOMER EDUCATION SERVICES

# **Design Compiler 1 Workshop**

## **Lab Guide**

10-I-011-SLG-013

2007.03

**Synopsys Customer Education Services**  
700 East Middlefield Road  
Mountain View, California 94043

Workshop Registration: **1-800-793-3448**

[www.synopsys.com](http://www.synopsys.com)

# Copyright Notice and Proprietary Information

Copyright © 2007 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

## Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

"This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of \_\_\_\_\_ and its employees. This is copy number \_\_\_\_\_."

## Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

## Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

## Registered Trademarks (®)

Synopsys, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, and Vera are registered trademarks of Synopsys, Inc.

## Trademarks (™)

Active Parasitics, AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BOA, BRT, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, DC Expert, DC Professional, DC Ultra, Design Advisor, Design Analyzer, Design Vision, DesignerHDL, DesignTime, Direct RTL, Direct Silicon Access, Discovery, Dynamic Model Switcher, Dynamic-Macromodeling, EDANavigator, Encore, Encore PQ, Evaccess, ExpressModel, Formal Model Checker, FoundryModel, Frame Compiler, Galaxy, Gatran, HANEX, HDL Advisor, HDL Compiler, Hercules, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSIM plus, HSPICE-Link, i-Virtual Stepper, iN-Tandem, Integrator, Interactive Waveform Viewer, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JvXtreme, Liberty, Libra-Passport, Libra-Visa, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Milkyway, ModelSource, Module Compiler, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Orion\_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Power Compiler, PowerCODE, PowerGate, ProFPGA, ProGen, Prospector, Raphael, Raphael-NES, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, Softwire, Source-Level Design, Star-RCXT, Star-SimXT, Taurus, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCSExpress, VCSi, VerificationPortal, VFormal, VHDLCompiler, VHDLSystem Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

## Service Marks (SM)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.  
ARM and AMBA are registered trademarks of ARM Limited.  
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.  
All other product or company names may be trademarks of their respective owners.

Document Order Number: 10-I-011-SLG-013  
Design Compiler 1 Lab Guide

# 2

## Setup and Synthesis Flow

### Learning Objectives

After completing this lab, you should be able to:

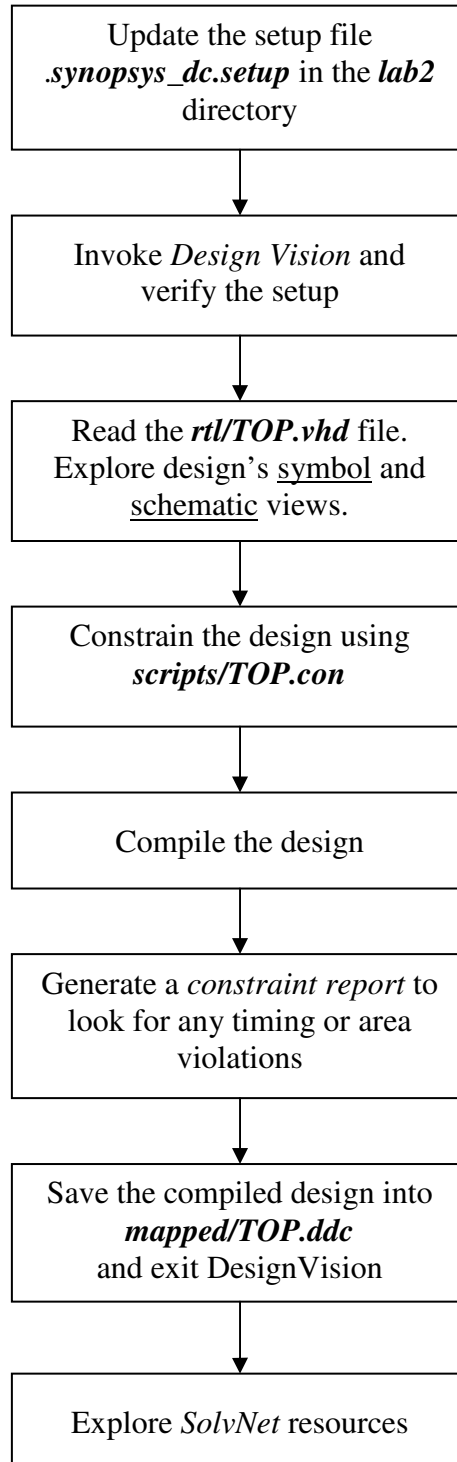
- Update a DC setup file
- Navigate the schematic in *Design Vision*
- Take a design through the basic synthesis steps
- Visit *SolvNet* to browse the user manual for Design Vision



**Lab Duration:**  
50 minutes

# Lab Flow

Follow the detailed step-by-step **Lab Instructions** on the following pages to perform the steps highlighted in this flow:



# Lab Instructions

## Task 1. Update the setup file

---

1. Make the *lab2* directory your working directory.

```
UNIX% cd lab2
```

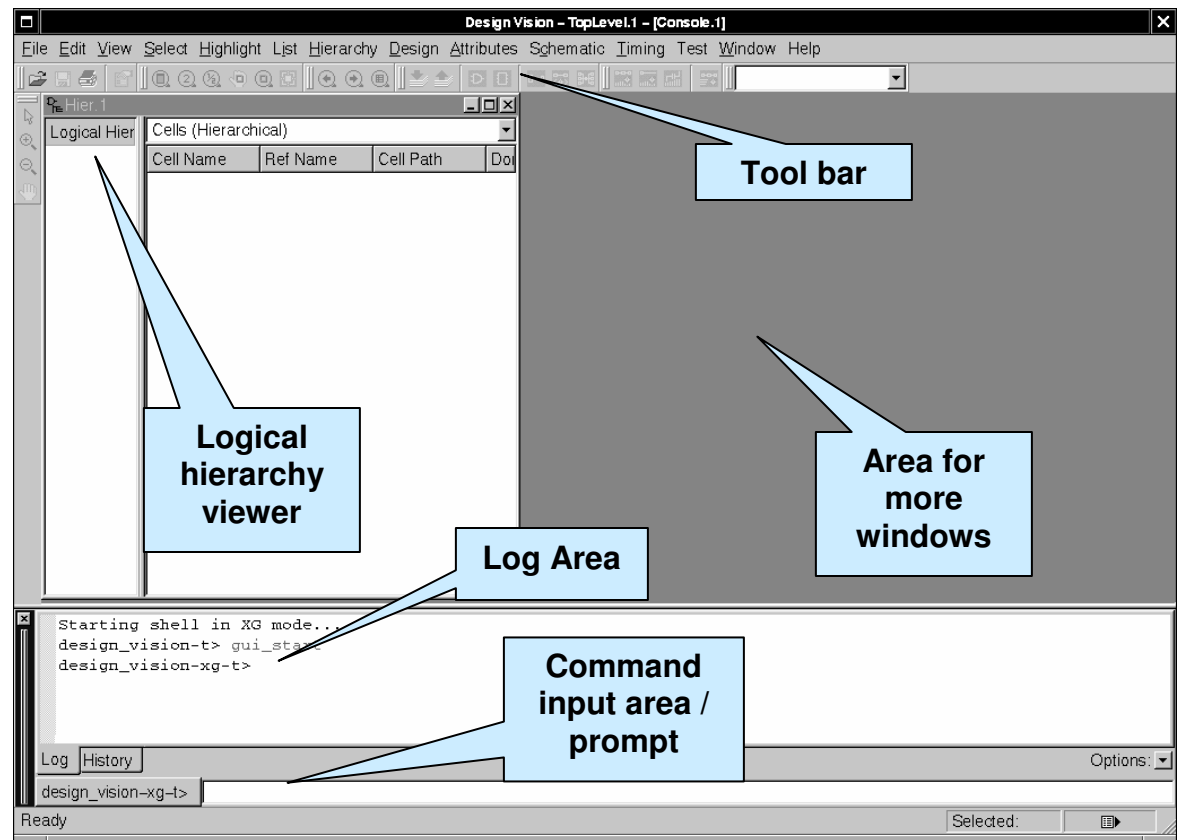
2. You are provided with a `.synopsys_dc.setup` file. Incorporate the following using a text editor of your choice:
  - The technology library file name is **`../ref/db/sc_max.db`**
  - The symbol library file name is **`../ref/db/sc.sdb`**
  - The directory **`scripts`** contains the constraint file to be sourced
3. Take a look at the other commands in the rest of the file. Notice that the variables which you have edited will be echoed upon tool start-up.
4. **Save** the `.synopsys_dc.setup` file, and **Quit** the editor.

### Task 2. Invoke Design Vision

---

1. Invoke Design Vision from the **lab2** directory.

```
unix% pwd
unix% design_vision
```



2. View the *Log Area* at the bottom of the GUI.

This area displays all the executed commands, their results, and shows any error messages.

Scroll to the top of the log messages.

You should see the values of the four key `.synopsys_dc.setup` variables echoed here, followed by the command `gui_start`, which invokes the GUI.

3. Choose menu **File → Setup** and verify that the libraries are set up correctly.

**Question 1.** What is the *Link library*?

.....

**Question 2.** What is the *Target library*?


.....

**Question 3.** What is the *Symbol library*?

.....

**Note:** If the libraries are called **your\_library.db**, (instead of **sc\_max.db**) this indicates you invoked DC from the wrong Unix directory. Exit the GUI (**File → Exit → OK**, or type **exit** at the command prompt, and choose **OK** when prompted). Make sure your current directory is **lab2** and re-invoke Design Vision.

**Note:** Check your answers against the **Answers/Solutions** section at the end of this lab, and fix your setup file accordingly. If you are stuck, compare your setup file with that provided in the **.solutions** directory.  
If you edit the **.synopsys\_dc.setup** file it is recommended that you exit Design Vision and re-invoke it.

4. Select the  icon to the right of the *Search path* field. This opens up the *Set Search Path* window, which shows an expanded list of each search path directory. Make sure that the first four entries at the top are as follows:

```
.
/<tool_installation_directory>/libraries/syn
/<tool_installation_directory>/dw/syn_ver
/<tool_installation_directory>/dw/sim_ver
```

**Note:** If you do NOT see the above directories, this implies that you have over-written the default setting of the search path variable, instead appending to it. Exit the GUI (**File → Exit → OK**, or type **exit** at the command prompt, and choose **OK** when prompted). Correct the **.synopsys\_dc.setup** file, re-invoke Design Vision and verify the new settings.

## Lab 2

**Question 4.** What user directories have been added to the *Search path*?

.....

5. Click **Cancel** to close the Application Setup window.
6. In the original Unix window from which you invoked Design Vision (the shell interface), type the following at the DC prompt to confirm the library setup variables, the search path and the user as well as default aliases.

**Note:** Command line editing allows for command, option, variable and file completion. Type a few letters and then hit the **[Tab]** key.


```
printvar target_library
printvar link_library
printvar symbol_library
printvar search_path
alias
```



### Task 3. Read the Design into DC Memory

---

Design Compiler can read VHDL, Verilog, as well as SystemVerilog RTL files.

1. Click on the Read button  at the top left of the GUI (or **File → Read**).
2. In the dialog box that appears, double-click on the directory `rtl/`, and then again on `TOP.vhd`.

In the “Logical Hierarchy” window on the left side of the GUI (you may need to widen the window), there is now an icon for `TOP`, which is the top-level *design* name. There are also icons for the lower-level instances or *cells*: `I_FSM`, `I_DECODE`, and `I_COUNT`.

3. Select **TOP** (single click with left mouse button), and look at the lower right corner to verify the selection.  
You should see **Design: TOP**. This ensures that your current design is properly set to the top-level design.
4. Select **File → Link Design → OK** to link the design and resolve all references. You should not see any warning or error messages in the *Log Area*.
5. Save the unmapped design in *ddc* format. Type the following in the Command Input Area or in the Unix window in which you invoked Design Vision:


```
write -hier -f ddc -out unmapped/TOP.ddc
```

6. You can also type the following non-GUI *dc\_shell* commands to see a list of designs and libraries in memory.


```
list_designs
list_libs
```

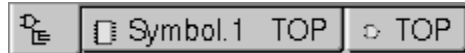
### Task 4. Explore Symbol and Schematic Views




---

1. Make sure that the lower right corner still shows that **Design: TOP** is selected. If not, select **TOP** with a single click of the left mouse button.
2. Select the **Symbol View** by clicking the  icon in the tool bar. You will see a block called **TOP**, with its input and output ports. This is referred to as the symbol view of the design, as indicated in the upper left corner of the new Window.

## Lab 2

- Now look at the **Schematic View** by clicking the  icon in the tool bar. The schematic of `TOP` contains instantiations of `FSM`, `DECODE` and `COUNT`.
- You now have three windows open in the GUI: The Hierarchical, Symbol and Schematic view windows. Maximize one of the windows. The other two windows are now also maximized, but are “behind” the window that you maximized. You can bring different windows to the foreground by selecting the appropriate tab below the view window. (You will learn how to “zoom” the image in the next task.)



- Minimize the view windows or select the left-most  Hierarchy tab to make the Hierarchical window visible.
- Explore `TOP` by visiting the **Symbol** and **Schematic Views** of the various subdesigns by selecting each design in the **Logical Hierarchy** window and clicking on the  and  icons respectively.


Because you have not compiled these designs yet, you will not see gates from the target technology library. You will see GTECH components. GTECH components are generic Boolean gates and registers that represent the generic, non-technology specific functionality of a design.

### Task 5. Explore the Mouse Functions

---

- Click and hold** the **right mouse button** in a **schematic view** to see the available mouse functions.
- Select **Zoom Fit All** with the left mouse button to maximize the view. Now repeat **Step 1** and select **Zoom In Tool**. With the left mouse button click and drag the rectangular area you want to zoom into. Press the **[ESC]** key to exit out of the ZOOM mode. Return to **Zoom Fit All** or **Zoom Out Tool** by using the appropriate mouse function.
- A quicker way to zoom in and out is using “strokes”. Press and hold the middle mouse button on the lower left corner of the rectangle you want to zoom into, then move the mouse while still pressed to the upper right corner. Only then release the middle mouse button. You just performed a zoom in gesture or stroke. By pressing the middle mouse button in place, a menu will appear with the defined strokes. Experiment with some more strokes.
- Close all window views except the `TOP` schematic window.
- This time switch to the Schematic View of **DECODE** by double clicking on the green block labelled `DECODE` in the `TOP` schematic.

Note the cell name **I\_DECODE** in the lower right corner of Design Vision. This signifies that `I_DECODE` is the **Current Instance**.

Go back to the TOP schematic view by clicking on the  **up-arrow** button from the menu (top banner).

Repeat this step for **FSM**, and **COUNT**.

6. To select multiple objects, experiment with using your **left mouse button** and the **CTRL key**. Use the left mouse button to select the first object, then **left mouse button and CTRL key** to select additional objects. *Selected objects are highlighted in white.* Click in any black area to un-select the selected objects.

## Recall the Basic Steps in Synthesis Flow

The four steps after “read” will be performed in the upcoming tasks:

- Read and translate RTL code (`read_vhdl/read_verilog`)
- Constrain the design (`source` a constraints file)
- Synthesize the design (`compile`)
- Generate reports (`report_*`)
- Save the resulting netlist (`write`)

## Task 6. Constrain TOP with a Script file

---

1. Open the **Symbol** view for TOP.

You may also view the Schematic view but the Symbol view gives you a clearer overview of your port names.

2. Type the following at the command prompt on the bottom of the Design Vision window. Remember to take advantage of command completion by hitting the tab key.

```
source TOP.con
```

**Note:** If the `source` command gives an error message, make sure that the `./scripts` directory has been appended to the `search_path` variable in the `.synopsys_dc.setup` file, or, type `source scripts/TOP.con` if you do not want to re-invoke the Design Vision.

This will execute a script file, which constrains the TOP design. You will not be able to “see” the constraints in the view window, but they are there. In

## Lab 2

upcoming labs you will learn how to generate reports to verify constraints that have been applied to a design.

The message “*Defining new variable 'lib\_name' “ is expected – this is a user-defined variable that is created in the constraints file to simplify the script – it does not indicate any changes to the `target_` or `link_library` variables that you defined.*”

### Task 7. Compile or Map to Vendor-Specific Gates


---

1. To compile the design, type the following command at the command prompt on the bottom of the Design Vision window:

```
compile
```

Monitor the log as *compile* progresses. You will see various tables for the different optimization phases of compile. The “AREA” column indicates the design size. The “WORST NEG SLACK” column indicates by how much the critical or worst path in the design is violating, relative to its constraint (Actual delay – Expected delay). The “TOTAL NEG SLACK” is the sum of all the violating path slacks. When the optimization reaches a point of “diminishing returns”, or, the slack numbers reach zero, which means that there are no violating timing paths in the design, the *compile* ends.

2. Explore the **Schematic View** of `DECODE`, `FSM` and `COUNT`. You will now see gates from the target technology library.

Use the right mouse zoom functions, the middle mouse button “strokes”, or the buttons  on the command bar to be able to see the details of the design.

### Task 8. Generate Reports and Analyze Timing

---

1. Go to the **Symbol View** of `TOP`.
2. At the *design\_vision-xg-t* prompt, type: **rc**

`rc` is an alias that was specified in the `.synopsys_dc.setup` file. It executes the following command:

```
report_constraint -all_violators
```

If there are any timing and/or area violations the report summarizes them.

You can also get more detailed timing information by generating a timing report: **rt**

By default, `report_timing` shows the timing of the critical path.

Record the following information:

**Worst Timing Violation:** Slack (VIOLATED) \_\_\_\_\_

3. Generate an area report, **ra**, and record the following:

**Total Area:** \_\_\_\_\_

4. Go back to the **Schematic View** of TOP.
5. Choose menu **Highlight → Critical Path** (CTRL-H).


The critical path, (the path with the largest violation), will be highlighted. Push in and out of the hierarchy to follow this critical path.

**NOTE:** The highlighted path should agree with the timing report path.

To undo the highlighting select:

**Highlight → Clear All** (CTRL-M).




6. Locate the histogram buttons  at the top of the window. Hover your mouse over the middle button – a “tool hint” should display “Create endpoint slack histogram”.
7. Click on the **Create endpoint slack histogram** button, and in the dialog box that appears, select **OK**.

You will see a histogram window displaying several “bins”. Each bin represents a number of timing paths. You can click on the bins, and Design Compiler will list some details of the paths contained in this bin (Timing Slack and Endpoint).

A “green” bin indicates that all timing paths within that bin meet timing. A “red” bin indicates violating paths.

When a bin is selected, it turns “yellow”

8. Select one of the bins (left mouse button), and on the right side, select one of the endpoints (left mouse button).
9. You can show the path to the selected endpoint in the schematic, as follows:

First select the left-most  “**Create PathSchematic of Selected Logic**” button – this shows the endpoint. Now select the middle “**Add Paths to Path Schematic**” button – this adds the path leading up to the endpoint. You can optionally select the right-most “**Add Fanin/Fanout to Path Schematic**” button to include fan-ins or fan-outs along the path.

### Task 9. Save the Optimized Design

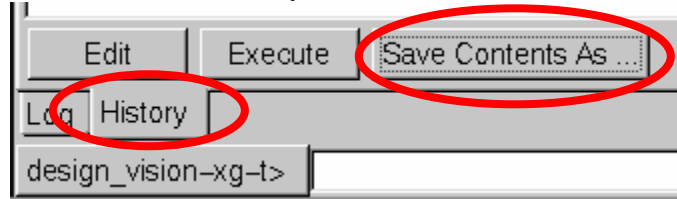
---

After compile, or after any major steps, it is advisable to save the design. The native Design Compiler XG-mode format is *ddc.*, both for unmapped or mapped (compiled) designs.

1. Go back to the Symbol View of `TOP`.
2. Choose menu **File → Save As**.
3. Double click on the *mapped/* directory.
4. Enter `TOP.ddc` in the *File name* field.
5. Verify that the **Save All Designs in Hierarchy** button is selected. This will save the entire design hierarchy into a single (.ddc) file.
6. Click **Save**.

You just saved the gate-level netlist (the entire hierarchy) in ‘ddc’ format under the *mapped* directory. You can verify that the file was created in the original Unix window from which you invoked Design Vision, using ‘`ls -l mapped`’.

7. Next, select the history tab.



8. Save the command history by selecting the button “Save Contents As” and specifying the name *run\_history.tcl* into the *scripts* directory.

## Task 10. Remove Designs and Exit Design Vision

---

1. Type *fr* at the command line prompt to remove all designs from DC memory.

Verify that all the icons in Design Vision have been deleted. The “*fr*” alias executes the following command:

```
remove_design -designs
```

You can also use the pull down menu **File → Remove All Designs**.

2. In the original Unix window from which you invoked Design Vision, type *h*. This will list a history of all commands you have executed since you started Design Vision.

**Note:** If you unintentionally exit out of Design Vision, you can recreate everything you did up to that point by executing the command log file that has been created in your project working directory (lab2), by doing the following:

```
unix% cp command.log lab2.log
unix% design_vision -f lab2.log
```

**Note:** Alternatively, you can “clean” the *run\_history.tcl* file (if necessary) and use it instead of the command.log file:

```
unix% design_vision -f scripts/run_history.tcl
```

3. Exit from Design Vision. Use the menu sequence **File → Exit → OK**, or type **exit** at the command prompt, and choose **OK** when prompted.

### Task 11. Browse Documentation on SolvNet

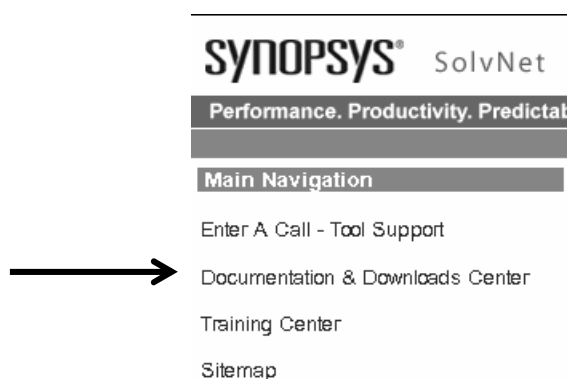
---

In this task we will pretend that we want to learn how to use Design Vision to print a design schematic. We will browse the on-line documentation to find the needed information.

1. Open a web browser and follow the link from *[solvnet.synopsys.com](http://solvnet.synopsys.com)* to log on to SolvNet (the link to SolvNet is also available from the Synopsys home page).

```
unix% firefox &
```

2. From the “*Main Navigation*” menu, click on the “*Documentation & Downloads Center*”



3. From the “*Documentation & Downloads Center*” click “*Documentation on the Web*”



4. From the resulting “Documentation” product list, select **Design Vision**.



[Design Budgeting](#)  
[Design Compiler®](#)  
[Design Compiler® FPGA](#)  
 → [Design Vision™](#)

5. In the next window – select the PDF icon to be able to browse or download the user guide for Design Vision.

#### Design Vision

- [Design Vision User Guide, version Z-2007.03](#) 

6. Once the user guide is open, go to chapter on “Performing Basic Tasks”. And then to [Printing Schematic and Symbol Views](#).

You do not need to print anything - this is just one example of using the online documentation.

## **Task 12. Pre-building the “alib” directory for next labs**

Before taking a break (or returning to lecture) execute the following command from UNIX, which invokes DC and builds an “alib” directory. This step will take a few minutes. Once completed it will automatically exit DC.

```
unix% ./make_alib
```

In some of the upcoming labs you will be invoking the more powerful `compile_ultra` synthesis command (instead of `compile`), which will be discussed in a later lecture. This command first builds an optimized version of your technology library file and stores it in a directory called “*alib*”. This library optimization step only occurs during the first `compile_ultra` and can take several minutes to complete. Once the “*alib*” directory is built, subsequent compiles simply use the existing library, and hence run faster. In order to save compile time in the upcoming labs, you are pre-building the “*alib*” directory now.

**Congratulations – this completes Lab 2.**

# Answers / Solutions

**Question 1.** What is the *Link library*?

\* sc\_max.db

**Question 2.** What is the *Target library*?

sc\_max.db

**Question 3.** What is the *Symbol library*?

sc.sdb

**Question 4.** What user directories have been added to the *Search path*?

../ref/db ../scripts

# 4

## Timing and Area Constraints

### Learning Objectives

After completing this lab you should be able to:

- Determine the unit of time used in the target library
- Create a Design Compiler timing and area constraints file based on a provided schematic and specification
- Verify the syntax of the constraints prior to applying them to a design
- Apply the constraints to a design
- Validate the completeness and correctness of the applied constraints

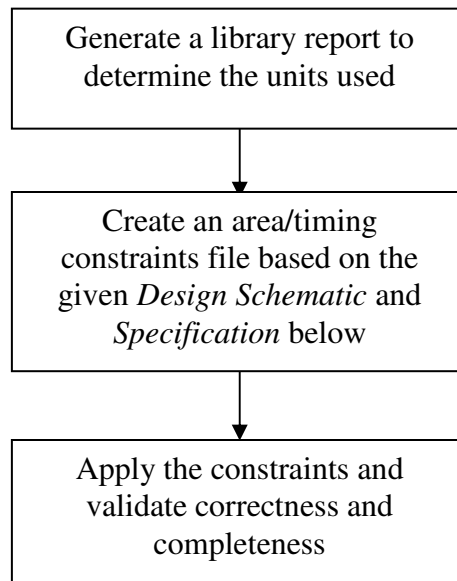


**Lab Duration:**  
**80 minutes**

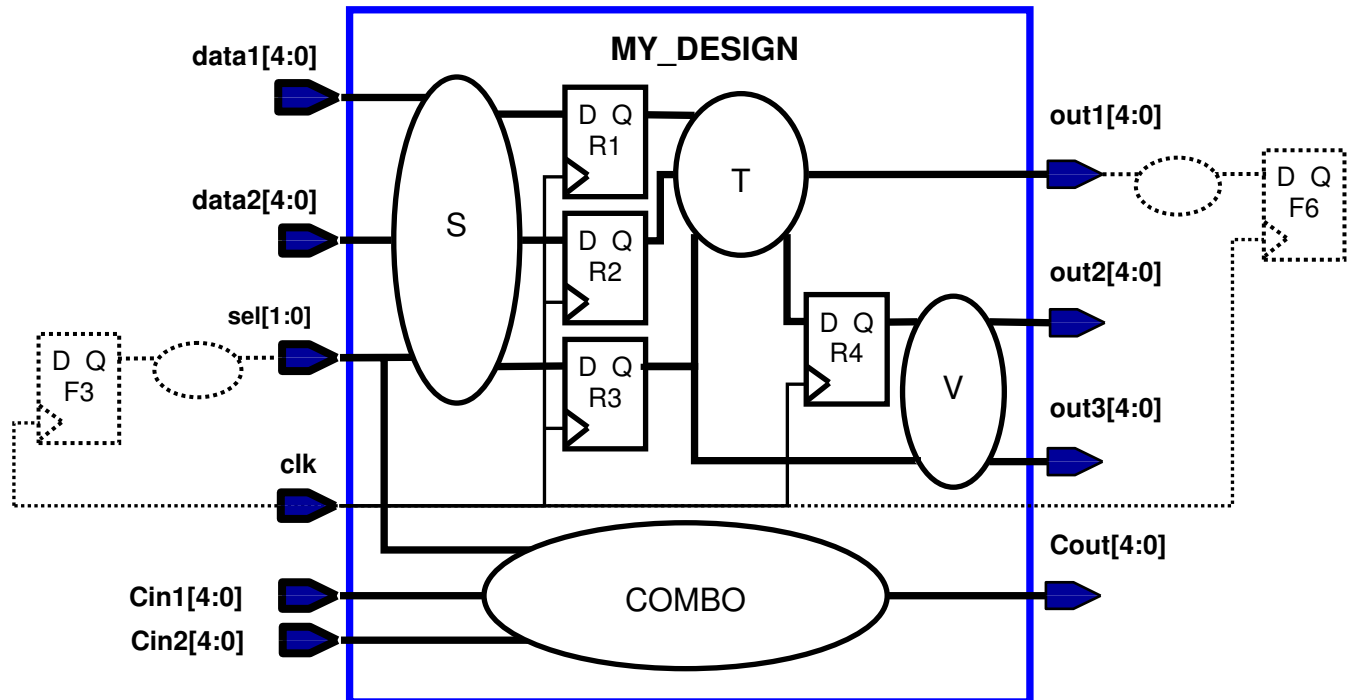
## Lab 4

### Lab Flow

Follow the step-by-step **Lab Instructions** on the following pages to perform the three tasks shown in the lab flow below. Refer to the **Design Schematic** and **Design Specification** sections are needed.



## Design Schematic



## Design Specification

Hint: **Read carefully!** Some of the specifications are described in “non-DC” language or terms, requiring translation and calculation to derive the DC constraints.

<b>Clock Definition</b>	<ol style="list-style-type: none"> <li>1. Clock <i>clk</i> has a frequency of <b>333.33 Mhz</b>.</li> <li>2. The maximum external clock generator delay to the clock port is <b>700ps</b>.</li> <li>3. The maximum insertion delay from the clock port to all the internal and external register clock pins is <b>300ps +/- 30ps</b>.</li> <li>4. The clock period can fluctuate <b>+/- 40ps</b> due to jitter.</li> <li>5. Apply <b>50ps</b> of “setup margin” to the clock period.</li> <li>6. The worst case rise/fall transition time of any clock pin is <b>120 ps</b>.</li> </ol>
<b>Register Setup Time</b>	Assume a maximum setup time of <b>0.2ns</b> for any register in MY_DESIGN
<b>Input Ports (sequential logic)</b>	<ol style="list-style-type: none"> <li>1. The maximum delay from ports <i>data1</i> and <i>data2</i> through the <i>internal input logic S</i> is <b>2.2ns</b>.</li> <li>2. The latest <i>F3 data arrival time</i> at the <i>sel</i> port is <b>1.4ns</b> (absolute time).</li> </ol>
<b>Output Ports (sequential logic)</b>	<ol style="list-style-type: none"> <li>1. The maximum delay of the <i>external</i> combo logic at port <i>out1</i> is <b>420ps</b>; <i>F6</i> has a setup time of <b>80ps</b>.</li> <li>2. The maximum <i>internal</i> delay to <i>out2</i> is <b>810ps</b></li> <li>3. The <i>out3</i> port has a <b>400ps</b> setup time requirement with respect to its capturing register data input.</li> </ol>
<b>Combinational Logic</b>	The maximum delay from <i>Cin1</i> and <i>Cin2</i> to <i>Cout</i> is <b>2.45ns</b> .
<b>Design Area</b>	The maximum design area goal is <b>540</b> area units

# Lab Instructions

## **Task 1. Determine the Target Library's *Time Unit***

---

1. Change to the *lab4* UNIX directory.
2. Using a text editor or viewer look at the *.synopsys\_dc.setup* file to answer this question:

**Question 1.** What is the *target library file* name?

.....

3. Invoke Design Compiler from the *lab4* directory:

```
UNIX% dc_shell-t
```

4. Read the above *target library file* into DC:

```
dc_shell-xg-t> read_db <target_library_FILE>
```

5. Determine the *library name* associated with this library file:

```
dc_shell-xg-t> list_libs
```

**Question 2.** What is the target *library name*?

.....

6. Generate a library report file for the above library:

```
redirect -file lib.rpt {report_lib <library_NAME>}
```

7. Exit Design Compiler:

```
dc_shell-xg-t> exit
```

## Lab 4

8. Use a text editor or viewer to look at the top portion of the *lib.rpt* file:

**Question 3.** What is the “Time Unit” of the target library?

.....

9. Exit the text editor or viewer.

### **Task 2. Create a Timing and Area Constraints File**

---

1. In the *scripts* directory use a text editor to create a new file called *lab4.con*.

**Question 4.** What is the recommended first command for any constraint file?

.....

2. Using the **Design Specification** and **Design Schematic** on the previous pages, as well as the appropriate time unit, enter the required constraints in *lab4.con*.

**Note:** You are encouraged to use the *Job Aid* as needed. You can also use DC’s **help** and **man** commands as needed.

**Note:** To use DC’s **help** and **man** you will need to invoke the DC shell. It is also possible to access the Design Compiler “man” pages without invoking DC: The recommended approach is to create a separate UNIX alias, **dcman**, for example:

```
UNIX% alias dcman "/usr/bin/man -M $SYNOPTSYS/doc/syn/man"
UNIX% dcman create_clock
```

3. After completing the constraints file, check your constraint syntax, and correct as necessary:

```
UNIX% dcprocheck scripts/lab4.con
```

**Note:** *dcprocheck* is a syntax checking utility that is included with the Design Compiler executable. It is available once you are able to launch Design Compiler – no additional user setup is required. If used to check a “run script”, you may ignore the warning “read\_verilog is unknown” – *dcprocheck* prefers the command `read_file -f verilog`.



### Task 3. Apply Constraints and Validate

---

1. Invoke the DC shell from the *lab4* directory.
2. Read, link and check the design *rtl/my\_design.v*.
3. Apply the constraints file and make any corrections as needed.

```
source scripts/lab4.con
```

4. Check that there are no missing or conflicting key constraints – correct as needed:

```
check_timing
```

5. Check that the constraints were correctly applied to the design – correct as needed:

```
report_clock  
report_clock -skew  
report_port -verbose
```

6. Write out the applied constraints, in expanded form, to a file for further checking:

```
write_script -out scripts/lab4.wscr
```

7. Ensure that your constraints are complete and correct by performing a UNIX “diff” between your write-script file, and the provided solution file – correct as needed:

```
tkdiff scripts/lab4.wscr .solutions/lab4.wscr  
OR  
diff scripts/lab4.wscr .solutions/lab4.wscr
```

8. If the above “diff” command uncovers differences which you do not understand, take a look at *.solutions/lab4.con*: This file contains comments explaining how each constraint was determined.
9. Save the design as *unmapped/MY\_DESIGN.ddc* and exit Design Compiler.

## Answers / Solutions

**Question 1.** What is the *target library* file name?

*sc\_max.db*

**Question 2.** What is the target *library name*?

*cb13fs120\_tsmc\_max*

**Question 3.** What is the “Time Unit” of the target library?

1ns

**Question 4.** What is the recommended first command for any constraint file?

```
reset_design
```

# 5

## Partitioning for better synthesis results

### Learning Objectives

After completing this lab, you should be able to:

- Improve a design's QoR (Quality of Results = Timing and/or Area) by repartitioning a design using the **group** and **ungroup** commands
- Perform basic analysis and manipulation using DesignVision

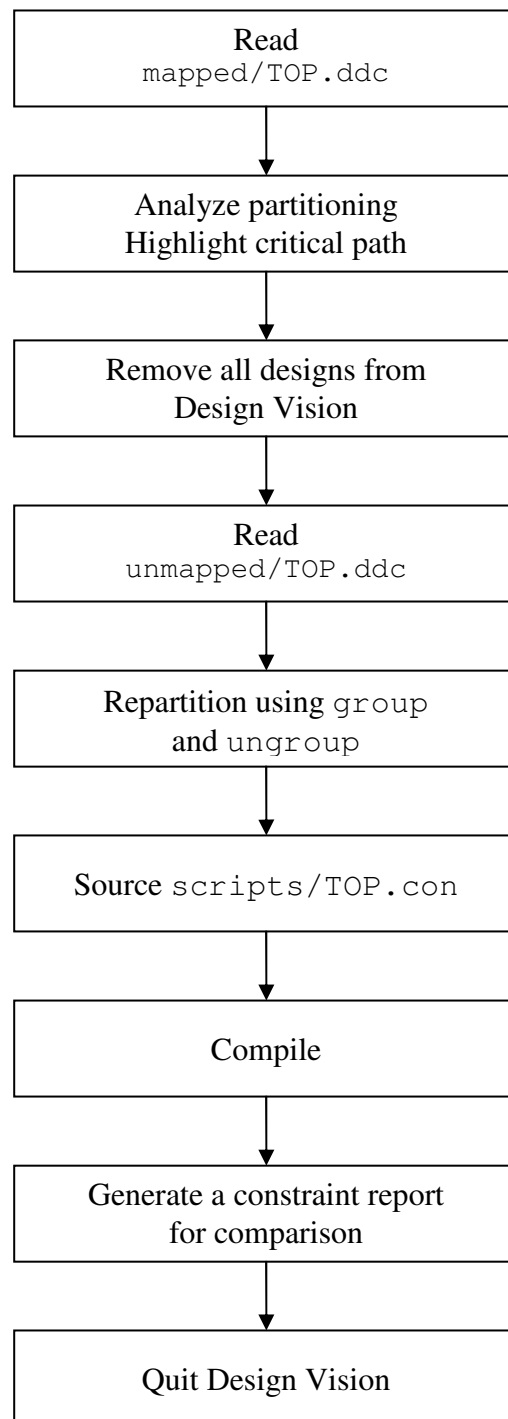


**Lab Duration:**  
30 minutes

## Lab 5

### Lab Flow

Follow the detailed step-by-step **Lab Instructions** on the following pages to perform the steps highlighted in this flow:



# Lab Instructions

## **Task 1. Analyze Partitioning of a Compiled Design**

1. Change to the **lab5** directory, then invoke Design Vision.
2. **Read** and **link** the design from **TOP.ddc** located in the **mapped** directory.
3. Generate the default “End Point Slack” histogram within DesignVision.

**Timing → Endpoint Slack → OK** (Accept defaults)

Write down the “Worst” slack amount from the histogram (Look under the Left most bar (Red = Negative slack) of the histogram:

**Max Delay:** Largest Violation (Slack) \_\_\_\_\_

Write down the Total Cell area by using `report_area`

**Total cell area:** \_\_\_\_\_

4. Go to the Schematic View of **TOP** and highlight the critical path.  
Push into the subblocks to see the critical path start and end points. Notice that the path traverses a purely combinational block.
5. In the space below, draw the block diagram for **TOP**, and indicate where the critical path start and end points are. Think about which partitioning guidelines the design violates.

How can you improve the partitioning of this design?

Original Partitioning	After Re-Partitioning

6. **Remove all the designs from DesignVision memory** (File → Remove All Designs in GUI (or) **fr** from the command line)

### Task 2. Repartition the Unmapped Design

---

1. Read and link the unmapped design **unmapped/TOP.ddc**. (Re-invoke Design Vision incase you exited the tool in the previous task)
2. From the Logical Hierarchy View, select both sub-designs with the instance names **I\_DECODE** and **I\_COUNT** (**CTRL + Left** mouse click).
3. To confirm your selection, type the following at the command prompt:

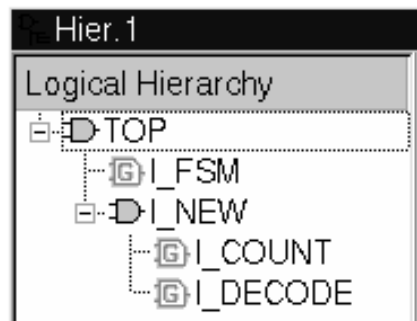
```
get_selection
```

The command should return → {I\_COUNT I\_DECODE}.

4. Group these two designs together with a new design name, **NEW** and a new instance name, **I\_NEW**, by typing the following:

```
group -design NEW -cell I_NEW [get_selection]
```

5. Validate this last step as follows: In the Logical Hierarchy View, you should see the following.



Open up a Schematic View of TOP to see the new instance **I\_NEW** that was just created.

You can also type the following to report the design hierarchy:

```
report_hierarchy -noleaf
```

6. The next step is to ungroup everything below level two in the hierarchy for the new design **NEW**. Do this by typing the following.

```
ungroup -start_level 2 I_NEW
```

Confirm this step using the same techniques as before. Your design hierarchy should now be the desired, optimum hierarchy for synthesis.

### Task 3. Compile and Analyze Results

---

1. Go to the Symbol or Schematic View of TOP.

Look at the bottom of the Design Vision window to make sure the current\_design is TOP.

2. Source the script file **scripts/TOP.con**.
3. Perform “compile” on TOP.
4. Generate a constraint report for All Violations (**rc**)

From Task-1 record the Delay and Area information into the table below and the same for this current design after compile. If no timing violations are reported, the design meets its timing constraints.

Task	Compiled Design	Timing Slack	Area
Task 1	Initial partitioning		
Task 3	After partitioning (group + ungroup)		

5. Highlight the critical path using Ctrl-h

Does the critical path cross any purely combinational blocks? \_\_\_\_\_

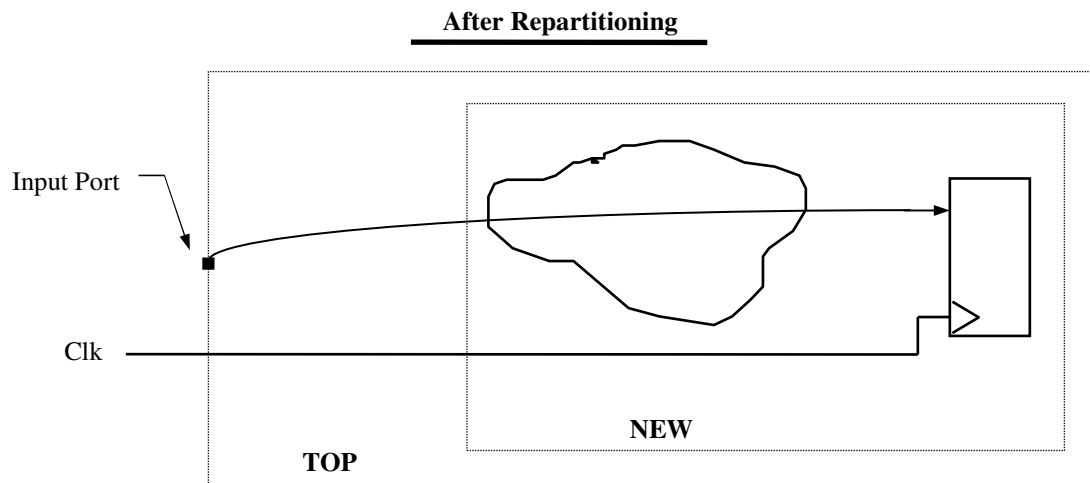
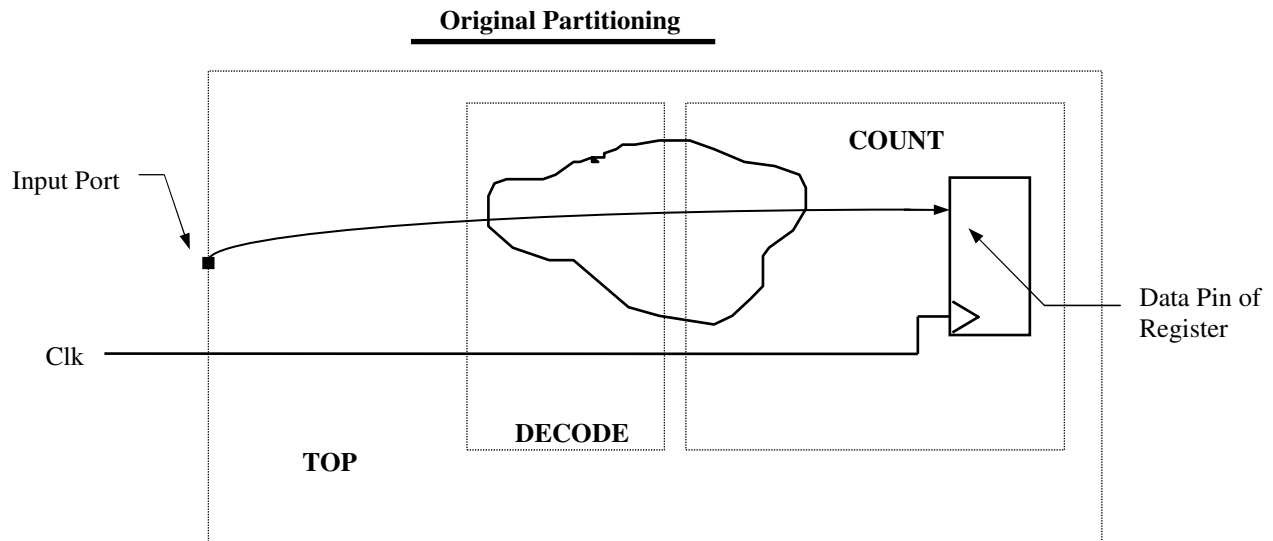
6. Quit Design Vision. There is no need to save this design, you will not be using it again.
7. Using the results from the table above, Did repartitioning (group + ungroup)
  - a) Improve Timing ? \_\_\_\_\_ By how much ? \_\_\_\_\_
  - b) Improve Area ? \_\_\_\_\_ By how much ? \_\_\_\_\_

Note that the results **are** design and constraints **dependent**.

#### Question 1.

## Answers / Solutions

### Answer to Block Diagram



**Note:** There is a solution script in `.solutions/run.tcl`



# 6

## Environmental Attributes

### Learning Objectives

After completing this lab you should be able to:

- Determine the available and appropriate wireload model and operating conditions model to use
- Define Design Compiler environmental attributes based on a provided schematic and specification
- Apply the attributes to a design
- Verify the applied attributes

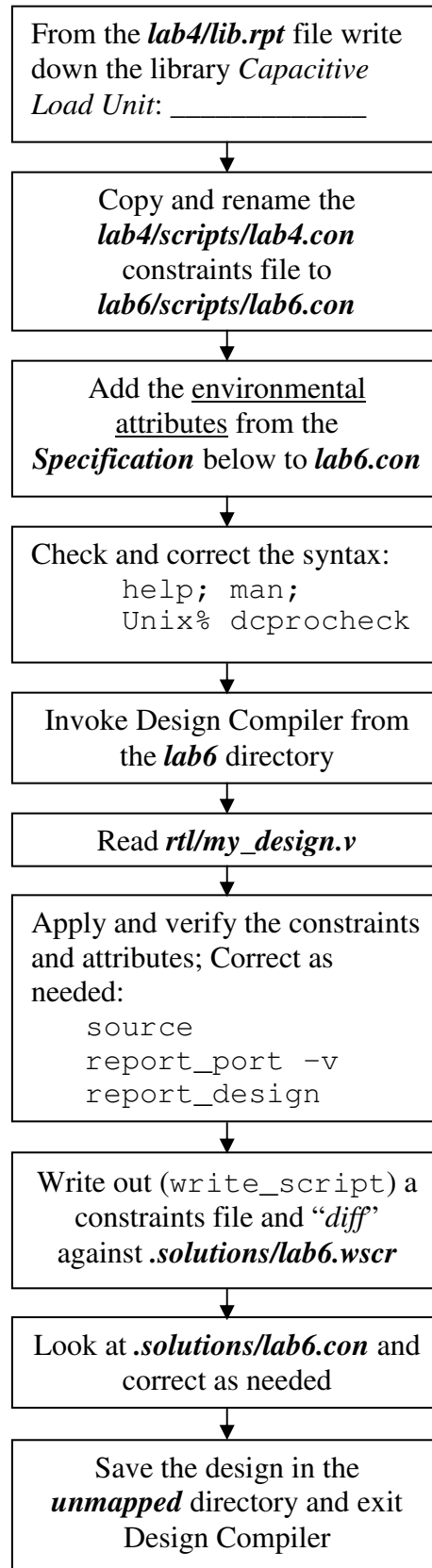


**Lab Duration:**  
**45 minutes**

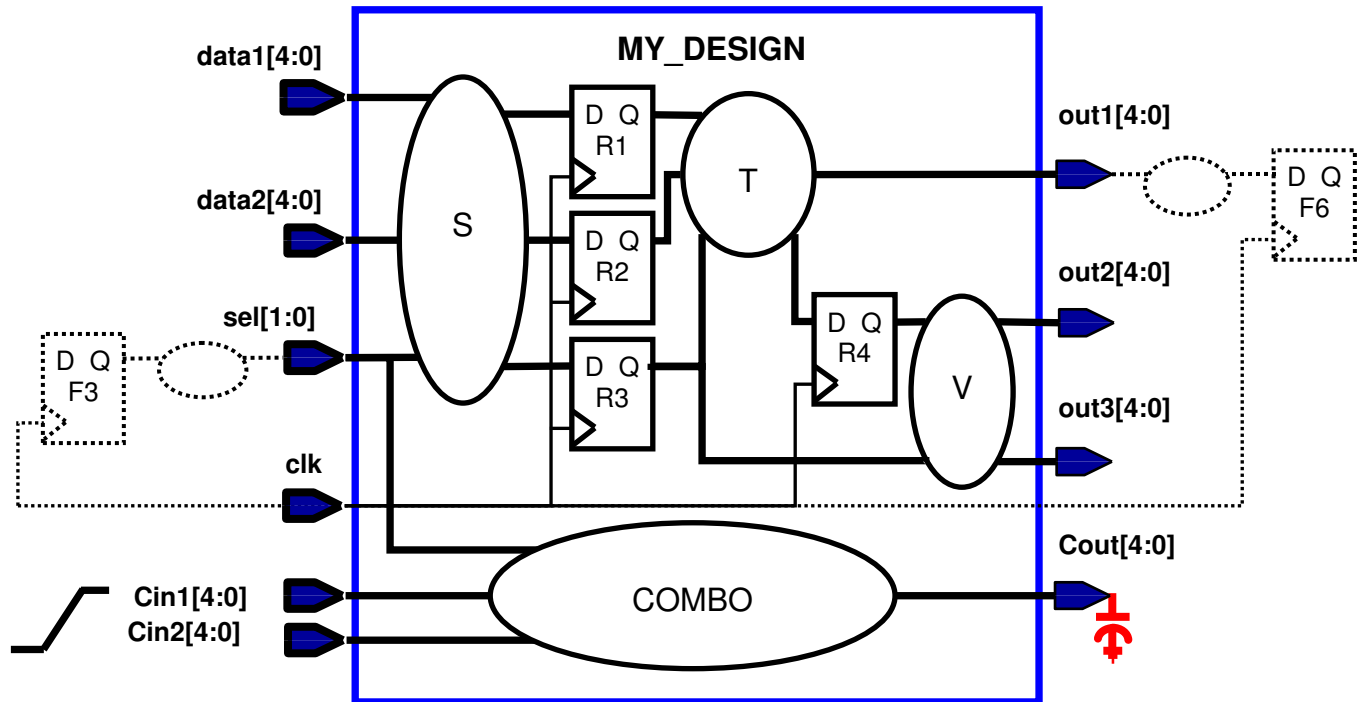
## Lab 6

# Lab Instructions

Perform the steps below. Refer to the *Lab 4* step-by-step instructions as needed.



# Design Schematic



## Design Specification

Hint: Use the *lib.rpt* file in *lab4* to obtain some of the information needed to apply these specs.

<b>Input Ports (drivers)</b>	<ol style="list-style-type: none"> <li>1. Specify a drive on all inputs, except <i>clk</i> and <i>Cin*</i>, using the buffer <i>bufbd1</i> in the library</li> <li>2. The <i>Cin*</i> ports are chip-level inputs and have a <b>120ps</b> maximum input transition.</li> </ol>
<b>Output Ports (loads)</b>	<ol style="list-style-type: none"> <li>1. All outputs, except <i>Cout</i>, drive a maximum load equivalent to <b>2</b> times the capacitance of the “<i>T</i>” pin of the cell <i>bufbd7</i>.</li> <li>2. The <i>Cout</i> port drives a maximum load of <b>25 fF</b>.</li> </ol>
<b>Wireload Model</b>	<p>“Manually” apply a wire load model appropriate for this design’s size (Hint: Look at the area constraint for the design in conjunction with the “<i>Wire Loading Model Selection Group</i>” table in the library to find the right model; Remember to <u>disable</u> <i>automatic wireload selection</i>)</p>
<b>Operating Conditions</b>	<p>There is only one <i>Operating Condition</i> model available in the library. Apply that model.</p>

# 9

## More Constraint Considerations

### Learning Objectives

After completing lab 9A you should be able to:

- Constrain any single-clock (single-cycle and dual-phase) design given the schematic and specification
- Compile the constrained design
- Identify constraints in a timing report

After completing the optional lab 9B you should be able to:

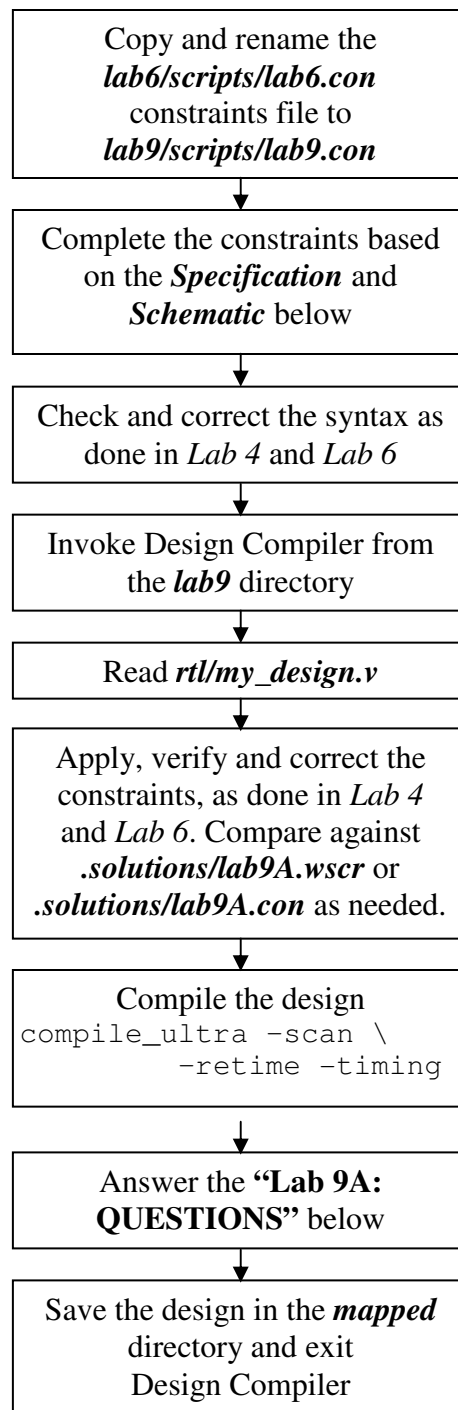
- Isolate any output port that fans out internally to the design, from external loads



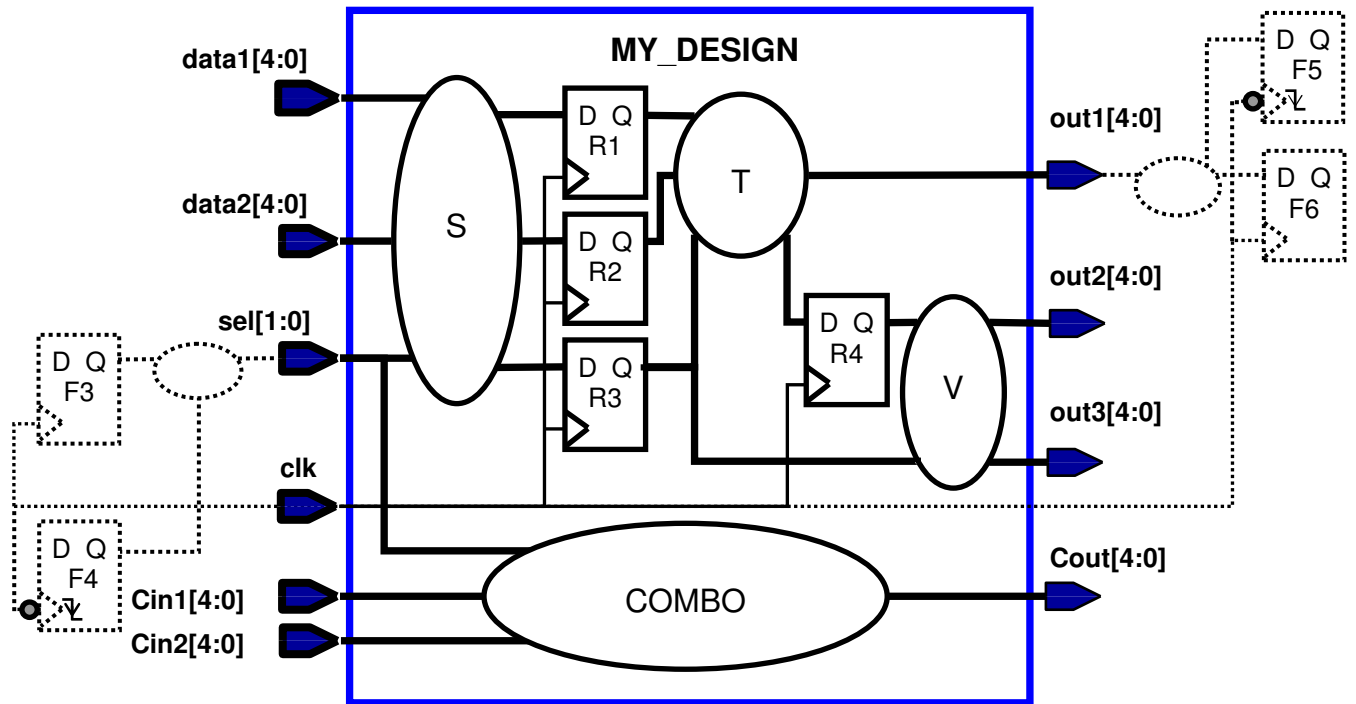
**Lab Duration:**  
**90 minutes**

# Lab 9A Instructions

Perform the steps below.



## Lab 9A: Design Schematic



## Lab 9A: Design Specification

<b>Clock Spec</b>	<ol style="list-style-type: none"> <li>1. Rename the clock from “<i>clk</i>” to “<i>my_clk</i>”</li> <li>2. Change the duty cycle to <b>40 % active high</b>, with <b>zero offset</b>.</li> </ol>
<b>Input Port Delays</b>	<ol style="list-style-type: none"> <li>1. The <i>sel</i> port data is being supplied by an additional register, <i>F4</i>. The data at <i>sel</i> arrives no later than <b>420ps</b> after the <b>negative</b> edge-triggered launching clock edge of <i>F4</i>.</li> <li>2. The clock signal at <i>F4</i> has a <b>600ps total (external + internal) insertion delay</b> from its clock source</li> </ol>
<b>Output Port Delays</b>	<ol style="list-style-type: none"> <li>1. The <i>out1</i> port data is captured by an additional register, <i>F5</i>. The data at <i>out1</i> must arrive no later than <b>260ps</b> before <i>F5</i>’s <b>negative</b> edge-triggered clock edge.</li> <li>2. The clock signal at <i>F5</i> has an <b>internal latency of 500ps</b> after the external <i>my_clk</i> clock generator delay</li> </ol>
<b>External Fanout, Input Ports</b>	<p>Each input port (except <i>clk</i>) fans out to <b>2 other sub-blocks</b>, which each drive the equivalent of <b>3 <i>bufbd1</i></b> (input pin <i>I</i>) buffers internally.</p> <ol style="list-style-type: none"> <li>1. Model the external <b>capacitive loading</b></li> <li>2. Model the external <b>fanout</b></li> </ol>
<b>External Fanout, Output Ports</b>	<p>Each output port is assumed to drive <b>3</b> other sub-blocks. The pin capacitance of these blocks has already been specified. Model the external <b>fanout</b> on the output ports.</p>
<b>Wire Load: Sub-Design</b>	<ol style="list-style-type: none"> <li>1. Apply the wireload model <i>ForQA</i> to sub-designs <i>ARITH</i> and <i>COMBO</i></li> <li>2. Set the wire load mode for the entire design to <b>enclosed</b>.</li> </ol>
<b>Wire Load: Ports</b>	<p><i>MY_DESIGN</i> is a sub-block of a chip which uses the WLM called <b>16000</b>. Apply this WLM to all ports of <i>MY_DESIGN</i>.</p>



## Lab 9A: QUESTIONS

1. Generate a constraint report:

```
report_constraint -all_violators
```

**Question 1.** Does the design have any timing, area and/or DRC violations?

.....

2. Generate an area report:

```
report_area
```

**Question 2.** Is it possible to know, just by looking at this report, how the actual design area compares to the applied max area constraint?

.....

3. Generate a more targeted constraint report to find the area “slack”. The `-verbose` option supplies more details about each constraint type (*max delay*, *min delay*, *max cap/tran/fanout*, and *max area*), while `-max_area` limits the report to just that constraint:

```
report_constraint -ver -max_area
```

4. Generate a timing report for the path to the **out1** output. Include options to show net *transition times* and *net delays* to 6 decimal places, as well as net *fanout* (`-trans -input -sig 6 -nets`). Use the report to locate and fill in the requested data. This data should match your constraints:

**Question 3.** What is the startpoint of the reported path?

.....

**Question 4.** From the report identify the following startpoint values.  
Include the constraint file command(s) which produce each of the report values:

Clock *my\_clk*

Launch edge time \_\_\_\_\_ Rising or falling? \_\_\_\_\_

Command(s) .....

.....

Clock network delay = .....

Command(s) .....

.....

Clock pin transition time = .....

Command(s) .....

.....

**Question 5.** From the report identify the following endpoint values.  
Include the constraint file command(s) which produce each of the report values:

Clock *my\_clk*

Capture edge time \_\_\_\_\_ Rising or falling? \_\_\_\_\_

Command(s) .....

.....

Clock network delay = .....

Command(s) .....

.....

Clock uncertainty = .....

Command(s) .....

.....

Output external delay = .....

Command(s) .....

.....

**Question 6.** Why is the output timing with respect to a falling clock edge?

.....

**Question 7.** Why are the report values for *uncertainty* and *output external delay* the negative of their corresponding constraint values?

.....

**Question 8.** How does the net delay on the output port compare to the internal net delays? What is the likely reason for such a difference?

.....

5. Generate a timing report for the path from the *sel* input to the *Cout* output. Include options to show *transition times* and *net delays* to 6 decimal places. Use the report to locate and fill in the requested data. This data should match your constraints:

**Question 9.** From the report identify the following startpoint values. Include the constraint file command(s) which produce each of the report values:

Clock *my\_clk*

Launch edge time \_\_\_\_\_ Rising or falling? \_\_\_\_\_

Command(s) .....

.....

## Lab 9

Clock network delay = .....

Command(s) .....

.....

Input external delay = .....

Command(s) .....

.....

Transition time of *sel* port = .....

Command(s) .....

.....

**Question 10.** What is causing the “Incr” (incremental) delay on the *sel* input port?

.....

**Question 11.** How can you verify that the *COMBO* path from *Cin\** to *Cout* is constrained to the required 2.45ns spec?

.....

6. Save the design in *mapped* and exit Design Compiler.

**Congratulations!** You are now able to create, verify and identify timing, area and environmental constraints for any single clock design given the schematic and specification.

**If you have some extra time left you can try the optional part 9B.**

## Lab 9B: OPTIONAL – Port Isolation

In this lab you will see the potential problem of having output ports which are connected to part of an internal timing path. You will then fix the problem by isolating the output ports from the internal path.

### Lab 9B Instructions

1. Invoke Design Vision from the *lab9* directory.
2. Read the *mapped/ISOLATE\_PORTS.ddc* design. The design has already been constrained and compiled.
3. Bring up the schematic view and examine the path that goes to the output ports. Notice that the two output ports are connected to each other, and the output net also fans out internally to another gate input.

**Question 12.** Does the design have any violations?

.....

4. With a text editor open the constraints file used for this design, *scripts/isolate\_ports.con*.

Notice that the output ports each have a 30fF load applied to them. It turns out that this output load is an estimate. The load could be larger or smaller. Let's find out next what happens if we increase the load.

5. Change the output load to **60fF** and save the file.
6. Apply the updated constraints to the design:

```
source scripts/isolate_ports.con
```

## Lab 9

7. Generate a default timing report and highlight the critical path (CTRL-H):

**Question 13.** Describe the resulting violating path:

.....  
.....

**Note:** Since the output ports are connected to one of the nets which is part of the loop-back path, an increase in the external output loading is now causing an internal register-to-register path violation!

**Question 14.** What command can be used to “isolate” the output ports from internal paths (allowing the use of inverters)?

.....

8. Edit the *isolate\_ports.con* file again: Change the output load back to **30fF**, and include the port isolation command from the question above. Save the file.
9. Apply the updated constraints.
10. Generate a report to verify that the new constraint was applied to the output ports:

```
report_isolate_ports
```

11. Perform an incremental compile:

```
compile -scan -boundary -map high -incr
```

**Question 15.** Does the design have any violations?

.....

**Question 16.** What did `set_isolate_ports` accomplish?

.....

12. Save the design into the *mapped* directory and exit Design Vision.

## Answers / Solutions

**Question 1.** Does the design have any timing, area and/or DRC violations?

From `report_constraint -all` you should see that there are no timing, area or design rule violations. If you have any violations: 1) Check and correct your constraints (compare against `.solutions/lab9.con`), 2) Remove the design from memory, 3) Re-apply the constraints and 4) Re-compile the design, 5) Generate another constraint report.

**Question 2.** Is it possible to know, just by looking at this report, how the actual design area compares to the applied max area constraint?

No. The report only shows the design's actual area. It does not include, or compare it against, the max area constraint.

**Question 3.** What is the startpoint of the reported path?

The startpoint is the clock pin, *CP* or *CPN*, of a register:  
***R\_#/CP* or *R\_#/CPN***

**Question 4.** From the report identify the following startpoint values. Include the constraint file command(s) which produce each of the report values:

Clock *my\_clk*  
Launch edge time: **0.0ns, Rising**

```
create_clock -period 3.0 \
  -name my_clk -waveform {0 1.2} \
  [get_ports clk]
```

Clock network delay = **1.0ns**

```
set_clock_latency -source 0.7 \
  [get_clocks my_clk]
set_clock_latency 0.3 \
  [get_clocks my_clk]
```

Clock pin transition time = **0.12ns**

```
set_clock_transition 0.12 \
    [get_clocks my_clk]
```

**Question 5.**

From the report identify the following endpoint values.  
Include the constraint file command(s) which produce each of the report values:

Clock *my\_clk*

Capture edge time: **1.2ns, Falling**

```
create_clock -period 3.0 \
    -name my_clk -waveform {0 1.2} \
    [get_ports clk]

set_output_delay -max -0.24 \
    -clock my_clk -add_delay \
    -clock_fall \
    -network_latency_included \
    [get_ports out1]
```

Clock network delay = **0.7ns**

```
set_clock_latency -source 0.7 \
    [get_clocks my_clk]

set_output_delay -max -0.24 \
    -clock my_clk -add_delay \
    -clock_fall \
    -network_latency_included \
    [get_ports out1]
```

Clock uncertainty = **-0.15ns**

```
set_clock_uncertainty -setup 0.15 \
    [get_clocks my_clk]
```



Output external delay = **0.24ns**

```
set_output_delay -max -0.24      \  
-clock my_clk -add_delay        \  
-clock_fall                     \  
-network_latency_included       \  
[get_ports out1]
```

**Question 6.** Why is the output timing with respect to a falling clock edge?

The *out1* port is constrained by two registers – a rising (*F6*) and a falling (*F5*) edge-triggered one. DC determined that the timing constraint to the falling edge-triggered register is the more restrictive of the two.

**Question 7.** Why are the report values for *uncertainty* and *output external delay* the negative of their corresponding constraint values?

The negated values simply mean that the constraint numbers are being subtracted from the “*data required time*”.

**Question 8.** How does the net delay on the output port compare to the internal net delays? What is the likely reason for such a difference?

The net delay on the output port is a lot larger than most of the internal net delays. This is probably due to the fact that the I/O ports have a larger WLM (*16000*) assigned to them compared to the WLM of the internal nets of the design (*8000*). Also, the output ports have an external fanout of 3 (not included in the *Fanout* column) compared to the smaller internal net fanouts.

```
set_wire_load_model -name 16000  \  
[all_outputs]  
set_port_fanout_number 3          \  
[all_outputs]
```

**Question 9.** From the report identify the following startpoint values. Include the constraint file command(s) which produce each of the report values:

Clock *my\_clk*

Launch edge time: **1.2ns, Falling**

```
create_clock -period 3.0 \
    -name my_clk -waveform {0 1.2} \
    [get_ports clk]

set_input_delay -max 1.02 \
    -clock my_clk -add_delay \
    -clock_fall \
    -network_latency_included \
    -source_latency_included \
    [get_ports sel]
```

Clock network delay = **0.0ns**

```
set_input_delay -max 1.02 \
    -clock my_clk -add_delay \
    -clock_fall \
    -network_latency_included \
    -source_latency_included \
    [get_ports sel]
```

Input external delay = **1.02ns**

```
set_input_delay -max 1.02 \
    -clock my_clk -add_delay \
    -clock_fall \
    -network_latency_included \
    -source_latency_included \
    [get_ports sel]
```

Transition time of *sel* port = **~0.2 - 0.6ns**

```
set_driving_cell \
    -lib_cell bufbd1 \
    -library cb13fs120_tsmc_max \
    [remove_from_collection \
    [all_inputs] \
    [get_ports "clk Cin*"]]
```

**Question 10.** What is causing the “Incr” (incremental) delay on the *sel* input port?

This represents the additional time for the input signal with the above transition time to reach the “switching point”. It is not due to net delay, which is reported separately at the input pin of the first gate.

**Question 11.** How can you verify that the *COMBO* path from *Cin\** to *Cout* is constrained to the required 2.45ns spec?

```
report_timing \
  -from [get_ports Cin*] \
  -to [get_ports Cout]
```

Subtract the “*data required time*” (**3.45 to 3.85ns**) from the data arrival time at the input port (**1.0 to 1.4ns**) to get **2.45ns**.

**Question 12.** Does the design have any violations?

From `report_constraints -all` you should see that there are no timing, area or DRC violations.

**Question 13.** Describe the resulting violating path:

The violating path is an internal register-to-register loop-back path.

**Question 14.** What command can be used to “isolate” the output ports from internal paths (allowing the use of inverters)?

```
set_isolate_ports -type inverter \
  [all_outputs]
```

**Question 15.** Does the design have any violations?

From `report_constraints -all` you should see that there are no timing, area or DRC violations.

**Question 16.** What did `set_isolate_ports` accomplish?

It isolated the output ports from the internal path as well as from each other. The internal path delay no longer depends on the external output loading. It did this even though there were no timing violations with the original constraints.

This page was intentionally left blank.

# 10

## Multiple Clocks and Timing Exceptions

### Learning Objectives

The goal of this lab is to give you a better understanding of how static timing analysis works and how timing exceptions are properly applied.

After completing this lab, you should be able to:

- Fully constrain and analyze a design using virtual clocks, false paths and multicycle paths.



**Lab Duration:**  
60 minutes

## Lab 10

### Background

The design you will be working with, called “exceptions”, contains parallel paths shown in the figure below (resets not shown). Between the inputs `adr_i` and `coeff`, and the output `dout` there is a purely combinatorial path as well as a sequential path.

The combinatorial and sequential paths have different constraints. When you perform timing analysis, you will discover that there are initially many violations – all due to incomplete or incorrect constraints.

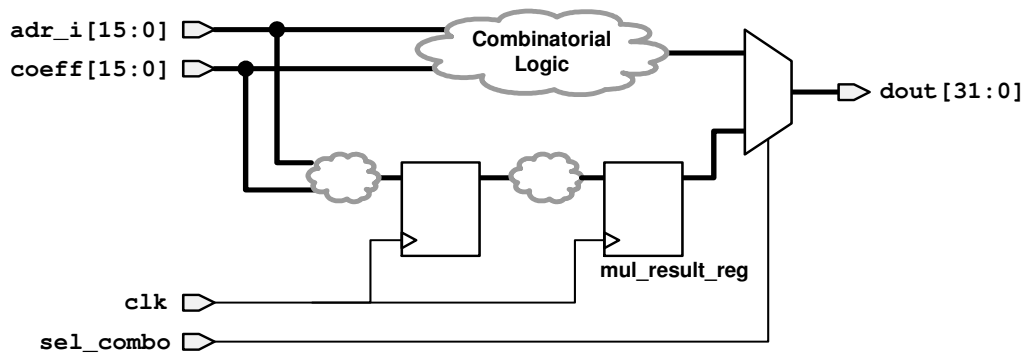


Figure 1. Design “exceptions”

*If you run into difficulties while performing the various tasks, check the Answers/Solutions section at the back of this lab.*

*You may also check the file `.solutions/run.tcl` for help.*

# Lab Instructions

## Task 1. Read a Mapped Design

---

1. Make sure your current working directory is **lab10** and invoke `dc_shell-t`.
2. Read the compiled design “exceptions” into memory.

```
read_ddc exceptions.ddc
link
```

3. Use the view utility to generate a constraint report, and a default timing report.

```
v rc
v rt
```

**Note:** “v” is an alias for the Tcl function “view”, which is defined in `../ref/tools/procs.tcl`. It is very useful during shell use.

**Question 1.** How large is the worst negative slack in comparison with the data required time?

.....

**Question 2.** Describe the start and end points of this violating timing path?

.....

**Question 3.** What is the maximum path delay constraint for this critical path ( = Clock Period – Input Delay – Output Delay)?

.....

This path is over constrained - it cannot meet a –1 ns maximum delay!

This design was given improper constraints for demonstration purposes. *The maximum propagation delay through the combinational path should be constrained to 10 ns.* The current input and output delay constraints are appropriate for the sequential paths (from the input ports to a flip-flop and from flip-flops to the output ports), but clearly not for the purely combinational paths.

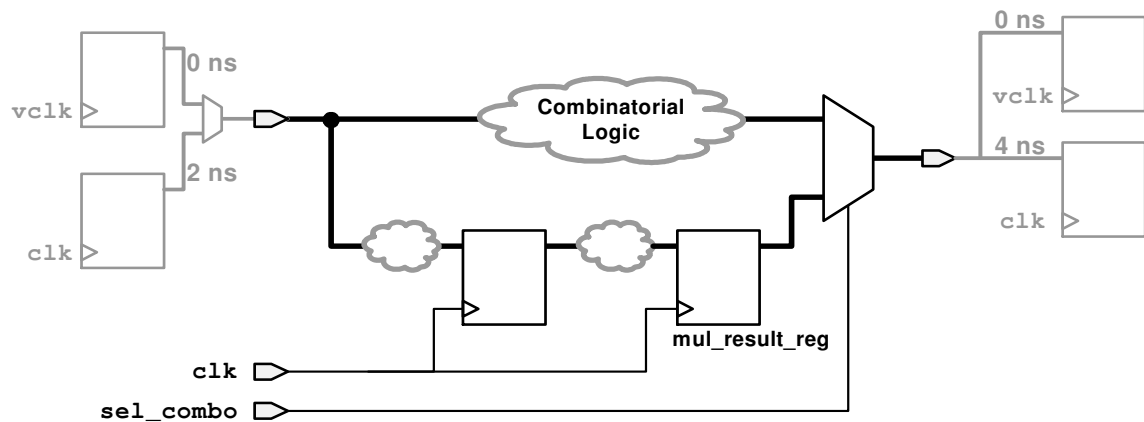
The constraints must be expanded such that the different paths are constrained separately.

## Task 2. Constrain Parallel Paths Separately

In this task you will use virtual clocks to constrain the combinatorial paths separately from the sequential timing paths.

1. Apply additional constraints so that the combinatorial logic is constrained independent of the sequential logic path constraints.

To accomplish this, constrain the combinatorial paths with respect to a different clock (**vclk**) than the sequential paths; this way multiple constraints can be overlaid without interfering with each other. The following schematic illustrates this concept (note that the two input ports, **adr\_i** and **coeff**, are collapsed into one port here to simplify the schematic):



The diagram shows that two clocks, **clk** and **vclk** are now clocking the input and output data of the design. The sequential paths, to and from the internal registers, are constrained using **clk**, and the purely combinatorial path should be constrained by the virtual clock **vclk**.

The combinatorial path must have a maximum delay of 10 ns. This is accomplished by applying a 10 ns period to **vclk**, while the input and output delays with respect to **vclk** are set to zero.

Remember that the input delay constraint should be applied to the ports **coeff\*** and **adr\_i\***

**Apply the appropriate commands based on what you learned in lecture. Use the *Job Aid* as needed.**

*If you're stuck, check the Answers section.*

2. Generate a timing report for a timing path constrained by the virtual clock.

The switch **-group** will generate a timing report for the path group **vclk** (i.e. the timing path with the worst slack for setup captured by the clock **vclk**).

```
v rt -group vclk
```



**Question 4.** Does the path violate or meet timing?

.....

**Question 5.** Describe the launch and capture clocks. Does this match your expectations?

.....

3. The clocks *clk* and *vclk* should not interact. Apply *false paths* to turn off timing analysis between these clocks.

*If you're stuck, check the Answers section.*

4. Repeat the last timing report

**Question 6.** Does the path violate or meet timing?

.....

**Question 7.** Describe the launch and capture clocks and does this match your expectations?

.....

5. This constraining situation is not quite over! Try the following report and answer the following questions.

```
v rt -group clk
```

**Question 8.** Does this path violate or meet timing?

.....

**Question 9.** Describe the start and end points of this timing path and does this match your expectations?

.....

6. As the final step, turn off timing analysis from clock *clk* to *clk* through the combinatorial logic using false paths.

Perform a timing report from all inputs to all outputs and confirm that the combinatorial paths are constrained by *vclk* and that these paths meet timing.

### Task 3. Constrain a Multicycle Path

---

The path to the `mul_result_reg` is a multicycle path, which is allowed to take up to 2 clock cycles (instead of 1, the default) for setup. The hold checks should be done at zero. These paths are not yet correctly constrained.

1. Execute a timing report for the clock `clk`.

```
v rt -group clk
```

**Question 10.** How large is this violation in comparison to the clock period?

.....

2. Apply a multicycle path of 2 cycles for setup to all paths that end at the `D` pin of `mul_result_reg`.
3. Verify that the multicycle path exception was applied correctly:

```
v rt -to mul_result_reg*/D
```

**Question 11.** What is the effective clock period for this timing path (the clock capture edge – the clock launch edge)?

.....

4. Generate a hold timing report to these same end points.

```
!! -delay min
```

**Question 12.** What are the launch and capture clock edges and does this match the specification described at the beginning of this task?

.....

5. Complete the multicycle path constraint by including a constraint for hold.

**Question 13.** What are the launch and capture clock edges and are they now correct for hold ?

.....

6. Generate a final “`report_constraint -all`” to make sure that all violations have been taken care of.
7. **Quit** Design Compiler.

 **Congratulations – This completes lab 10.**

## Answers / Solutions

- Question 1.** How large is the worst negative slack in comparison with the data required time?
- The data is required at 5ns. The WNS (~ -9 ns) is twice as large as the data required time!
- Question 2.** Describe the start and end points of this violating timing path?
- The start point is an input port and the end point is an output port.
- Question 3.** What is the maximum path delay constraint for this critical path ( = Clock Period – Input Delay – Output Delay)?
- 1ns (5ns – 2ns – 4ns)! This is not an achievable constraint.

### *Solution for task 2 step 1:*

```
create_clock -name vclk -period 10
set_in_ports [get_ports "coeff* adr_i*"]
set_input_delay 0 -clock vclk -add_delay $in_ports
set_output_delay 0 -clock vclk -add_delay [all_outputs]
```

- Question 4.** Does the path violate or meet timing?
- It violates.
- Question 5.** Describe the launch and capture clocks and does this match with your expectations?
- The launch clock is **clk** and the capture clock is **vclk**. This is not correct! The combinatorial paths should be constrained only by **vclk**. The two clocks should not interact.

### *Solution for task 2 step 3:*

```
set_false_path -from [get_clocks clk] -to [get_clocks vclk]
set_false_path -from [get_clocks vclk] -to [get_clocks clk]
```

**Question 6.** Does the path violate or meet timing?

It should meet timing.

**Question 7.** Describe the launch and capture clocks and does this match your expectations?

Both the launch and capture clocks are **vclk**. This is correct.

**Question 8.** Does this path violate or meet timing?

It violates.

**Question 9.** Describe the start and end points of this timing path and does this match your expectations?

The start point is an input port and the end point is an output port. This should not happen! The combinatorial paths should only be constrained by **vclk** and not by **clk**.

*Solution for task 2 step 5:*

```
set_false_path -from [get_clocks clk] \  
  -through $in_ports \  
  -through [all_outputs] -to [get_clocks clk]  
v rt -from $in_ports -to [all_outputs]
```

**Question 10.** How large is this violation in comparison to the clock period?

The violation is ~ 3.5 ns, very close to the clock period.

*Solution for task 3 step 2:*

```
set_multicycle_path 2 -setup -to mul_result_reg*/D
```

**Question 11.** What is the effective clock period for this timing path (the clock capture edge – the clock launch edge)?

The effective clock period is 10 ns, 2 times the actual clock period of **clk** (5 ns).

**Question 12.** What are the launch and capture clock edges and does this match the specification described at the beginning of this task?

The launch clock edge is 0ns and the capture clock edge is at 5 ns (one edge preceding the setup clock edge). This does not match the original specification which stated the hold checks should be done at zero.

*Solution for task 3 step 5:*

```
set_multicycle_path 1 -hold -to mul_result_reg*/D
```

**Question 13.** What are the launch and capture clock edges and are they now correct for hold?

The launch and capture clock edges are at zero and this is now correct.

# 11

## Synthesis Techniques

### Learning Objectives

After completing this lab, you should be able to:

- Select the recommended compile flow based on license and library availability
- Apply the recommended synthesis techniques to meet the required constraints
- Verify applied directives and variables before compile
- Analyze the gate level netlist:
  - a) To ensure that all constraints have been met
  - b) To observe the results of the various optimization techniques invoked
- Use the *Job Aid* to find all the necessary commands to accomplish the above steps



**Lab Duration:**  
**90 minutes**

# Lab Overview

The objective of this lab is to compile a design called STOTO using the information listed in the **Synthesis Specification** table. You will accomplish this by following the step-by-step **Lab Instructions**, which will guide you in performing the following tasks:

- Select the appropriate compile flow based on the stated **Available Resources**
- Use the DC shell interactively to read in and constrain the design
- Interactively apply and execute the appropriate compile flow techniques and commands, to achieve the stated **Design Specification**
- Verify the applied compile directive commands and variables before each compile or optimization
- Analyze the results after each compile or optimization to determine what step, if any, to perform next
- While performing all the above steps interactively you will also create a “run script” so that your steps can be easily corrected and re-applied as necessary

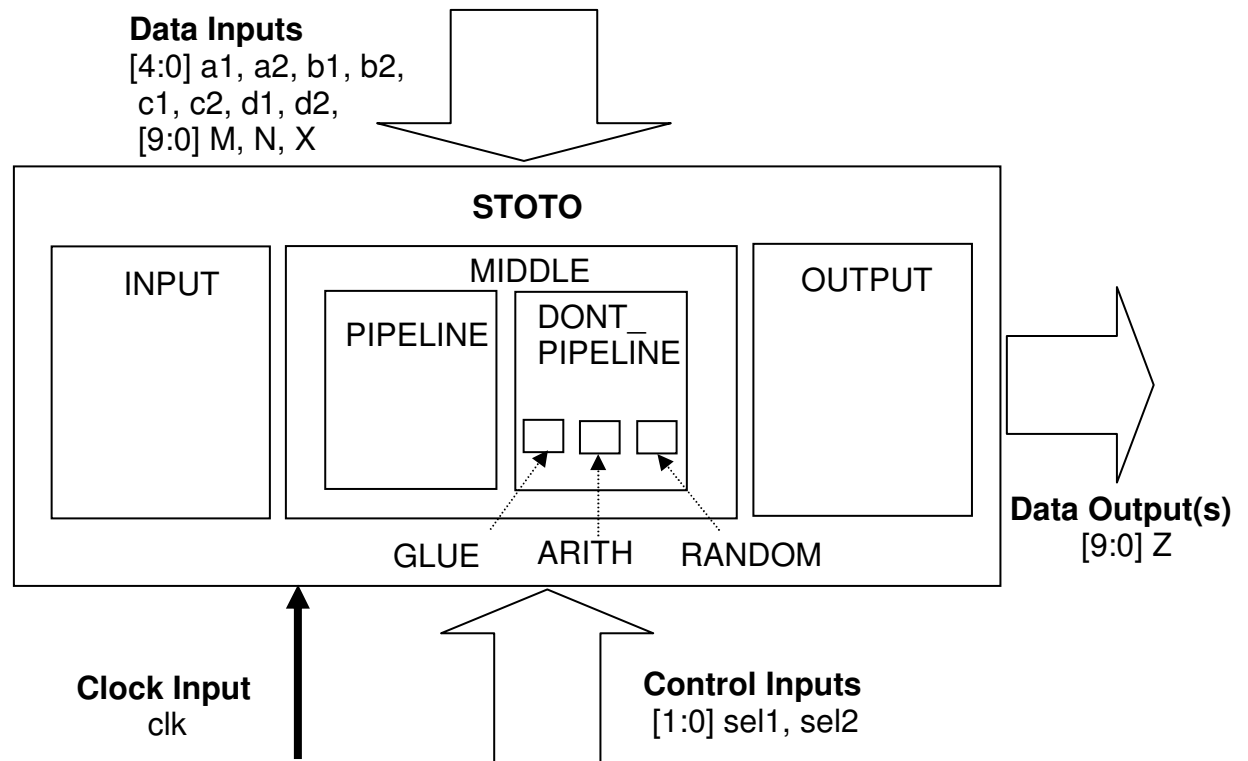


## Design Schematic

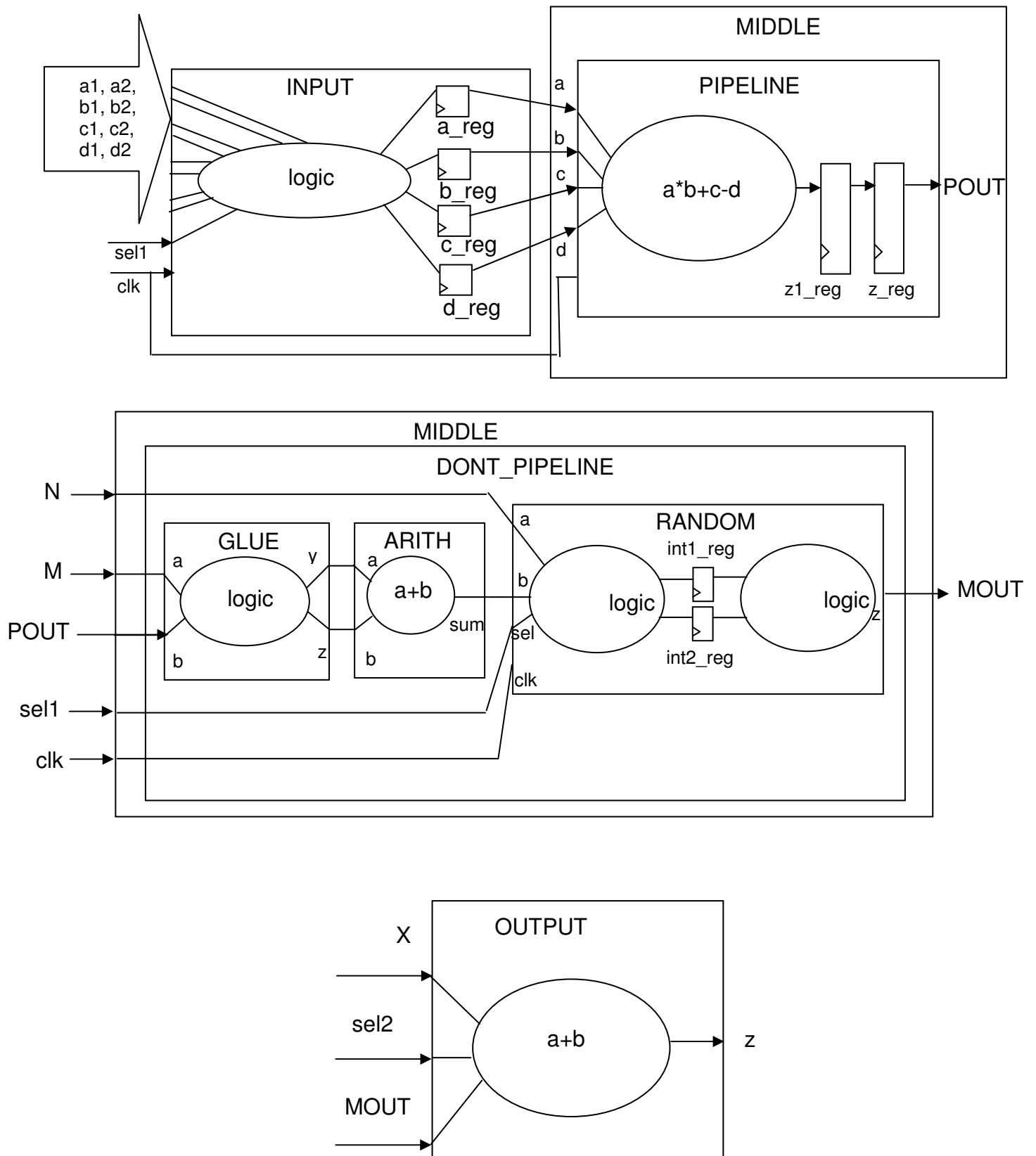
You are provided with the RTL code of the design (**rtl/stoto.v**) and the design constraints (**scripts/stoto.con**).

### RTL design hierarchy:

<u>Design Name</u>	<u>Cell Name</u>
STOTO	
INPUT	(I_IN)
MIDDLE	(I_MIDDLE)
PIPELINE	(I_PIPELINE)
DONT_PIPELINE	(I_DONT_PIPELINE)
GLUE	(I_GLUE)
ARITH	(I_ARITH)
RANDOM	(I_RANDOM)
OUTPUT	(I_OUT)



## Lab 11



# Synthesis Specification

<b>Available Resources</b>	<ol style="list-style-type: none"> <li>1. All Design Compiler and related licenses are available, but only a single license is available for each feature</li> <li>2. Neither the physical library (Milkyway) nor a design floorplan is available at this time</li> </ol>
<b>Design and Constraints Files</b>	<ol style="list-style-type: none"> <li>1. RTL code location: <i>rtl/stoto.v</i></li> <li>2. Design Name: <b>STOTO</b></li> <li>3. Constraints file location: <i>scripts/stoto.con</i></li> <li>4. These files may not be modified</li> </ol>
<b>Design Specification</b>	<ol style="list-style-type: none"> <li>1. The design is timing-critical</li> <li>2. The I/O constraints are estimates and have been conservatively constrained</li> <li>3. The final compiled design should meet setup timing on all internal register-to-register paths</li> <li>4. Area and design rule constraints must also be met, if possible</li> <li>5. Scan insertion will be performed by the Test group after the design has met these specifications</li> <li>6. The <b>INPUT</b> block hierarchy should be preserved to facilitate post-synthesis verification</li> <li>7. The arithmetic logic (<math>a * b + c - d</math>) in <b>PIPELINE</b> is expected to take almost 2 clock cycles</li> <li>8. The output (<b>POUT</b>) of <b>PIPELINE</b> must remain registered</li> <li>9. The positions of non-pipelined registers (i.e., registers in the <b>INPUT</b> and <b>DONT_PIPELINED</b> blocks) are fixed and cannot be modified</li> </ol>

# Lab Instructions

1. Look at the **Available Resources** section of the **Synthesis Specification** table to answer the following question:

**Question 1.** Which of the following is the appropriate flow for this project: Expert, Ultra-WLM, Ultra-Topographical or ACS?

.....

**Note:** You are encouraged to check your answers against the *Answers/Solutions* section at the end of the lab.

2. Invoke *DC shell* in the appropriate “mode” (WLM versus Topographical) from the *lab11* directory.
3. Create a new file called *run.tcl* in the *scripts* directory. For each of the following steps, whenever you apply a command interactively in the DC-shell environment, copy and paste the command into the *run.tcl* file.
4. In your *Job Aid*, locate the **Run Script** section and the appropriate **Ultra Compile Flow** section from your answer above. Refer to these sections while performing each of the following steps.
5. In the DC-shell environment, interactively read, constrain and check the design listed in the **Design and Constraints Files** section of the **Synthesis Specification** table.

Remember to also “copy and paste” the commands into your run script file *run.tcl*!

6. **Interactively apply** the recommended commands, directives, variables and attributes, up until, but NOT including the first *compile*, which address **Design Specification** #2, 3, 6 and 7.

Remember to copy and paste into your run script.

**Note:** Refer to your *Job Aid* sections called **Run Script** and **Ultra Compile Flow**.

**Note:** If you are **really stuck** you can refer to the *run scripts* in the **.solutions** directory:  
**run.tcl** is the “plain” script which contains only the required run script commands – no “checking” and no explanations;  
**run\_w\_check\_expl.tcl** contains additional “checking” commands to verify each applied directive, variable or attribute, as well as comments explaining each step.

7. Enter the following commands to verify that all of the appropriate variables, directives and attributes have been correctly applied prior to *compile*:

```
report_path_group
printvar compile_auto_ungroup*
get_attribute [get_designs "PIPELINE INPUT"] ungroup
report_timing_requirements
report_timing_requirements -ignored
```

**Question 2.** Are you seeing the expected results for each check?

.....

.....

.....

.....

.....

8. Save your design as *mapped/pre\_compile.ddc*.

**Question 3.** What `compile_ultra` options will you apply? Based on which four **Design Specification** items?

.....

.....

9. **Compile** the design with the appropriate options.

While you are waiting for the compile to finish, use the design schematic on the previous pages to answer the following two questions:

## Lab 11

**Question 4.** Is the design well partitioned for synthesis?  
Explain.

.....

**Question 5.** Which sub-designs would need to be ungrouped to obtain ideal partitioning for synthesis?

.....

**10.** After the *compile* is completed, check the design's hierarchy:

```
get_designs *  
report_hierarchy -noleaf  
report_auto_ungroup
```

**Question 6.** Did auto-ungrouping eliminate the expected sub-designs?

.....

**11.** Generate a constraint report (rc) to the screen as well as to a file:

```
redirect -tee -file rc_compile_ultra.rpt {rc}
```

**Question 7.** Are there any constraint violations? If so, describe them.

.....

.....

.....

**Question 8.** Should you be concerned about these violations? Explain.

.....

.....

12. Look at the “Endpoint” of one of the “min-delay” violations. Notice that the endpoint “I\_MIDDLE/I\_PIPELINE/z1\_reg...” contains the instance name “I\_MIDDLE”, even though the sub-design MIDDLE was auto-ungrouped!

When ungrouping, Design Compiler keeps the original hierarchical path name to a child cell and converts it into a non-hierarchical cell name – the cell name of the PIPELINE sub-block is now I\_MIDDLE/I\_PIPELINE – here the slash no longer denotes a hierarchy separator but is just a character that is part of the cell’s new, longer name. This makes it easy to “trace” ungrouped cells to know where they originally came from.

13. Save your design as *mapped/compile\_ultra.ddc*.

**Question 9.** Since the violations are of no concern, are you done with synthesis? Explain.

.....

.....

.....

14. Remove the “timing exception” from the design.

**Question 10.** Are there any “worriesome” constraint violations now? Explain.

.....

.....

15. **Optimize the pipeline** while taking into consideration **Design Specification #8 and 9**.

Registers that have been moved or repositioned by `optimize_registers` end with the following cell name: `clockname_r_REG#_S#`. Use this fact to answer the following questions:

**Question 11.** How can you verify that registers were moved?

.....

.....

## Lab 11

**Question 12.** How can you verify that only **PIPELINE** registers were moved?

.....  
.....

**Question 13.** How can you verify that the **z\_reg\*** registers were not moved?

.....  
.....

**16.** Generate another constraint report.

**Question 14.** Have the violations improved? Explain.

.....  
.....

**17.** Save your design as *mapped/optimize\_reg.ddc*.

**18.** Refer to the bottom portion of the “**Ultra Compile Flow**” section of the *Job Aid* to answer the following two questions:

**Question 15.** What are the next suggested steps if there are still violations after `optimize_registers`?

.....  
.....  
.....

**Question 16.** Based on the **Design Specification**, does it make sense to perform these steps?

.....

CONGRATULATIONS! You have successfully completed the “Synthesis Techniques” lab! With the help of the provided *Job Aid* you should now be able to apply the recommended synthesis flow and techniques to any real-world design!

Continue with the OPTIONAL LAB next if you have some extra time.



## OPTIONAL LAB: Adaptive Retiming

In this optional lab you will compare the results of a *compile* with, and without, *adaptive retiming* (`-retime`).

1. Edit your *run.tcl* script as follows:
  - Change the saved design name prior to `compile_ultra` to *pre\_retime.ddc*
  - Add the `-retime` option to the `compile_ultra` command
  - Change the saved design name after `compile_ultra` to *compile\_retime.ddc*
  - Redirect a constraint report to *rc\_compile\_retime.rpt* immediately after `compile_ultra`
  - Add the TCL `return` command just before resetting the multicyle path constraint to terminate script execution at this point
  - Save and close the run script file
2. Remove the current design from DC memory (`fr`).
3. Execute the updated run script (`source`)
4. Compare the constraint report results from this run (*rc\_compile\_retime.rpt*) to that of the run without adaptive retiming (*rc\_compile\_ultra.rpt*) which you generated previously.

**Question 17.** What difference do you notice in the worst slack of the *INPUTS* path group?

.....

**Question 18.** How can you tell that the endpoint registers in the *INPUTS* path group were affected by `-retime`?

.....

.....

## Lab 11

**Question 19.** Can you explain why there is now an area violation?

.....

.....

You have now seen how adaptive retiming can be effectively used to improve timing in a timing-critical design. You have also noticed that the design size can increase due to register splitting.

CONGRATULATIONS!! This concludes Lab 11.

## Answers / Solutions

**Question 1.** Which of the following is the appropriate flow for this project: Expert, Ultra-WLM, Ultra-Topographical or ACS?

**Ultra-WLM.** Since, according to the **Available Resources**, “All Design Compiler and related licenses are available”, we should use the Ultra flow. Since “only a single license is available for each feature”, we can not use ACS. Lastly, since “Neither the physical library (Milkyway) nor a design floorplan is available at this time” we can not use Ultra’s “topographical mode”.

**Question 2.** Are you seeing the expected results for each check?

`report_path_group` should confirm that there are 4 path groups, in addition to the default group: an input, output, combinational and *clk* path group. The *clk* group should have a *weight* of **5** and a *critical range* of **0.21**.

`printvar compile_auto_ungroup*` should confirm the following variable values:

```
compile_auto_ungroup_area_num_cells =
"30" (not used – default value shown)
compile_auto_ungroup_count_leaf_cells =
"true"
compile_auto_ungroup_delay_num_cells =
"99999999" (or any large number)
compile_auto_ungroup_override_wlm =
"true"
```

`get_attribute [get_designs "PIPELINE \ INPUT"] ungroup` should return “**false false**” as the value of the ungroup attribute on each of the two designs.

`report_timing_requirements` should confirm that there is a **setup timing exception** applied, for **2 cycles**, to or from the appropriate end/startpoint, respectively.

`report_timing_requirements -ignored` should not return anything, confirming that the applied exception was accepted (no start/endpoint errors).

**Question 3.** Which `compile_ultra` options will you apply? Based on which three **Design Specification** items?

```
compile_ultra -timing -scan
```

(Note: No `-retime` option!!)

From #1 “The design is timing-critical”, and from #4 “Area and design rule constraints must also be met, if possible”, you should use the **-timing** option.

From #5 “Scan insertion will be performed by the Test group after the design has met these specifications”, you should use the **-scan** option.

From #9 “The positions of non-pipelined registers are fixed and cannot be modified”, you should NOT use **-retime**.

**Question 4.** Is the design well partitioned for synthesis? Explain.

No! Logic optimization will be restricted at the interfaces between the following sub-designs, due to hierarchical partitioning:

```
GLUE → ARITH  
ARITH → RANDOM  
RANDOM → OUTPUT
```

**Question 5.** Which sub-designs would need to be ungrouped to obtain ideal partitioning for synthesis?

MIDDLE, DONT\_PIPELINE, GLUE, ARITH, RANDOM and OUTPUT.

**Question 6.** Did auto-ungrouping eliminate the expected sub-designs?

If the auto-ungrouping variables were correctly applied, and `compile_ultra` was used, the designs listed in the previous answer should have all been auto-ungrouped. The only designs left in DC memory should be: STOTO, INPUT, PIPELINE, and INCORRECT (which is an “empty, standalone” module included in the RTL just to trap you in case you forgot to set the `current_design` to STOTO after reading in the RTL).

**Question 7.** Are there any constraint violations? If so, describe them.

There are **max-delay** violations in the *input* path group. You may also see **min-delay** violations associated with the **PIPELINE** registers in the *clk* path group, unless you applied a `set_multicycle_path -hold` constraint along with the `-setup` constraint.

**Question 8.** Should you be concerned about these violations? Explain.

Since, according to the Design Specification, the I/O constraints are conservative, and the goal is to meet reg-to-reg timing, the max-delay violations are not critical.

The min-delay violations, if there, are a product of the multi-cycle timing exception which was temporarily applied to the pipeline arithmetic logic for the first compile. The exception increased the *setup* timing cycle by one additional clock cycle, and, since no *hold* exception was applied, the hold time is also increased by one clock cycle, by default, causing the violations. These violations are therefore of no concern.

**Question 9.** Since the violations are of no concern, are you done with synthesis? Explain.

No! The multi-cycle timing exception in the PIPELINE design is probably masking some critical reg-to-reg timing violations through the arithmetic logic in PIPELINE.

**Question 10.** Are there any “worriesome” constraint violations now? Explain.

Yes!

You should now see large **max-delay** violations in the *clk* group, ending at the *z1\_reg* registers, which are in the PIPELINE sub-design.

You should also see the same **max-delay** violations in the *input* group as before.

You should notice that **min-delay** violations are gone.

**Question 11.** How can you verify that registers were moved?

`get_cell -hier *REG*_S*` returns cell names, which proves that `optimize_registers` did in fact move some registers.

**Question 12.** How can you verify that only **PIPELINE** registers were moved?

Since every single cell name starts with **I\_MIDDLE/I\_PIPELINE** you know that only PIPELINE registers were moved.

**Question 13.** How can you verify that the **z\_reg\*** registers were not moved?

Since all the register cells end with **SI** you know that only **z1\_reg**, the first stage registers, were moved.

This can be further verified with additional checks:  
`get_cell -hier *z_reg*` shows that the original register names still exist, which means that `optimize_registers` did not move them. You can also generate a timing report which shows that the PIPELINE output is registered:

```
report_timing -from \
                I_MIDDLE/I_PIPELINE/z_reg[*]/*
```

**Question 14.** Have the violations improved? Explain.

Yes! All the reg-to-reg **clk** group violations (from PIPELINE) should be gone.

**Question 15.** What are the next suggested steps if there are still violations after `optimize_registers`?

1) Apply more focus on violating critical paths, as necessary: `group_path -weight -critical`

2) Enable Ultra optimization:  
`set_ultra_optimization true`

3) Perform an incremental high-effort compile  
`compile -bound -scan -map high -incr`

**Question 16.** Based on the **Design Specification**, does it make sense to perform these steps?

No. We have met the stated specification that all reg-to-reg setup timing must be met. Since the I/O constraints are estimates and have been conservatively constrained, it does not make sense to spend any more time trying to get these I/O violations to pass.

**Question 17.** What difference do you notice in the worst slack of the *INPUTS* path group?

The worst or largest negative slack should be markedly lower with adaptive retiming. You may also notice an increased number of smaller violations.

**Question 18.** How can you tell that the endpoint registers in the *INPUTS* path group were affected by `-retime`?

The registers are named *R\_##* as opposed to *\*\_reg[#]*. Adaptive retiming renames moved registers with the former naming convention.

**Question 19.** Can you explain why there is now an area violation?

This is because the number of registers has increased substantially, due to register “splitting”. You can see the register area increase by generating an area report (`report_area`) for the retimed and the non-retimed designs, and comparing the “Noncombinational area” numbers.

This page was intentionally left blank.