SYNOPSYS®

CUSTOMER EDUCATION SERVICES

DFT Compiler 1 Workshop

Lab Guide

30-I-011-SLG-012 2007.12

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

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

www.synopsys.com

Copyright Notice and Proprietary Information

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, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, Columbia,Columbia-CE, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, Direct Silicon Access, Discovery, Encore, Galaxy, HANEX, HDL Compiler, Hercules, Hierarchical Optimization Technology, HSIMplus, HSPICE-Link, iN-Tandem, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport,Library Compiler, Magellan, Mars, Mars-Rail, Milkyway, ModelSource, Module Compiler, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Raphael-NES,Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, Taurus, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, 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: 30-I-011-SLG-012
DFT Compiler 1 Lab Guide

4 Creating Test Protocols

Learning Objectives

After completing this lab, you should be able to:

- Provide enough information about the test features of your design that DFTC can create a test protocol.
- Verify that you have told DFTC what it needs to know
- Verify that your protocol and design are compatible
- Verify that the design has been compiled to the correct library and that all flip-flops have been replaced with scan flip-flops
- Modify a test protocol to include an initialization sequence for applying an internal test mode signal

Lab Duration: 90 minutes

Part A Overview

The files for **Part A** of this lab are located in the directory **`lab4a_protocol`**.

Directory Structure

`Lab4a_protocol`	Current working directory
`analyzed`	Intermediate acs_read_hdl files
`logs`	Session log files
`unmapped`	Unmapped protocol
`mapped`	Gate-level netlist
`reports`	DFTC reports
`tmax`	Files for downstream tools
`scripts`	Constraint and run scripts
`../ref/rtl/RISC_CORE/vhdl`	Design files
`../ref/db`	Library files

If you need help...

```
dc_shell> help prev*
dc_shell> printvar *scan*
dc_shell> preview_dft -help
dc_shell> man compile
```

When working interactively, you do not need to type the entire command or option.

```
dc_shell> read_test_p -v
dc_shell> all_inputs -cl
```

Answers & Solutions

This lab guide contains answers and solutions to all questions. If you need some help with answering a question, consult the back portion of this lab for help.

Instructions

During this lab, you will use the Design Vision GUI for debugging `dft_drc` problems related to test protocols, write a script to create and verify a test protocol and explore a `dc_shell` log file.

Task 1. Set Up Your Environment For Reading

1. Change directories to the project directory **`lab4_protocol`**.

   Look at the "dot" files in that directory.

```
UNIX% cd lab4a_protocol
UNIX% pwd
UNIX% ls –a
```

2. Specify the libraries and append to the search path by editing the **`.synopsys_dc.setup`** file with a text editor.

   Add the following lines to the `.synopsys_dc.setup` file.

```
# for reading and linking
set target_library  "sc_max.db"
set link_library  "* sc_max.db dw_foundation.sldb"
lappend search_path ./scripts
```

The pound (#) at the beginning introduces a TCL comment.

Question 1. You use the `set` command to assign values to 'things' such as `target_library` and `link_library`; what are these 'things' called in TCL?

.....

Question 2. When you write out a design to a .ddc file, what is the difference in how values defined with the **set** command are treated, compared to how values defined with commands starting with **set_,** such as **`set_scan_configuration`** or **`set_dft_signal`**, are treated?

.....

.....

3. Create shortcuts by editing the **.synopsys_dc.setup** file.

   a. Create the following commonly-used alias for the frequently-used **`history`** command.

   b. Then turn on the capability to edit commands in `dc_shell`.

   c. Then save the **.synopsys_dc.setup** file and quit the editor.

```
# shortcuts
history keep 100
alias h "history"
set sh_enable_line_editing true ;# wow - read man page
```

Question 3. To use the history command in the Design Compiler environment (design_vision/dc_shell/etc.) what would you type at the command line after specifying: **`alias h history`**?

.....

4. Further examine the .synopsys_dc.setup file. Note the two variables **sh_output_log_file** and **sh_command _log_file**. By setting these two variables, you control where a command and an output log file are created. The two files are created using the current time stamp and pid. This sequence of commands can be inserted in most Synopsys tools.

5. Invoke Design Vision from the UNIX prompt in the lab4_protocol directory:

```
UNIX% design_vision
```

Note: You must open Design Vision from the same directory where the `.synopsys_dc.setup` exists, because, as shown in the previous steps, variables such as `search_path` are defined relative to the directory containing the setup file.

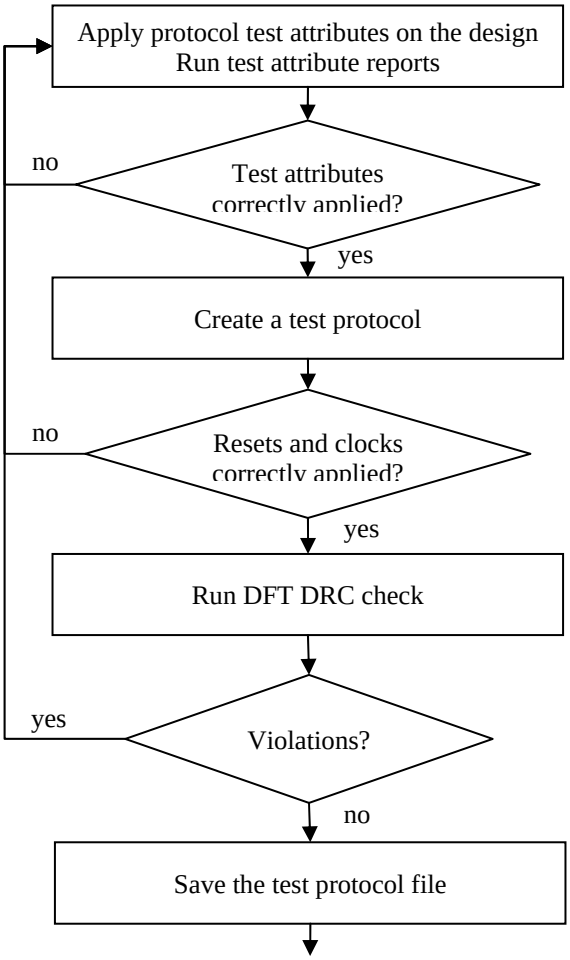
Note: If you see any errors while invoking DV, refer to the golden solution at **./.solutions/dot.synopsys_dc.setup**

6. List the contents of the **logs** directory. You should see the two log files discussed above.

Task 2. Create a test protocol with a script

Utilize the flowchart below and the protocol test attributes specification to its right as a guide to write and verify a DFTC script to create a test protocol.

Test Protocol Definition Flow



Protocol Test Attributes	
Test Clock Port	Clk
Test Clock Active Time	45 ns
Test Clock Inactive Time	55 ns
Test Mode Port	TEST_MODE
Reset port	Reset (asynchronous active low)

1. Open the script called `scripts/unmapped.tcl`.

   The tasks are defined in the flow chart above; the values for the attributes in the chart beside it. Your task is to write the DFTC commands to perform the tasks. For this lab, set the values for the test attributes in the script.

2. Execute your script by running the **unmapped.tcl** script in batch mode.

   Create a log file for the run:

```
unix% dc_shell -f scripts/unmapped.tcl | \
         tee logs/run_unmapped.log
```

3. Determine if there were any problems in the unmapped flow by looking at **logs/run_unmapped.log**.

Question 4. What command is used to see which DFT signals have been applied to the design?

.....

Question 5. a. What value is the TEST_MODE port set to?
b. What is the name of the test clock port?
c. What is the period of the test clock?
d. When does the test clock go high?
e. When does the test clock go low?
f. What is the name of the reset port?
g. What type of reset has been assumed?

.....

.....

.....

.....

.....

.....

Question 6. Look at the results of running the `dft_drc` command, are there any errors or warnings?

.....

Task 3. Verify that the Design was Compiled Properly

Question 7. What command tells whether the entire design was compiled to the target library?

.....

1. Verify that the entire design was compiled to the target library.

Question 8. What target library was it compiled to?

.....

Question 9. Now that you know the design has been properly compiled, how do you determine if the flip-flops have been scan replaced?

.....

2. Determine if the design has been properly scan replaced.

Task 4. Part A Review Questions

Question 10. What is the purpose of the **.synopsys_dc.setup** file?

.....

Question 11. How do you verify the library variables are set up correctly?

.....

Question 12. How do you read VHDL or Verilog code into Design Vision?

.....

Question 13. What is the function of the **target_library** variable?

.....

3. Exit from DFTC.

Part B Overview

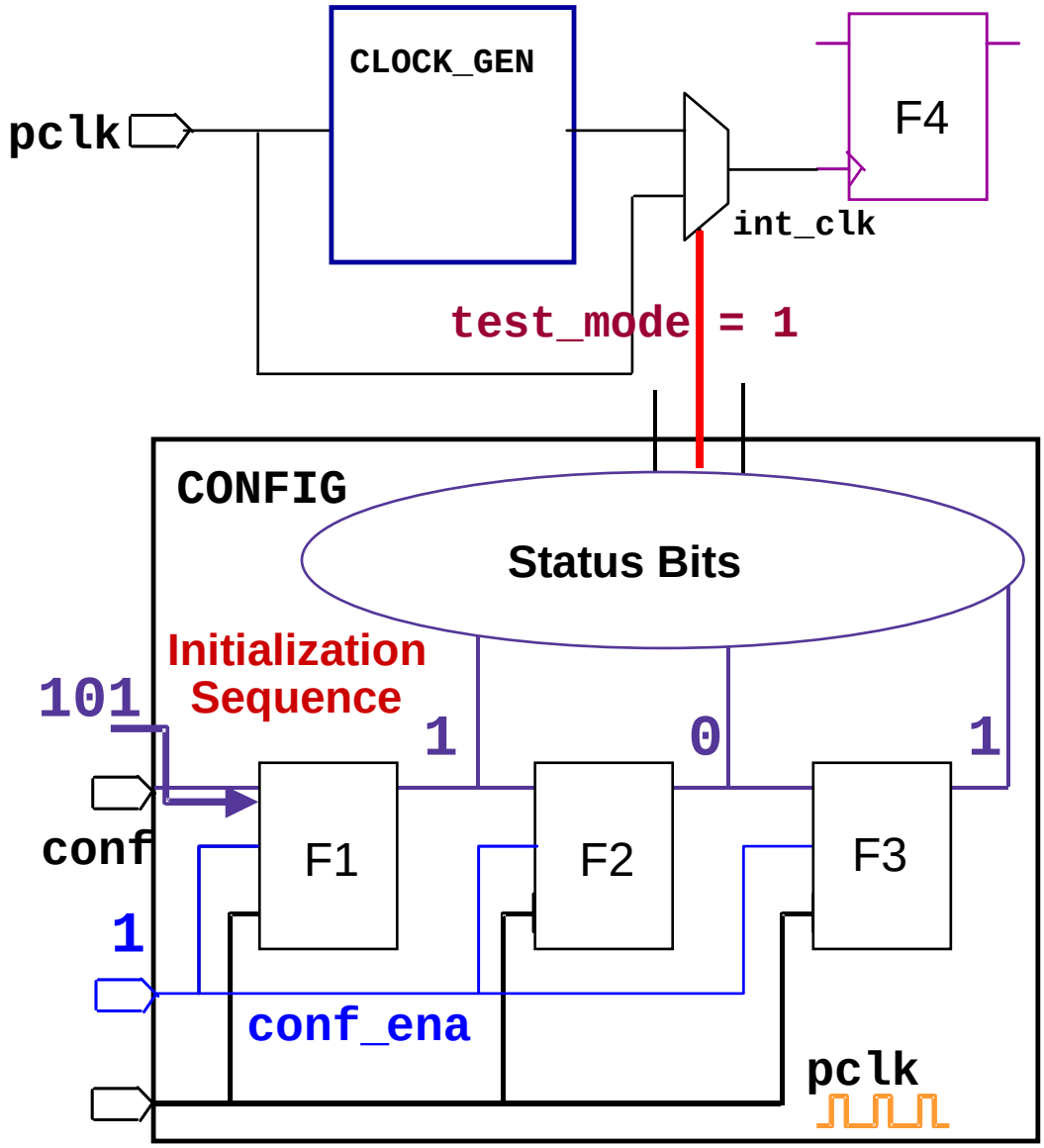
All files for Part B of this lab except for designs and libraries are located in the directory **`lab4b_init`**.

Directory Structure

`Lab4b_init`	Current working directory
`analyzed`	Intermediate acs_read_hdl files
`logs`	Session log files
`unmapped`	Unmapped protocol
`mapped`	Gate-level netlist
`mapped_scan`	Gate-level scan design netlist
`reports`	DFTC reports
`tmax`	Files for downstream tools
`scripts`	Constraint and run scripts
`../rtl/ORCA_init/vhdl`	Design files
`../libs`	Library files

Design Introduction

The design used in this task requires a custom test initialization protocol. You obtained the following schematic from the ORCA design documentation that describes the process for asserting the internal test_mode signal.



Instructions

Since DFTC cannot infer a test protocol of this type you must develop one for this design. Answer the following questions in preparation for doing the subsequent lab tasks.

Task 1. Internally Generated `test_mode` Signal Using the CONFIG Block

Question 14. What value must be placed on the conf_ena signal to initialize the configuration register flip-flops F1, F2 and F3?

.....

Question 15. How many cycles must you pulse pclk before test_mode is asserted?

.....

Question 16. What values should conf_ena and pclk have the cycle after test_mode is asserted?

.....

Question 17. Write in STIL format the Vector statements that would go into the "`test_setup`"?

```
. . .
MacroDefs {
"test_setup"
{
. . .




. . .
}
```

You will use this information later when you develop the initialization protocol for ORCA.

Instructions

During this lab you will, develop a custom test initialization protocol for a design that has an internal test_mode signal.

Task 2. Read a Mapped Design and Create a Test Protocol

1. Change directories to the project directory **`lab4b_init`**.

```
UNIX% cd lab4b_init
```

2. Invoke DFTC.

```
dc_shell
```

3. Create a shortcut to make running the scripts easier.

```
alias s "source -echo -verbose"
```

4. A mapped design has been prepared for you in the interest of saving time during lab (Reading the RTL and compiling).

   Examine the contents of the script that reads the saved ddc file for the ORCA design and then source the script to read in ORCA.

```
exec cat scripts/1read_design.tcl

s scripts/1read_design.tcl
```

Question 18. Where is the design read from? Is the design RTL or mapped gates?

.....

Before you create the starting test protocol you must specify all your clocks, resets, constants, etc. The clocks and the resets are already defined in the `2create_test_protocol.tcl` script.

Question 19. After the test initialization asserts the internal `test_mode` signal what must you do to keep the ORCA design in that mode throughout the rest of the scan testing?

.....

5. Add the appropriate command to the **2create_test_protocol.tcl** script.
6. Create the test protocol for ORCA by running the script.

```
s scripts/2create_test_protocol.tcl
```

Review the results of the `dft_drc` run on the ORCA RTL design.

Question 20. Are there few or many testability problems reported?

.....

Task 3. Create an Initialization Sequence

Developing the initialization test protocol from scratch is not practical for most engineers. A better method is to modify the protocol files that DFTC or TetraMAX create for you.

1. Save out the current test protocol for the design.

```
write_test_protocol –o orca_mapped.spf
vi orca_mapped.spf
```

2. Add the required Vector statements ("`V {}`") to the orca_mapped.spf.

   Question 21. Since you only changed the "`test_setup`" section of the STIL test protocol, which command should you use to read it back into DFTC?

   .....

3. Verify that your initialization vectors are correct by reading in your custom protocol and rerunning `dft_drc`.

   Question 22. What happens when you first try to read orca_`mapped.spf` back into DFTC?

   .....

4. Solve the problem by removing the previous protocol and reading in your custom initialization.

```
remove_test_protocol
read_test_protocol -section test_setup  \
                                  orca_mapped.spf
```

Note: The "`test_setup`" or initialization section is only part of a complete test protocol.

5. Re-create the rest of the test protocol before attempting to run `dft_drc`.

```
create_test_protocol –capture_procedure multi_clock
dft_drc
```

Question 23. How do the results of `dft_drc` compare to your first run?

.....

Congratulations! You have completed the lab.

Answers / Solutions

Task 1. Set Up Your Environment For Reading

Question 1. You use the `set` command to assign values to 'things' such as `target_library` and `link_library`. What are these 'things' called in TCL?

Variables

Question 2. When you write out a design to a .ddc file, what is the difference in how values defined with the **set** command are treated, compared to how values defined with commands starting with **set_,** such as **`set_scan_configuration`** or **`set_dft_signal`**, are treated?

When a variable is applied to a design, it does not "stick" to the design when the design is saved in a .ddc format. This is different from a "set_" command that can be applied to a .ddc file and is saved with the .ddc file. A variable must be reapplied each time the design is brought back into Design Compiler.

Question 3. To use the history command in the Design Compiler environment (design_vision/dc_shell-t/etc.) what would you type at the command line after specifying: **`alias h history`**?

h

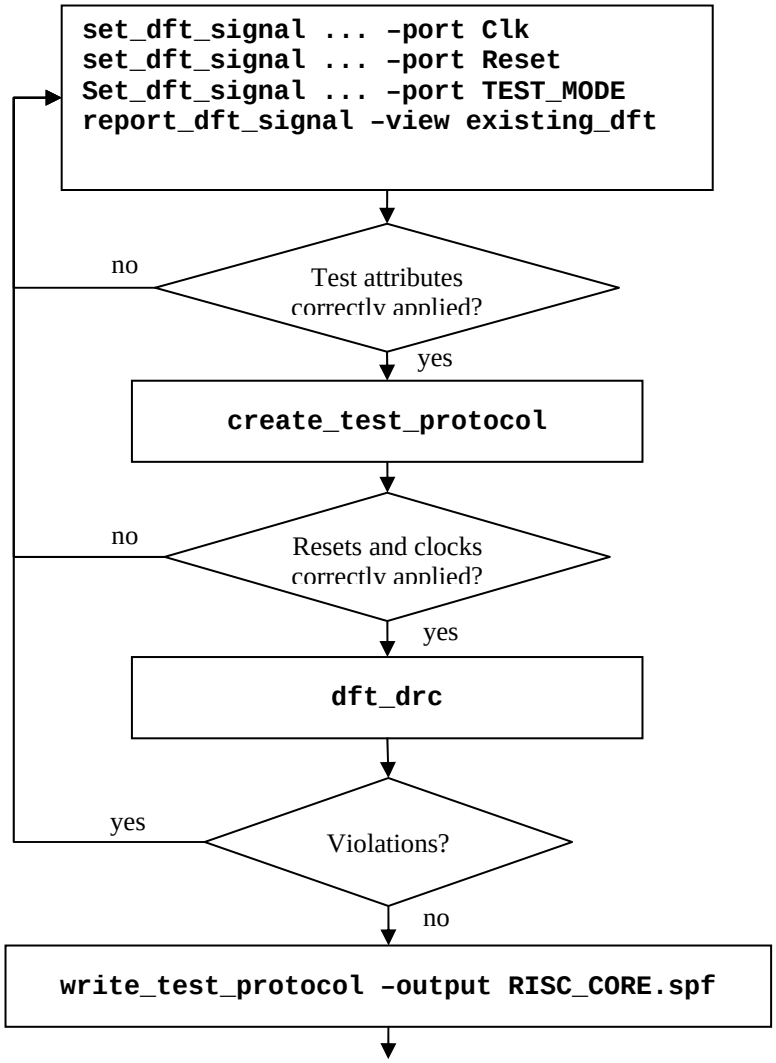
Task 2. Create a Test Protocol with a script

1. Modify the script called **unmapped.tcl**.

   If you run into problems, consult **.solutions/unmapped.tcl**

   This flowchart shows you the command flow of the create_test_protocol.tcl file as well as the decision points.

Test Protocol Definition Flow



Question 4. What command is used to see which DFT signals have been applied to the design?

`report_dft_signal`

Question 5. a. What value is the TEST_MODE port set to?
b. What is the name of the test clock port?
c. What is the period of the test clock?
d. When does the test clock go high?
e. When does the test clock go low?
f. What is the name of the reset port?
g. What type of reset has been assumed?

a. TEST_MODE = 1

b. Test clock port name is: Clk

c. Period of the test clock is:100 ns

d. Test clock high: 45 ns

e. Test clock low: 55 ns

f. Reset port name is: Reset

g. Reset type is: Asynchronous, active low

Question 6. Looking at the results of running the `dft_drc` command, are there any errors or warnings?

No

Task 3. Verify that the Design was Compiled Properly

Question 7. What command tells whether the entire design was compiled to the target library?

Use the `report_hierarchy` command. All cells should be cb13fs120_tsmc_max cells (target library).

Question 8. What target library was it compiled to?

Run the `report_hierarchy` command from the command line. All cells were cb13fs120_tsmc_max cells, so the design was properly compiled to the target library

Question 9. Now that you know the design has been properly compiled, how do you determine if the flip-flops have been scan replaced?

Run `report_scan_state` after `compile -scan` has occurred. Design was properly scan replaced:

dc_shell> **`report_scan_state`**

```
****************************************

Report : test
        -state
Design : RISC_CORE
****************************************
```

Scan state : scan cells replaced with loops.

Task 4. Part A Review Questions

Question 10. What is the purpose of a **.synopsys_dc.setup** file?

The **`.synopsys_dc.setup`** file is a script file, which is executed upon invocation of either Design Vision or **`dc_shell`.** It is a useful file for commands that are to be executed each time DC is invoked. For example, it is useful to initialize library variables in this file.

Question 11. How do you verify the library variables are set up correctly?

`acs_read_hdl` does an automatic link, which produces warnings or errors if the variables are not set correctly.

Question 12. How do you read VHDL or Verilog code into Design Compiler?

To read individual files in, use the **`read_vhdl`** or **`read_verilog`** command. To read files that have subblocks, use the **`acs_read_hdl`** command

Question 13. What is the function of the **`target_library`** variable?

The **`target_library`** variable defines the technology library used during compile mapping phase.

Task 1. Internally Generated `test_mode` Signal Using the CONFIG Block

Question 14. What value must be placed on the conf_ena signal to initialize the configuration register flip-flops F1, F2 and F3?

conf_ena must be logic 1 to initialize the flip-flops.

Question 15. How many cycles must you pulse pclk before test_mode is asserted?

pclk must pulse three times.

Question 16. What values should conf_ena and pclk have the cycle after test_mode is asserted?

Both should be logic 0.

Question 17. Write in STIL format the Vector statements that would go into the “test_setup”?

. . .

```
MacroDefs {
“test_setup”
{
. . .
V { “conf_ena”=1; “conf”=1; “pclk”=P; }
V { “conf_ena”=1; “conf”=0; “pclk”=P; }
V { “conf_ena”=1; “conf”=1; “pclk”=P; }
V { “conf_ena”=0; “pclk”=0; }
. . .
}
```

Task 2. Read a Mapped Design and Create a Test Protocol

Question 18. Where is the design read from? Is the design RTL or mapped gates?

mapped/ORCA.ddc

The design is mapped gates.

Question 19. After the test initialization asserts the internal test_mode signal what must you do to keep the ORCA design in that mode throughout the rest of the scan testing?

You must hold the conf_ena at logic 0 by using the following command:

```
set_dft_signal -view existing_dft -type Constant \
               -active_state 0 -port conf_ena
```

Question 20. Are there few or many testability problems reported?

Many.

```
4010 PRE-DFT VIOLATIONS
  2828 Uncontrollable clock input of flip-flop violations (D1)
  1133 DFF set/reset line not controlled violations (D3)
    32 Clock feeding both clock and data input violations (D11)
     6 Clock feeding multiple clock/set/reset inputs violations (D12)
     1 Clock path affected by clock captured by clock in trailing edge
       clock_port violation (D16)
    10 Clock_port not capable of capturing data violations (D17)
```

Task 4. Create an Initialization Sequence

8. Add the required vector statements to the unmapped_flow.spf.

```
V { "conf"=1; "conf_ena"=1; "pclk"=P; }
V { "conf"=0; "conf_ena"=1; "pclk"=P; }
```

```
V { "conf"=1; "conf_ena"=1; "pclk"=P; }
V { "conf_ena"=0; "pclk"=0; }
```

Question 21. Since you only changed the "`test_setup`" section of the STIL test protocol, which command should you use to read it back into DFTC?

```
read_test_protocol –section test_setup
```

Question 22. What happens when try to read `orca_mapped.spf` back into DFTC?

```
dc_shell> read_test_protocol -section
test_setup orca_mapped.spf
Warning: a protocol already exists in mode
'all_dft.'                Please use
remove_test_protocol command to delete the
existing test protocol. (TEST-1402)
```

Question 23. How do the results of `dft_drc` compare to your first run?

Fewer PRE-DFT violations, but new "cell is constant" violations that reflect the 1-0-1 initialization sequence.

```
47 PRE-DFT VIOLATIONS
     1 DLAT clock line not controlled
violation (D4)
     6 Clock feeding multiple
clock/set/reset inputs violations (D12)
    20 Data path affected by clock captured
by clock in trailing edge clock_port
violations (D14)
    20 Clock_port not capable of capturing
data violations (D17)

3 OTHER VIOLATIONS
     1 Cell is constant 0 violation (TEST-
504)
     2 Cell is constant 1 violations (TEST-
505)
```

This page was intentionally left blank.

5 DFT Design Rule Checks

Learning Objectives

After completing this lab, you should be able to:

- Perform DRC checks on RTL, a pre-scan netlist, and a post-scan netlist
- Interpret the results of the DRC run from the `dft_drc` report
- Obtain an ATPG test coverage estimate

Lab Duration: 45 minutes

Overview

All files for this lab except for designs and libraries are located in the directory **`lab5_drc`**.

Directory Structure

`lab5_drc`	Current working directory
`analyzed`	Intermediate acs_read_hdl files
`logs`	Session log files
`unmapped`	Unmapped protocol
`mapped`	Gate-level netlist
`mapped_scan`	Will contain final netlist
`reports`	DFTC reports
`tmax`	Files for downstream tools
`scripts`	Constraint and run scripts
`../ref/rtl/RISC_CORE/vhdl`	Design files
`../ref/db`	Library db files

Answers & Solutions

This lab guide contains answers and solutions to all questions. If you need some help with answering a question, consult the back portion of this lab for help.

Instructions

During this lab, you will perform DFT design rule checks and various points in the scan insertion flow.

Task 1. RTL DFT DRC

1. Invoke DC/DFTC.

```
UNIX%  cd lab5_drc
UNIX%  dc_shell | tee logs/rtl_drc.log
```

2. Read the RTL design into DFTC and create the test protocol.

```
s scripts/task1_setup.tcl
```

Question 1. What command verifies that the RTL design and the protocol are compatible?

.....

Question 2. What variable needs to be set to enable file and line number tracking for RTL DFT DRC?

.....

3. Run dft_drc

```
dft_drc
```

4. Analyze the `dft_drc` report

Question 3. Were there any violations? What type?

.....

5. The TEST_MODE signal (active high) was not declared prior to creating the test protocol. Declare the signal with `set_dft_signal` and rerun `dft_drc`

```
set_dft_signal ...
remove_test_protocol
create_test_protcol –capture_procedure multi_clock
dft_drc
```

Question 4. Are there any violations now?

.....

6. Exit `dc_shell`

```
exit
```

Task 2. Pre-DFT DRC

1. Invoke DC/DFTC.

```
UNIX%  dc_shell | tee logs/gate_drc.log
```

2. Read the design and perform a test-ready compile.

```
s scripts/task2_setup.tcl
```

Question 5. What command is used to perform a test-ready compile?

.....

Question 6. From looking at the logfile, can you tell if automatic shift register identification was performed?

.....

Question 7. What command verifies that the gate-level design and the protocol are compatible?

.....

3. Run pre-DFT DRC

```
dft_drc
```

Question 8. Are the results the same as when `dft_drc` was run on the RTL? Is the report format the same?

.....

4. Enable enhanced `dft_drc` reporting and re-run dft_drc

```
set test_disable_enhanced_dft_drc_reporting FALSE
dft_drc
```

Question 9. What is the difference in the `dft_drc` report?

.....

Task 3. Post-DFT DRC

1. Specify scan constraints and preview the scan architecture that will result.

```
s scripts/task3_setup.tcl
```

Question 10. How many scan chains does `preview_dft` report will be inserted?

.....

2. Insert the scan chain(s).

```
insert_dft
```

Question 11. What command verifies that the inserted scan chain follows design rules?

.....

3. Perform DFT DRC on the post-inserted netlist

```
dft_drc
```

Question 12. Were there any DRC violations?

.....

Question 13. What section of the `dft_drc` report is now included when performing DRC check on a post-scan inserted netlist?

.....

Question 14. What dft_drc command option is used to report an ATPG test coverage estimate after scan insertion?

.....

4. Get a coverage estimate.

```
dft_drc ...
```

Question 15. What is the estimated coverage?

.....

Congratulations! You have completed the lab.

Answers / Solutions

Task 1. RTL DFT DRC

Question 1. What command verifies that the RTL design and the protocol are compatible?

`dft_drc`

Question 2. What variable needs to be set to enable file and line number tracking for RTL DFT DRC?

set hdlin_enable_rtldrc_info true

Question 3. Were there any violations? What type?

Yes. 310 D1's, 90 D3's, 1 D15

401 PRE-DFT VIOLATIONS

310 Uncontrollable clock input of flip-flop violations (D1)

90 DFF set/reset line not controlled violations (D3)

1 Clock path affected by clock captured by clock in level sensitive clock_port violation (D15)

```
set_dft_signal -view existing_dft -type Constant -
active_state 1 -port TEST_MODE
remove_test_protocol
create_test_protcol –capture_procedure multi_clock
dft_drc
```

Question 4. Are there any violations now?

No.

Task 2. Pre-DFT DRC

Question 5. What command is used to perform a test-ready compile?

`compile_ultra –scan` (or "`compile –scan`")

Question 6. From looking at the logfile, can you tell if automatic shift register identification was performed?

Yes, it is enabled.

Note: Automatic shift-register identification is enabled for scan. Not all registers will be scan-replaced (OPT-467)

Question 7. What command verifies that the gate-level design and the protocol are compatible?

`dft_drc`

Question 8. Are the results the same as when dft_drc was run on the RTL? Is the report format the same?

Yes. Still 0 violations.

Question 9. What is the difference in the `dft_drc` report?

The results are now summarized in a table

```
---------------------------------------------------------------------------
 Sequential Cell Report:       Sequential       Core         Core
                               Cells            Segments     Segment Cells
---------------------------------------------------------------------------
 Sequential Elements Detected:  303               0            0
 Clock Gating Cells          :  0
 Synchronizing Cells         :  0
 Non-Scan Elements           :  0
 Excluded Scan Elements      :  0                 0            0
 Violated Scan Elements      :  0                 0            0
 (Traced) Scan Elements      :  303     (100.0%)  0            0 (   0.0%)
---------------------------------------------------------------------------
```

Task 3. Post-DFT DRC

Question 10. How many scan chains does `preview_dft` report will be inserted?

One.

Question 11. What command verifies that the inserted scan chain follows design rules?

`dft_drc`

Question 12. Were there any DRC violations?

No.

Question 13. What section of the `dft_drc` report is now included when performing DRC check on a post-scan inserted netlist?

The “Traced Scan Chains” section

```
---------------------------------------------------------------------------

Traced Scan Chains

---------------------------------------------------------------------------

Chain "1" traced through 303 flip-flops
```

Question 14. What dft_drc command option is used to report an ATPG test coverage estimate after scan insertion?

```
help –v dft_drc
```

The option is `-coverage_estimate`

Question 16. What is the estimated coverage?

99.77%

6 Introduction to Design Vision

Learning Objectives

After completing this lab, you should be able to:

- Read a design and related files into Design Vision, the graphical interface to **Design For Test Compiler (DFTC)**
- Compile that design into a test-ready design
- Explore that design in Design Vision
- Save the test-ready design

Lab Duration: 45 minutes

Overview

All files for this lab except for designs and libraries are located in the directory **`lab6_gui`**.

Directory Structure

`lab6_gui`	Current working directory
`analyzed`	Intermediate acs_read_hdl files
`logs`	Session log files
`unmapped`	Unmapped protocol
`mapped`	Gate-level netlist
`mapped_scan`	Will contain final netlist
`reports`	DFTC reports
`tmax`	Files for downstream tools
`scripts`	Constraint and run scripts
`../ref/rtl/RISC_CORE/vhdl`	Design files
`../ref/db`	Library db files

Answers & Solutions

This lab guide contains answers and solutions to all questions. If you need some help with answering a question, consult the back portion of this lab for help.

Instructions

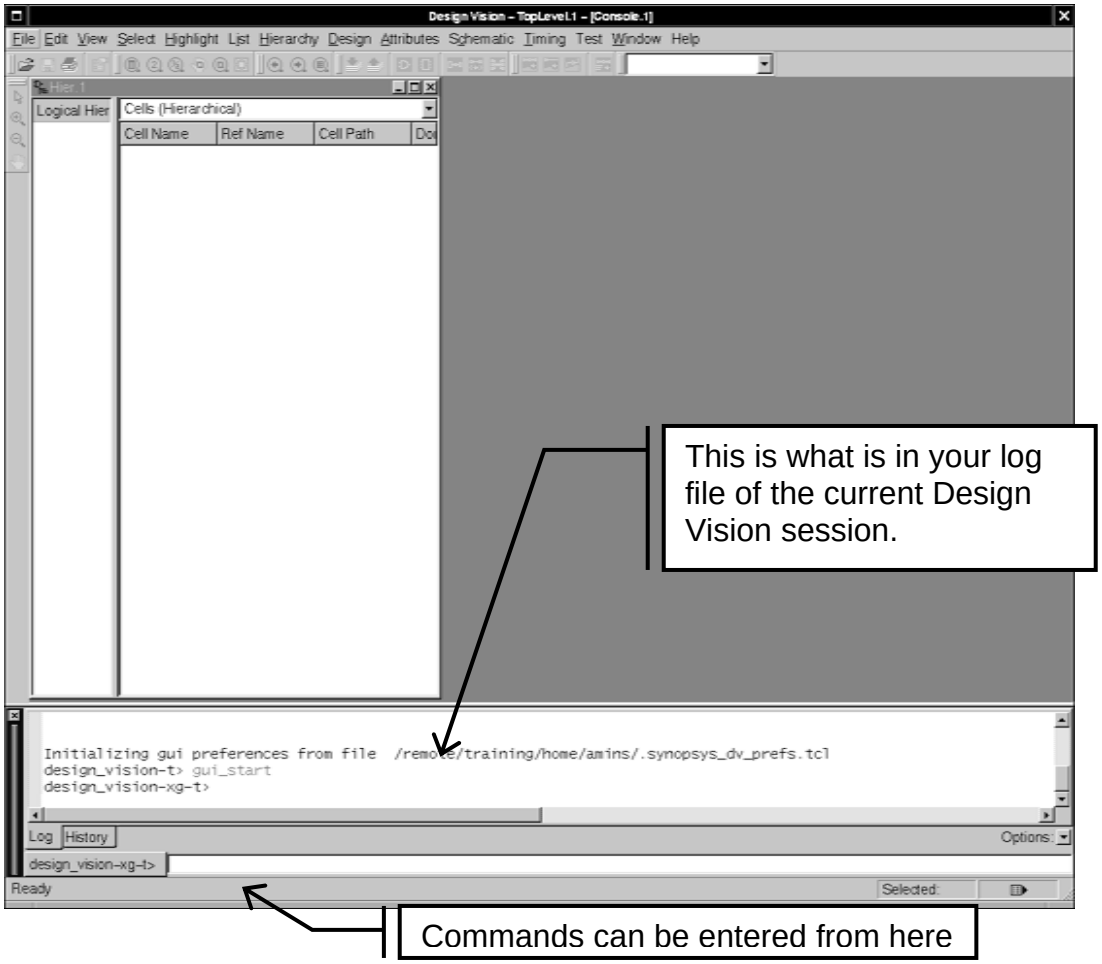
Task 1. Read a design into Design Vision

1. Invoke Design Vision.

```
UNIX%  cd lab6_gui
UNIX%  design_vision | tee logs/gui.log
```

Question 1. What is the difference between **Design Vision** and **`dc_shell`**?

.....



Once in Design Vision, you can take advantage of its graphical interface for debugging and troubleshooting problems, or work in `dc_shell`, taking advantage of the speed of a shell.

2. Read the file containing the `RISC_CORE` entity or module, and read all associated subfiles, by applying the following command:

   Before executing the command, click on the **Log tab** in the lower left corner of the GUI, and make the Session Window as large as possible, so that you can follow the progress of the read.

```
acs_read_hdl -f vhdl \
      -hdl_source ../ref/rtl/RISC_CORE/vhdl  RISC_CORE
```

Question 2. What does the **`acs_read_hdl`** command do in addition to reading in the model you specify?

.....

Task 2. Compile and Save the Design

1. **Constrain** the design using the constraints file provided for you in the scripts/ directory called: constraints.tcl.

   Design timing and area constraints are needed to direct the next step: compile.

Note: These constraints drive the standard Design Compiler logic synthesis, and are not the focus in this class. Do not spend your lab time trying to learn them.

```
source -e -v constraints.tcl
```

Question 3. The constraint file is in the **scripts** subdirectory; you did not have to specify the directory when you sourced the constraints file, why not?

.....

2. **Compile** the design using the following command:

```
compile_ultra -scan
```

   The compile command synthesizes the design to the target library by taking the GTECH (Generic Technology) library parts and translating them into the target library parts (in this case, a 0.13µ training library).

   Compile maps to ordinary D flip-flops by default. The `-scan` switch directs DC to translate the GTECH sequential elements directly to special test flip-flops called scan flops. This is called “test-ready compile” or “1-Pass Scan Synthesis”.

3. **Save** the design using the command:

```
write -format ddc -hierarchy \
       -output mapped/RISC_CORE.ddc
```

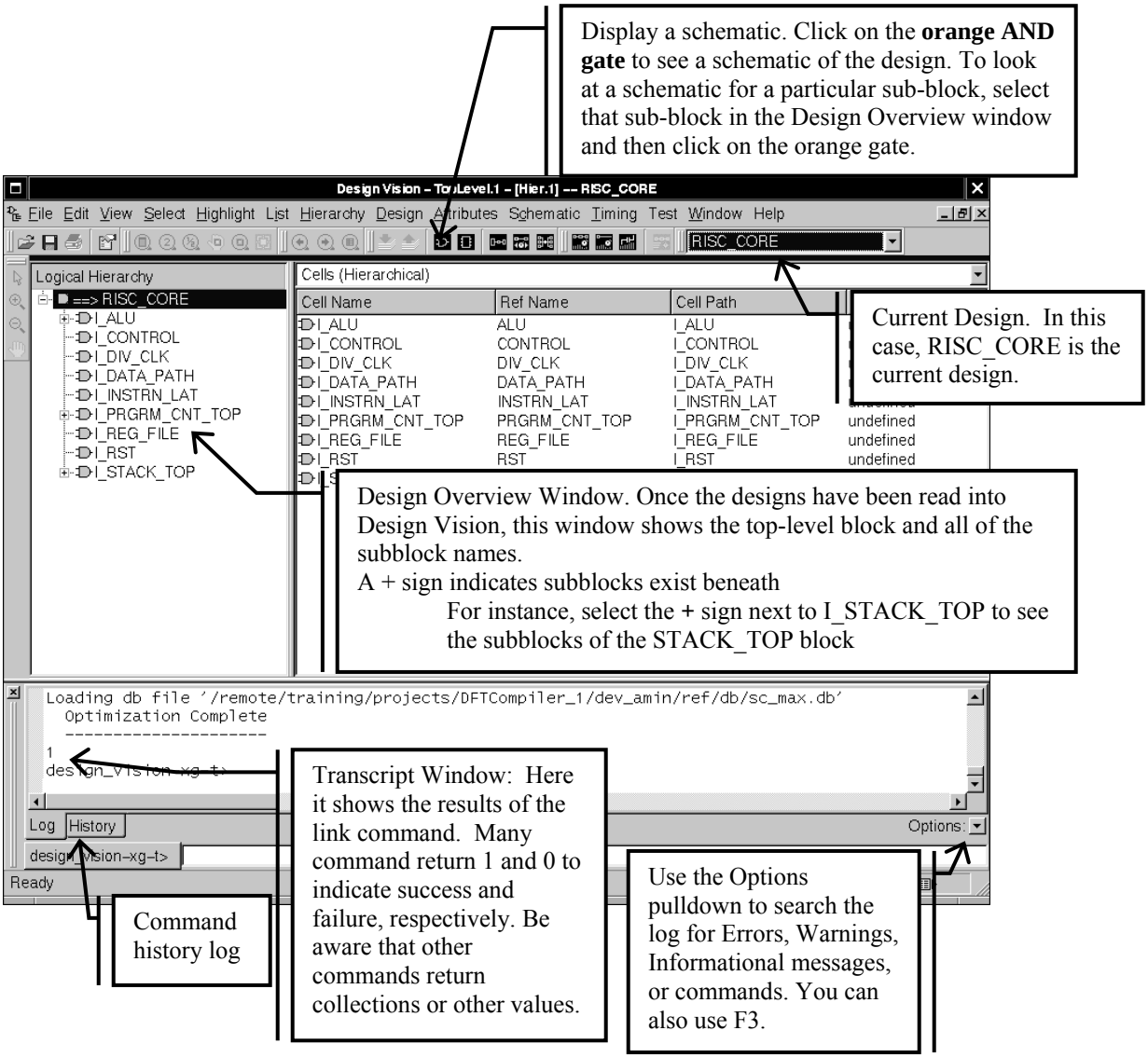
This command saves your design in Synopsys internal (DDC) format into the mapped subdirectory. The name of the file is `RISC_CORE.ddc`. Saving all the sub blocks in the design is enabled by the `-hierarchy` switch.

Task 3. Explore the Design

When inserting scan into your design, problems may occur. Being able to explore your design graphically is a powerful debugging aid.

1. Resize your Session Window, making it smaller so that you can focus on the graphical display at the top portion of the GUI.

The following figure shows some key points for navigating in Design Vision.



Question 4. What is the current design?

.....

2. Display the current design from the TCL shell.

You can also display the current design from the TCL shell by using the `current_design` command. Try it:

```
current_design
```

3. Display all the designs from the TCL shell by using the `list_designs` command.

```
list_designs
```

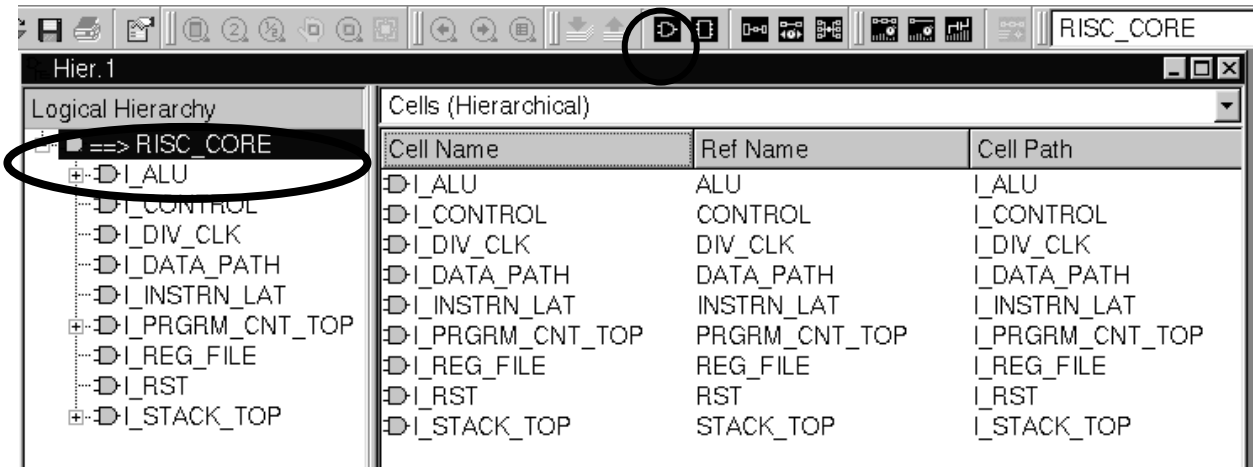
4. Display all the designs graphically.

Select the pull down arrow to the right of the current design; the results match those from the `list_designs` command.

5. Display a schematic of the entire RISC_CORE ASIC.

Specify which part of the ASIC you want to see.

In this case, select RISC_CORE from the Design Overview Window. Notice a list of the subdesigns below RISC_CORE is displayed



Notice the Cell name and ref name of each subdesign are shown. **Cells** are also called **instances**. **Ref names** are also called **designs**.

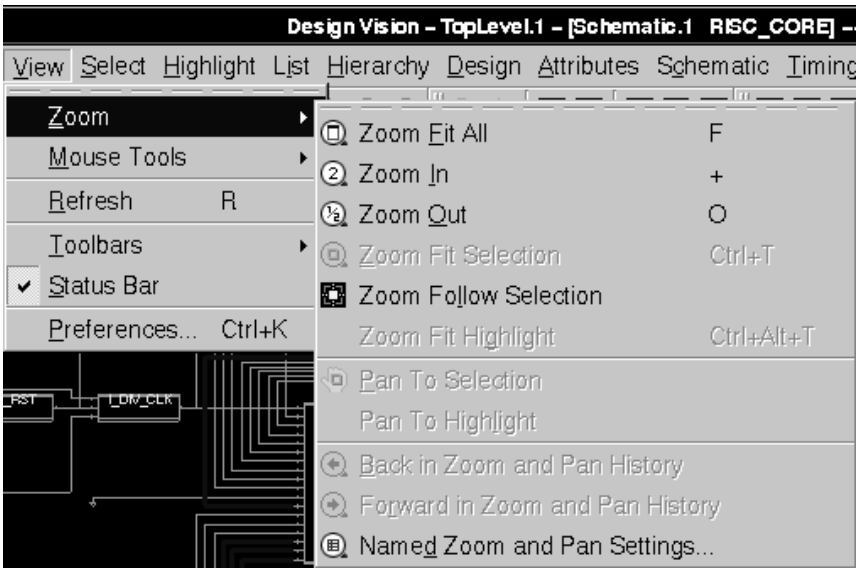
6. Display a schematic of the RISC_CORE ASIC.

Select the orange AND gate in the toolbar. You should see a schematic of the RISC_CORE ASIC.

You can zoom in and out to see things more clearly.

7. Zoom in to examine the schematic in more detail.

Select the **view** pull-down menu to zoom in and out



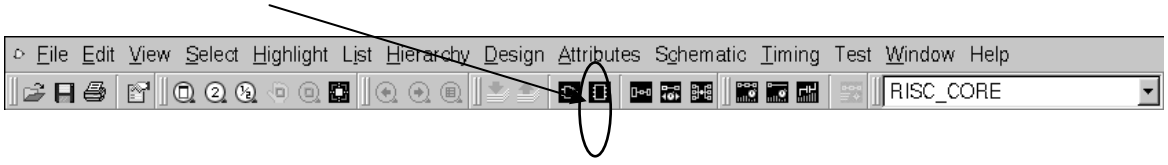
Note: You may first have to select an area of the schematic which has nothing in it with the mouse to be able to use the Zoom features.

Once in Zoom In mode, you may need to click on the Selection Tool before changing to another mode, such as Zoom Out. The escape key (ESC) is the shortcut for the Selection Tool.

You can also use the middle mouse button to zoom and pan, using so called strokes, e.g. depress the middle mouse button then move the mouse from SW to NE. You should have zoomed in.

8. Display the symbol view (input and output ports) of the RISC_CORE ASIC.

Select the green IC chip icon next to the orange AND gate.



9. Verify that your entire design is mapped to the gates.

The **`report_hierarchy`** command shows you the entire hierarchy of the design in a textual format, as well as the cells in the design and libraries to which they belong. In this case, you are using the **tsmc** library. If any cells belong to the **gtech** library, you can conclude that there are unmapped cells in your design.

Run the command:

```
report_hierarchy
```

10. Compare the previous result to an unmapped design.

Remove all of the designs, bring in the unmapped design once more, and run the `report_hierarchy` command:

```
remove_design –design
acs_read_hdl -f vhdl \
    –hdl_source ../ref/rtl/RISC_CORE/vhdl   RISC_CORE
report_hierarchy
```

Notice the RPT-7 message and that all the cells are GTECH library cells because the design is unmapped.

11. Verify that all the library variables have been defined correctly.

```
printvar *_library
```

12. Exit Design Vision

```
exit
```

Task 4. Debug Violations using the GUI

13. Invoke Design Vision

```
UNIX%  design_vision | tee logs/debug.log
```

14. Read the mapped design saved in the previous task

```
read_ddc mapped/RISC_CORE.ddc
```

15. Run a design rule check.

```
dft_drc
```

Question 5. What error message do you get?

.....

16. Resolve DFTC's complaint; that is, create a test protocol.

```
create_test_protocol
```

17. Run a design rule check again.

```
dft_drc
```

Question 6. Is the design free of DFT problems or does the report contain many violations?

.....

18. Look at the GUI. You should see a Violation Browser showing the same errors shown by the command output of dft_drc. You can bring up the violation browser any time using the menu **Test → Browse Violations**.

19. Click on the "+" infront of "PRE-DFT", then click on the D1 category.

Select the first violation on the right. On the bottom, you will see an explanation of the error.

20. Right click on the first D1 violation, then select "Inspect Violation", or click on the "Inspect Violation" button on the lower right. A schematic should appear.

Question 7. What logic value is present on the clock?

.....

Question 8. What DFTC command is needed to specify this test clock?

.....

21. Type the command that you just noted (check the back for help).

22. Recreate the test protocol, then switch back to the Violation Browser, and click on “Run DFT DRC”. The schematic window should disappear.

Note: The test protocol is recreated by simply re-running the create_test_protocol command.

Question 9. Was the number of violations affected at all?

.....

23. Once again, “Inspect Violation” for the first D1 violation shown.

Question 10. What value is shown for TEST_MODE?

.....

Question 11. What DFTC command is needed to specify this test mode?

.....

24. Type the command that you just noted (check the back for help).

25. Recreate the test protocol, then switch back to the Violation Browser, and click on “Run DFT DRC”.

Question 12. Was the number of violations affected at all?

.....

26. Analyze the first D3 violation.

Question 13. What value is shown for Reset?

.....

Question 14. What DFTC command is needed to specify this asynchronous reset (the reset is active low)?

.....

.....

27. Type the command you just noted down (check the back for help).

28. Recreate the test protocol, then switch back to the Violation Browser, and click on “Run DFT DRC”.

Question 15. Was the number of violations affected at all?

.....

29. Exit Design Vision.

Congratulations! You have completed the lab.

Answers / Solutions

Task 1. Read a design into Design Vision

Question 1. What is the difference between **Design Vision** and **`dc_shell`**?

Design Vision (design_vision) is the GUI to the Synopsys logic synthesis tools; it is built on **`dc_shell,`** which is the command line interface.

Question 2. What does the `acs_read_hdl` command do in addition to reading in the model you specify?

It finds the files containing all the designs and subdesigns, reads them in, and links them, setting the `current_design` to the top-level design.

Task 2. Compile and Save the Design

Question 3. The constraint file is in the **scripts** subdirectory; you did not have to specify the directory when you sourced the constraints file, why not?

The "scripts" directory was appended to the `search_path` variable in .synopsys_dc.setup. You can verify this by using `"printvar *path".`

Task 3. Explore the Design

Question 4. What is the current design?

The current design is: RISC_CORE (top-level of the ASIC)

Task 4. Debug Violations using the GUI

Question 5. What error message do you get?

```
Error: No model found on design
0
```

Question 6. Is the design free of DFT problems or does the report contain many violations?

```
394 PRE-DFT VIOLATIONS
   303 Uncontrollable clock input of
flip-flop violations (D1)
      1 DFF set/reset line not
controlled violation (D2)
     90 DFF set/reset line not
controlled violations (D3)
```

Question 7. What logic value is present on the clock?

X meaning unknown or unspecified in this case.

Question 8. What DFTC command is needed to specify this test clock?

```
set_dft_signal -view exist –type ScanClock \
               -timing {45 55} -port Clk
```

Question 9. Was the number of violations affected at all?

Yes, the number of D1 violations goes down to 300.

Question 10. What value is shown for TEST_MODE?

X meaning unknown or unspecified in this case.

Question 11. What DFTC command is needed to specify this test mode?

```
set_dft_signal -view exist -type Constant \
               -active 1 -port TEST_MODE
```

Question 12. Was the number of violations affected at all?

All D1 violations are gone now. 1 D2 and 90 D3 violations remain.

Question 13. What value is shown for Reset?

X meaning unknown or unspecified in this case.

Question 14. What DFTC command is needed to specify this asynchronous reset?

```
set_dft_signal -view exist -type Reset \
               -active 0 -port Reset
```

Question 15. Was the number of violations affected at all?

All violations should be gone now.

Question 16. How do you optimize, map, and scan-replace the flip-flops in a design with **Design Vision**?

Use the **`compile -scan`** command

Command Sequence For This Lab

```
acs_read_hdl -f vhdl \
   –hdl_source ../ref/rtl/RISC_CORE/vhdl RISC_CORE
source -e -v constraints.tcl
compile_ultra -scan
write –format ddc –hier -output mapped/RISC_CORE.ddc
set_dft_signal -view exist -type ScanClock \
         -timing {45 55} -port Clk
set_dft_signal -view exist -type Constant \
         -active 1 -port TEST_MODE
set_dft_signal -view exist -type Reset \
         -active 0 -port Reset
create_test_protocol
dft_drc
```

Appendix: Searching Synopsys Online Documentation using Indexes

1. Synopys stores its product documentation in a collection called **Synopsys Online Documentation** (**SOLD**). In addition to the command man pages, **SOLD** is a good place for you to look for more information about Synopsys tools. To access SOLD type **sold** from the UNIX command line followed by an ampersand (&).

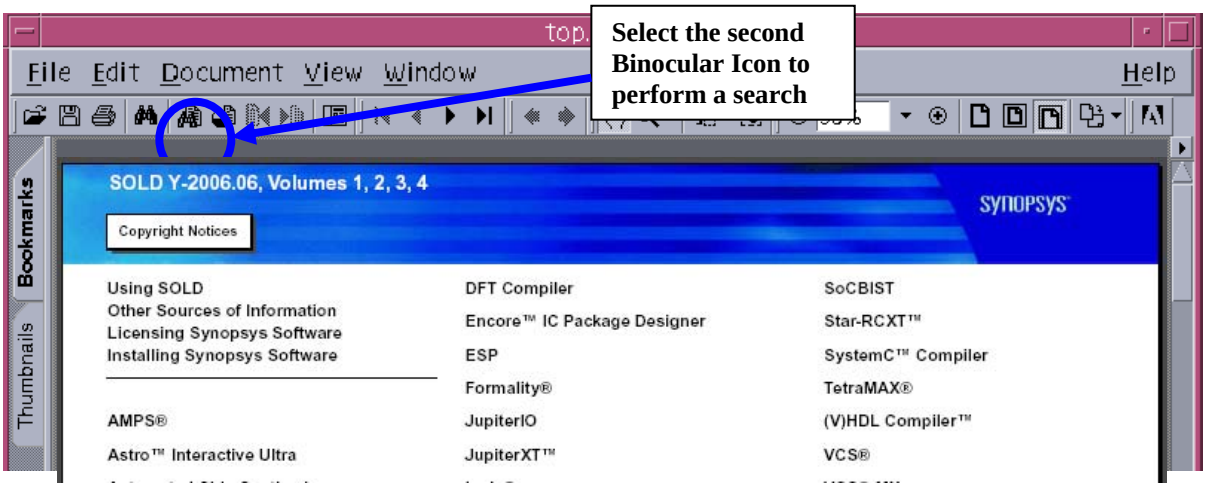
```
UNIX> sold &
```

sold is a UNIX alias which invokes **Adobe Acrobat reader** and opens up a file called **top.pdf**, which is the top-level view of all documents.

2. To search for information in SOLD and look only in specific documents, you need to **set up indexes** to customize the searches. SOLD consists of a host of documentation for the various Synopsys tools. Each set of documentation is accessed through its **corresponding index**. For instance, there is an index for the Man Pages and one for **Test Automation**, as well as many others. When the indexes are set up properly, you can use SOLD to **search in one or many indexes** to control how much information you will receive from any search you perform.

3. For information on how to setup indexes consult this SolvNet article https://solvnet.synopsys.com/retrieve/903898.html "Script for building SOLD search indexes".

4. To search for information in a SOLD index, first select the binoculars icon (the binoculars icon with documents behind it is the Search button).



7 Debug and Fix DFT DRC Violations

Learning Objectives

After completing this lab, you should be able to:

- Debug dft_drc problems with clocks and resets
- Propose a manual solution that could be implemented by you or by another designer
- Use AutoFix to implement a fix

Lab Duration: 45 minutes

Overview

All files for this lab except for designs and libraries are located in the directory **`lab7_fixing`**.

Directory Structure

`Lab7_fixing`	Current working directory
`analyzed`	Intermediate acs_read_hdl files
`logs`	Session log files
`unmapped`	Unmapped protocol
`mapped`	Gate-level netlist
`reports`	DFTC reports
`tmax`	Files for downstream tools
`scripts`	Constraint and run scripts
`../ref/rtl/RISC_CORE_nodft/vhdl`	Design files
`../ref/db`	Library files
`./.solutions`	Solution files

Answers & Solutions

This lab guide contains answers and solutions to all questions. If you need some help with answering a question, consult the back portion of this lab for help.

Instructions

During this lab you will debug Uncontrollable Clock and Reset DFT problems using the Design Vision GUI, propose a fix and implement a fix using AutoFix.

Task 1. Read in Mapped Design

1. Change directories to the project directory **`lab7_fixing`**.

2. Invoke DC in the GUI mode.

```
UNIX% design_vision
```

3. Read in the mapped design. Remember file name completion!

```
source scripts/4read_gate_and_protocol.tcl
```

Task 2. Debug Uncontrollable Clocks and Resets

1. From information in the log window, answer the following:

Question 1. How many violations are there?
What type of violations are they?

.....

.....

.....

2. In the Violation Browser, analyze the first D1 violation and you should observe a schematic similar to the one shown below.

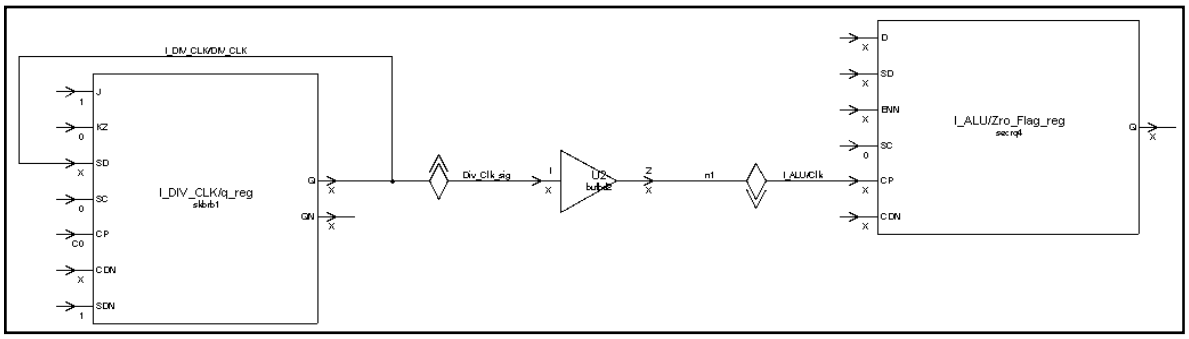


Figure 1.

Question 2. What is the netlist name for the flip-flop on the left, the cause for the D1 violation?

.....

Question 3. Draw a schematic "fix" for this uncontrollable clock problem.

.....

3. Analyze some of the other D1 violations. Select the Violation Browser tab, then select some of the other D1 violations and examine the schematic.

Question 4. What can you conclude about the D1 violations in the RISC_CORE design?

.....

Analyze one of the D3 violations in the same manner as you did the D1 violations. To expand the second X input to I_RST/q2_reg, double click on the D input.

Question 5. Based on your analysis draw a schematic "fix" for the uncontrollable reset problem.

.. ..

4. Select some of the other D3 violations and analyze them.

Question 6. What can you conclude about the D3 violations in the RISC_CORE design?

.....

Task 3. Autofix The D1, D2, and D3 violations

You have now analyzed both the uncontrollable clock and uncontrollable reset DFT violations in the RISC_CORE design. Your next task is to fix the violations. As you learned in the lecture there are three choices.

1) You could re-code the RTL design and check the design once again using the unmapped DFTflow. When you have no more violations you can continue on to the mapped flow.
2) You could draw the solution and have someone else re-code the design for you.
3) You can use the AutoFix capability of DFTC.

Question 7. What kind of DFT problems can Autofix solve?

.....

.....

1. The 5preview_dft.tcl file is setup to **source** the autofix.tcl script **after** sourcing the **settings_insert_dft.tcl** file.

Set up to run AutoFix by modifying a script called **scripts/autofix.tcl**.

Utilize the lecture material and the man pages to modify the script called **autofix.tcl** in the **scripts/** directory. Enable AutoFix to fix uncontrolled clocks, uncontrolled resets, and uncontrolled sets.

Here is a list of commands related to AutoFix to get you started:

- **`set_dft_configuration`**
- **`set_autofix_configuration`**
- **`set_dft_signal`**

2. Preview what AutoFix will add by running step 5 of the Mapped Flow.

```
source –e –v scripts/5preview_dft.tcl
```

3. Evaluate the previewed AutoFix results.

Take a careful look at the results of running the `preview_dft` command. Notice the sections of the report titled **Test Point Plan Report**, and **DFT signals**. These sections relate directly to the AutoFix portion of the flow.

Compare your results carefully with those in the Answers / Solutions section—it is easy to make a mistake and not realize it.

Question 8. How many AutoFix test points are there? Why?

.....

Question 9. What do you think would happen if a register had an unclontrolled reset **and** and uncontrolled set and the same top-level signal was specifyed as the TestData?

.....

4. Perform both AutoFix and scan insertion by running step 6 of the DFT flow.

```
source –e –v scripts/6insert_dft.tcl
```

5. Look at the results of runing `dft_drc –coverage_estimate`.

Question 10. What were the number of violations?
Were there any cells with violations?
What was the fault coverage?

.....

.....

Question 11. What kind of fault coverage would you expect to get, if you had not AutoFixed the design?

.....

6. Try it and compare coverages.

7. Exit DFTC.

Congratulations! You have completed the lab.

Answers / Solutions

Task 2. Debug Uncontrollable Clocks and Resets

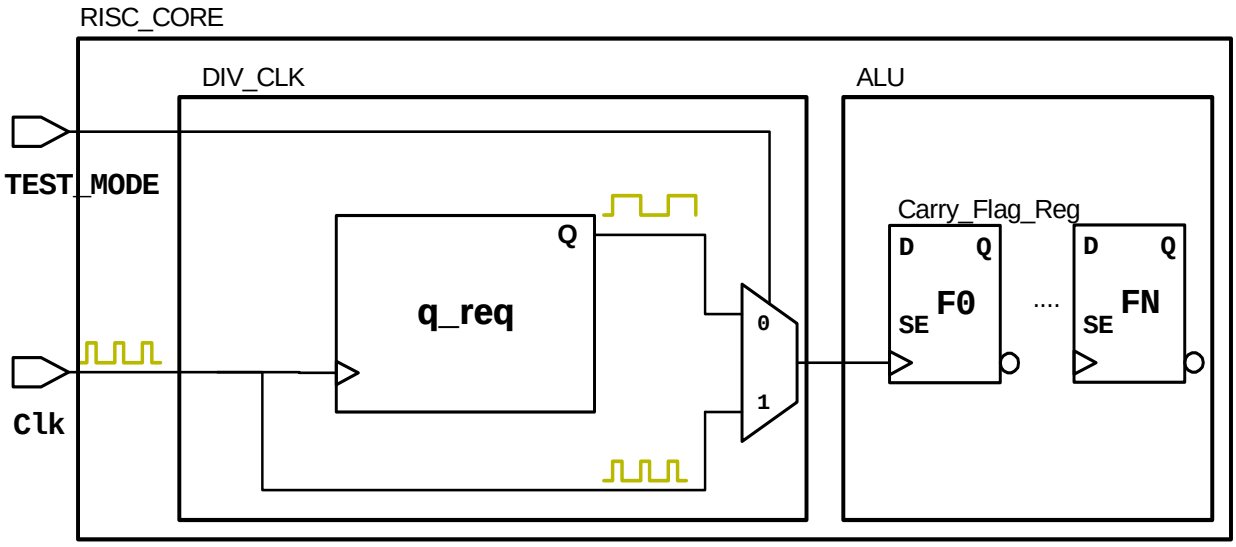
Question 1. How many violations are there? What type of violations are they?

```
389 PRE-DFT VIOLATIONS
   300 Uncontrollable clock input of flip-flop violations (D1)
     1 DFF set/reset line not controlled violation (D2)
    88 DFF set/reset line not controlled violations (D3)
```

Question 2. What is the netlist name for the flip-flop on the left, the cause for the D1 violation?

I_DIV_CLK/q_reg

Question 3. Draw a schematic "fix" for this uncontrollable clock problem.

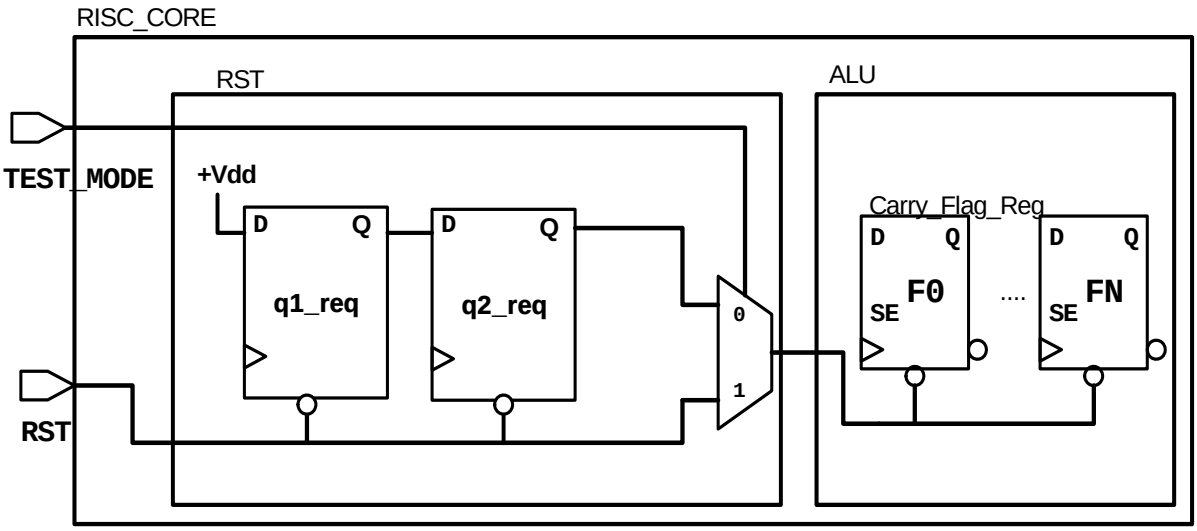


Note: The image above only shows I_DIV_CLK/q_req/Q going to the ALU block in RISC_CORE, but it is actually connected to several other blocks (a total of 310 FFs).

Question 4. What can you conclude about the D1 violations in the RISC_CORE design?

All the D1 violations are coming from the same launch point and have the same solution.

Question 5. Based on your analysis draw a schematic “fix” for the uncontrollable reset problem.



Question 6. What can you conclude about the D3 violations in the RISC_CORE design?

The D3 violations all come from the same launch point and the solution in Question 5 is adequate to fix all of the D3 violations.

Task 3. AutoFix The D1, D2, and D3 violations

Question 7. What kind of problems can AutoFix solve?

Uncontrollable clocks and sets/resets, contention issues during scan shift on internal tristate busses and bidirectional ports.

Set up to run Autofix by modifying the 5preview_dft.tcl file.

Contents of file **autofix.tcl** (also available in the .solutions directory)**:**

```
set_dft_configuration -fix_set enable \
           -fix_reset enable -fix_clock enable
set_dft_signal -view spec -type TestMode \
                     -port TEST_MODE


# use the clock Clk as the autofix clock
set_dft_signal -view spec -port Clk -type TestData
set_autofix_configuration -type clock \
                   -control TEST_MODE -test_data Clk

# Use the Reset line as the reset fix
set_dft_signal -view spec -port Reset -type TestData
set_autofix_configuration -type reset –method mux \
                   -control TEST_MODE -test_data Reset

# Use the same Reset line to fix the sets
set_autofix_configuration -type set -method mux \
                   -control TEST_MODE -test_data Reset
```

```
************ Test Point Plan Report ************
Total number of test points  : 2
Number of Autofix test points: 2
Number of Wrapper test points: 0
Number of test modes         : 1
Number of test point enables : 0
Number of data sources       : 2
Number of data sinks         : 0
************************************************

Dft signals:
 TestMode: TEST_MODE (no hookup pin)
 TestData: Clk (no hookup pin)
 TestData: Reset (no hookup pin)
```

Question 8. How many AutoFix test points are there? Why?

2 AutoFix test points. One to fix the uncontrollable clocks and one to fix the uncontrollable resets and sets.

Question 9. What do you think would happen if a register had an unclontrolled reset **and** uncontrolled set and the same top-level signal was specifyed as the TestData?

This situation would create a race condition if the same signal controlled both the reset and set pin of scan register. Therefore, DFTC will not allow this condition and will create a new port to fix the sets.

If in this situation and you don't want to use a separate port to fix the sets, you have a couple options:

- Use a the "gate" method to fix all sets and/or resets
- Use set_autofix_element to specify the "gate" method to fix uncontrolled sets or resets for only the specific registers that have both and uncontrolled reset and set.

Question 10. What were the number of violations?
Were there any cells with violations?
What was the fault coverage?

0 violations.

No cells with violations.

99.73%

Question 11. What kind of fault coverage would you expect to get, if you had not AutoFixed the design?

Very low (0.36%) because almost every flip-flop in the design will not be included on the scan chain.

8 Top-Down Scan Insertion

Learning Objectives

After completing this lab, you should be able to:

- Use the preview_dft command to evaluate different scan architectures and iterate by revising scan specifications until you produce balanced top-level scan chains.
- Use the `preview_dft` command to confirm your scan specification for desired scan signals was applied correctly.

Lab Duration: 45 minutes

Overview

During this lab you will:

1. Preview the results of different scan specifications before implementing one that yields well-balanced top-level chains.
2. Perform the Mapped DFT flow for a test-ready design through scan insertion and test coverage estimation.

Unmapped Flow:

1. Read the RTL design into DFT.
2. Create and save a test protocol; verify that the design and the test protocol are compatible.
3. Compile and save the gate-level design; exit.

Mapped Flow:

4. Read the gate-level design and test protocol into DFTC.
5. Specify scan constraints and preview the scan architecture that will result from applying them.
6. Insert the scan chain.
7. Hand off the design and related files for use by downstream tools; exit.

File Locations

All files for this lab except for designs and libraries are located in the directory **`lab8_topdown`**.

Directory Structure

`Lab8_topdown`	Current working directory
`analyzed`	Intermediate acs_read_hdl files
`logs`	Session log files
`unmapped`	Unmapped protocol
`mapped`	Gate-level netlist
`mapped_scan`	Gate-level scan design netlist
`reports`	DFTC reports
`tmax`	Files for downstream tools
`scripts`	Constraint and run scripts
`../ref/rtl/ORCA/vhdl`	Design files
`../ref/db`	Library files

Instructions

Your goal is to complete and run the first three steps of the Mapped Flow portion of the DFT Insertion Flow. The three you will modify are for reading the test-ready design and test protocol, previewing the scan architecture, inserting scan chains and finally, estimating test coverage.

The three scripts:

```
Unmapped Flow
  1read_design.tcl
  2create_test_protocol.tcl
  3compile_and_save.tcl
Mapped Flow
  4read_gates_and_protocol.tcl
  5preview_dft.tcl
  6insert_dft.tcl
  7handoff.tcl
```

Task 1. Read Gate-level Design and its Test Protocol

1. Invoke DFTC.

```
cd lab8_topdown
dc_shell | tee logs/lab8.log
```

2. Examine and execute the script that performs step 4 of the DFT flow: reading in the gate-level design and the saved protocol.

```
exec cat scripts/4read_gate_and_protocol.tcl


s scripts/4read_gate_and_protocol.tcl
```

3. Check the scan state of the current design using this command:

```
report_scan_state
```

Question 1. If the Unmapped Flow ended with a `compile -scan` command, what should the scan state of the design you just read in be? Is that the design's current scan state?

.....

At the end of the **4read_gate_and_protocol.tcl**, a DFT check was performed.

4. Examine the last several lines in your transcript to review the DFT check results for the test-ready design.

Question 2. What percentage of the total number of flip-flops are valid scan cells?

.....

Question 3. Considering just that percentage and no other factors, what range of estimated fault coverage would you predict for this design?

.....

5. Enable enhanced DRC reporting and re-run `dft_drc`

```
set test_disable_enhanced_dft_drc_reporting FALSE
dft_drc
```

Question 4. Does the percentage reported by the enhanced `dft_drc` reporting match your expectations?

.....

Note: Step 4 of the DFT flow reads the gate-level design and the test protocol. The dft_drc command used the saved test protocol for DFT design rule checking. The test protocol is one set of information and the scan specifications that may have been used to originally create the test protocol is another set of information.

6. Check this by reporting which dft signals are currently defined.

```
report_dft_signal
```

Question 5. How many ScanClocks and Resets are reported?

.....

Task 2. Specify and Preview Scan Architectures

1. Display the default scan architecture that will be implemented for this design.

```
preview_dft
```

Question 6. How many scan chains will be inserted?
What are the names of the preview scan chains?
Will the chains be well-balanced?

.....

.....

.....

Question 7. Will scan insertion use existing functional pins as the scan signals (scan in, scan out for each scan chain) or will it create new ones (test_si, test_so ports)?

.....

2. Display more scan architecture detail about the test clocks used for each scan chain.

```
preview_dft -show scan_clocks
```

Note: The ORCA design has only 3 clocks in test mode: pclk, sys_clk and sdr_clk.

Question 8. Why are four scan chains previewed by default?

.....

3. Use another `preview_dft` option and compare the content of the former and current `preview_dft` output.

```
preview_dft -show scan_summary
```

Question 9. Is there any information in the scan_clocks preview that is not also reported in the scan_summary preview?
When might you want to use the format of the scan_summary preview rather than the first?

.....

.....

.....

4. Try mixing the rising and falling sdr_clk scan flip-flops into the same chain.

```
set_scan_configuration -clock_mixing mix_edges
preview_dft -show scan_clocks
```

Question 10. How many scan chains are previewed now and are they now balanced?

.....

5. Allow mixing of different clock domain scan flip-flops into the same scan chain.

```
set_scan_configuration -clock_mixing mix_clocks
preview_dft -show scan_clocks
```

Question 11. How many scan chains are previewed now and what do you think the default `set_scan_configuration –chain_count` value is?

.....

Question 12. What specifications would yield 6 balanced scan chains?

.....

6. Apply those specifications (6 chains) and confirm using preview_dft.

```
set_scan_configuration    # specify your options
preview_dft -show scan_clocks
```

Question 13. What is the difference in scan chain length between the longest and shortest chains now?

.....

.....

Question 14. Will DFTC insert any lock-up latches into this scan design? Which chains will include them?

.....

.....

Note: The functional pins of ORCA to use for scan are specified in the settings_insert_dft.tcl script.

7. Source that script and use `preview_dft` to answer the following questions.

```
s scripts/settings_insert_dft.tcl
preview_dft
```

Question 15. What are the names of the previewed scan chains now and what command defined their new names?
Why were the chains given explicit names?

.....

.....

Question 16. Which pins are used for scan in, scan out and scan enable?

.....

.....

Task 3. Insert Scan Chains and Estimate Test Coverage

One of the commands in the settings_insert_dft.tcl script avoids all the subdesigns being renamed during the scan insertion process.

Question 17. What is the name of that command?

.....

Note: Use the man pages if needed to find the option to this command that disables all the gate-level optimization that occur by default during the last phases of the `insert_dft` process.

Question 18. Which option to this command will disable gate-level optimization?

.....

1. Apply those two options whose effects will be to speed-up the runtime of the subsequent `insert_dft` and to limit the number of changes to the existing design.

```
set_dft_insertion_configuration . . .
```

2. Perform scan insertion using the script provided.

```
s scripts/6insert_dft.tcl
```

Question 19. What is the estimated test coverage for the ORCA scan design?

.....

Note: Actual ATPG runs in TetraMAX obtained about 99% test coverage for this design. Doing so though required using functional RAM models and both TetraMAX's Fast Sequential and Full Sequential engines. Using DFT techniques to avoid the S30 violation, for example, allows that very high coverage much easier to obtain by merely using the Fast Sequential engine.

Question 20. How does this estimate compare to your earlier prediction?

.....

Note: When obtaining the estimated fault coverage for a scan inserted design, `dft_drc` performs Post-DFT checking and reports the DFT violations using the same ones that TetraMAX does.

Note: You can obtain detailed information for both the D* PRE-DFT violations and the {C,S,Z,X}* Post-DFT violations using the man command in DFTC.

3. Scroll back from the coverage report and answer the following questions.

Question 21. The S category of violations cannot be reported during Pre-DFT checks, why?

.....

Question 22. What does the S22 violation mean and does this explain why 2 Sequential Cells are reported as synchronization elements in the **Sequential Cells Without Violations** summary?

.....

.....

Congratulations! You have completed the lab.

Answers / Solutions

Task 1. Read gate-level Design and its Test Protocol

Question 1. If the Unmapped Flow ended with a `compile -scan` command, what should be the scan state of the design you just read in? Is that the design's current scan state?

The scan state should be "test-ready" which is reported as "scan cells replaced with loops".

Question 2. What percentage of the total number of flip-flops are valid scan cells?

2925 / (2925 + 33) equals 98.9%

Question 3. Considering just that percentage and no other factors, what range of estimated fault coverage would you predict for this design?

Close to 99% test coverage.

Question 4. Does the percentage reported by the enhanced dft_drc reporting match your expectations?

Yes. It reports 98.9%.

Question 5. How many ScanClocks and Resets are reported?

Three ScanClocks (sys_clk, sdr_clk, and pclk) and one Reset (prst_n) are reported. The DFT signals are defined when the test protocol is read with the read_test_protocol command.

Task 2. Specify and Preview Scan Architectures

Question 6. How many scan chains will be inserted?
What are the names of the preview scan chains?
Will the chains be well-balanced?

4 scan chains
Named '1', '2', '3', and '4'.
No, the longest scan chain has 1074 cells while the shortest chain has half that many, 512, a very unbalanced scan chain architecture.

Question 7. Will scan insertion use existing functional pins as the scan signals (scan in, scan out for each scan chain) or will it create new ones (test_si, test_so ports)?

New test_si and test_so ports are previewed because you have not yet specified your scan signals.

Question 8. Why are four scan chains previewed by default?

Default is to not mix clocks or edges on scan chains. There are three clock domains. The sdr_clk domain has both rising-edge and falling-edge triggered flip-flops.

Question 9. Is there any information in the scan_clocks preview that is not also reported in the scan_summary preview? When might you want to use the format of the scan_summary preview rather than the first?

The previewed information is almost the same. The key difference is that the scan_clocks preview gives the head and tail of each scan chain, plus the first scan flip-flop whenever the scan path crosses clock domains. The scan_summary only previews the head scan flip-flop. The scan_summary format of the preview report might be easier to parse with Perl or TCL.

Question 10. How many scan chains are previewed now and are they now balanced?

Three chains, still unbalanced with the longest chain having 1132 cells and the shortest chain having 719 cells.

Question 11. How many scan chains are previewed now and what do you think the default `set_scan_configuration \ –chain_count` value is?

Only one chain, the default for `–chain_count`.

Question 12. What specifications would yield 6 balanced scan chains?

```
set_scan_configuration \
 -chain_count 6 -clock_mixing mix_clocks
```

Question 13. What is the difference in scan chain length between the longest and shortest chains now?

Only one, with the longest chains being 488 cells long and the shortest 487.

Question 14. Will DFTC insert any lock-up latches into this scan design? Which chains will include them?

Yes.
Chain ‘4’ has both pclk and sdr_clk rising-edge flops at time 45 and Chain ‘5’ has both sdr_clk and sys_clk rising-edge flops at time 45.

Question 15. What are the names of the previewed scan chains now and what command defined their new names? Why were the chains given explicit names?

Chain names: chain0, chain1, chain2, chain3, chain4 and chain5.
The set_scan_path command was used to define the scan chain names.
To associate a given pair of ScanDataIn and ScanDataOut scan signals to a specific chain. You need to specify the name of chain when you apply the `set_scan_path` command.

Question 16. Which pins are used for scan in, scan out and scan enable?

pads[0] to pad[5] are the scan in pins and sd_A[0] to sd_A[5] are the corresponding scan out pins. The scan enable port is called scan_en.

Task 3. Insert Scan Chains and Estimate Test Coverage

Question 17. What is the name of that command?

`set_dft_insertion_configuration`

Question 18. Which option to this command will disable gate-level optimization?

```
set_dft_insertion_configuration -synthesis none \
    -preserve_design_name true
```

Question 19. What is the estimated test coverage for the ORCA scan design?

~95%

Question 20. How does this estimate compare to your earlier prediction?

This is probably lower than what you predicted, likely 99%

Question 21. The S category of violations cannot be reported during Pre-DFT checks, why?

There are no scan chains then, and the S rules are related to problems with scan chain shifting.

Question 22. What does the S22 violation mean and does this explain why 2 Sequential Cells are reported as synchronization elements in the **Sequential Cells Without Violations** summary?

S22 warns that ORCA has scan chains that are driven by more than one test clock. Scan chains chain3 and chain4 have lock-up latches and thus avoid the S22 violation. The S22 warnings about a negative-edge triggered scan flop in the sdr_clk domain being stitched directly to a rising-edge triggered scan flop in the pclk domain.

9

Scan Design Handoff

Learning Objectives

After completing this lab, you should be able to:

- Name the 3 types of DFTC output files downstream tools require specifically related to scan testing.
- Write out the final test protocol and a scandef file.

Lab Duration:
30 minutes

Overview

Unmapped Flow:

1. Read the RTL design into DFT.
2. Create and save a test protocol; verify that the design and the test protocol are compatible.
3. Compile and save the gate-level design; exit.

Mapped Flow:

4. Read the gate-level design and test protocol into DFTC.
5. Specify scan constraints and preview the scan architecture that will result from applying them.
6. Insert the scan chain.
7. Hand off the design and related files for use by downstream tools; exit.

Answers & Solutions

This lab guide contains answers and solutions to all questions. If you need some help with answering a question, check the back portion of this lab for help.

File Locations

All files for this lab except for designs and libraries are located in the directory **`lab9_export`**.

Directory Structure

`Lab9_export`	Current working directory
`analyzed`	Intermediate acs_read_hdl files
`logs`	Session log files
`unmapped`	Unmapped protocol
`mapped`	Gate-level netlist
`mapped_scan`	Gate-level scan design netlist
`reports`	DFTC reports
`tmax`	Files for downstream tools
`scripts`	Constraint and run scripts
`../rtl/ORCA_init/vhdl`	Design files
`../libs`	Library files

Instructions

Given a mapped design, export from DFTC the files required by downstream tools such as ATPG and P&R.

Task 1. Execute the Mapped Flow and Export Files

1. Change directories to the project directory **`lab9_export`**.

```
UNIX% cd lab9_export
```

2. Invoke DFTC.

```
dc_shell
```

3. Read the gate-level design and test protocol into DFTC.

```
s scripts/4read_gate_and_protocol.tcl
```

4. Specify scan constraints and preview the scan architecture that will result from applying them by using the script provided.

```
s scripts/settings_insert_dft.tcl
preview_dft
```

Question 1. Explain why the 3 configuration registers will not be placed in the scan chains?

.....

.....

5. Insert the scan chain.

```
s scripts/6insert_dft.tcl
```

Question 2. What is the estimated test coverage for the scan design?

.....

6. Complete the script that writes out the design and related files to be used by downstream tools (scripts/7handoff.tcl).

Add to this script:

- The command to save the needed ATPG SPF file (to the `./tmax` directory) that is not currently being saved.
- The command to write out the scan chains to a SCANDEF file (to the `mapped_scan` directory).
- The command to check the SCANDEF

```
unix%  vi scripts/7handoff.tcl
```

7. Execute the handoff script and examine that additional file.

```
s scripts/7handoff.tcl
```

Question 3. Is the custom initialization sequence still there? What has changed about it?

.....

.....

The test protocol for `preview_dft` contains your scan chain in and out signal declarations.

Question 4. What additional information has DFTC added to the ScanStructures section now that the protocol is for a scan inserted design?

.....

Question 5. How many scanchains does the SCANDEF file show, and how many partitions were created (use grep or consult the `check_scan_def` report)?

.....

8. Exit dc_shell

```
exit
```

Task 2. Verify Exported Files in TetraMAX

1. Change directories to the TetraMAX directory **`tmax`**.

```
UNIX% cd tmax
```

2. Inspect the TetraMAX run script. Ensure that the paths for the scan inserted netlist and STIL protocol file referenced in the script match the file written out by DFT Compiler.

```
unix%  vi orca_tmax.tcl
```

3. Run TetraMAX.

```
tmax –tcl orca_tmax.tcl
```

Question 6. What is the Test Coverage reported by TetraMAX?

.....

Question 7. How does the Test Coverage compare to the estimated coverage reported in DFTC? Can you explain the reason for the difference?

.....

Congratulations! You have completed the lab.

Answers / Solutions

Solution scripts are available in the `.solutions` directory.

Task 1. Execute the Mapped Flow and Export Files

Question 1. Explain why the 3 configuration registers will not be placed in the scan chains?

The PRE-DFT violations of “cell is constant 0/1” mean DFTC cannot shift arbitrary logic 0s and 1s through these cells as required for scan shift; they are always 0 or always 1.

Question 2. What is the estimated test coverage for the scan design?

The test coverage is around 92%.

Question 3. Is the custom initialization sequence still there? What has changed about it?

Yes. In the unmapped_flow.spf file, the `test_setup` vectors you added may have specified the same values for the pins for adjacent Vector statements. In the STIL file DFTC wrote for TetraMAX, notice “pclk”=P is only done once, for the first configuration cycle.

Question 4. What additional information has DFTC added to the ScanStructures section now that the protocol is for a scan inserted design?

The ScanStructures section now includes for each chain a ScanLength specification that declares how many scan cells are in each chain.

Question 5. How many scanchains does the SCANDEF file show, and how many partitions were created?

Use “grep PARTITION mapped_scan/ORCA.scandef”.

9 scan chains and 4 partitions.

Task 2. Verify Exported Files in TetraMAX

Question 6. What is the Test Coverage reported by TetraMAX?

Over 96 %

Question 7. How does the Test Coverage compare to the estimated coverage reported in DFTC? Can you explain the reason for the difference?

The difference can be attributed to TetraMAX running fast sequential ATPG (enabled by "`set_atpg -capture_cycles 4`"). The test coverage estimate provided by "`dft_drc -coverage_estimate`" only runs basic scan ATPG (combination ATPG).

10 Improving Scan Insertion Run-Time and Capacity

Learning Objectives

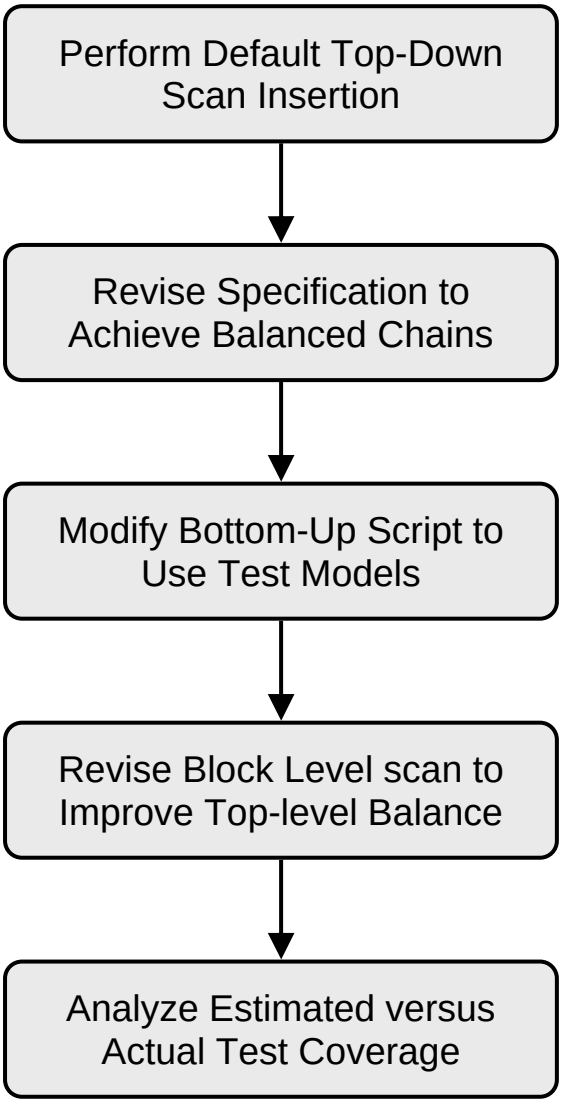
After completing this lab, you should be able to:

- Given existing scan insertion scripts, implement RSS in a top-down scan insertion flow achieving well-balanced scan chains
- Modify a bottom-up scan insertion script for full gate-level designs to use test models with RSS and run it
- Preview top-level chain balance using test models after block level scan insertion and revise the block level scan architecture as needed to improve top-level scan chain balance

Lab Duration: 75 minutes

Overview

Lab 10 Tasks



Answers & Solutions

This lab guide contains answers and solutions to all questions. If you need some help with answering a question, check the back portion of this lab for help.

File Locations

All files for this lab are located in the directory **`lab10_hicap`**.

Directory Structure

`Lab10_hicap`	Current working directory
`dc_ilms`	ILMs with test models for subblocks
`mapped`	ORCA design and its subblocks
`mapped_scan`	Scan-inserted design and subblocks
`reports`	DFT reports
`scripts`	Scan insertion scripts
`test_models`	Test models for subblocks
`tmax`	TetraMAX data
`.solutions/scripts`	Solutions for this lab

Relevant Files

`scripts/`

`bottom_up.tcl`	Bottom-up scan insertion script.
Calls the following two scripts:	
`scan_blocks.tcl`	Inserts scan in to the subblocks
`scan_top.tcl`	Inserts scan in to the top-level
`top_down.tcl`	Top-Down scan insertion script

Instructions

During this lab you will perform a top-down and a bottom-up scan insertion flow on the design ORCA. Some scripts are already provided. You will modify the scripts for the various tasks.

These scripts contain almost all the DFTC commands you would need to use when working on a real-world design back in your office. After completing this lab, you will have practiced the steps needed to improve both the run-time and the capacity of your scan insertion scripts. When you re-use the scripts back at your office, focus on the design-specific items (names of clocks, resets, and so forth).

The lab starts with a top-down script since it is easier to understand and provides reference data (number and length of top-level scan chains, estimated test coverage) that you can later use to compare the results of the bottom-up scan insertion script.

Task 1. Top-Down Scan Insertion using RSS

1. Make sure your current directory is **lab10_hicap**:

```
UNIX% cd lab10_hicap
```

2. Open the file scripts/top_down.tcl in your favorite editor (vi, emacs, etc.).

The top_down.tcl script is used to insert scan in a top-down fashion on the ORCA design.

```
UNIX% vi scripts/top_down.tcl
```

3. Modify this script to use the RSS features you learned about in lecture.

Compared to the original script in the lab directory, your revised script should:

- **a.** Modify the original design as little as possible
- **b.** Run scan insertion faster
- **c.** Consume less memory

If needed, check the ***answers*** section for hints.

Once you are done, save the modified script file.

4. Start Design Compiler from within the **lab10_hicap** directory:

```
UNIX%  dc_shell | tee logs/lab10.log
```

5. Execute the script you just edited:

```
s scripts/top_down.tcl
```

6. Open the report generated by `preview_dft` and analyze it.

Question 1. Are the scan chains well balanced? If not, why?

.....

Note: It is important to balance the scan chains if tester time and the cost of test are to be reduced.

7. Open the **top_down.tcl** file again and **change** it so that you can achieve balanced top-level scan chains.

8. Remove the first pass of scan design files before improving the scan architecture.

```
remove_design -design
```

9. Rerun the top_down.tcl script in DFTC.

Question 2. Are the scan chains more balanced now?

.....

Question 3. What DFT structure did DFTC have to add to achieve these balanced scan chains?

.....

Note: The ORCA design has only 3 test clocks, but your real-world design might have many more test clocks. The more test clocks, the greater the probability for having more of these DFT structures. If minimizing die area is an important design requirement, you may need to monitor their number.

10. Determine the number of lockup latches DFTC inserted. DFTC names the latches using an intuitive naming convention that is good for wildcard searches.

```
get_cells LOCKUP* -hier
```

Question 4. How many lockup latches were added?

.....

Note: If your design has many lockup latches counting them yourself can become tedious. Have DFTC count them using:

```
sizeof_collection [get_cells LOCKUP* -hier]
```

Question 5. Where in the ORCA hierarchy were they placed?

.....

Note: If the ORCA design will be placed and routed using a top-down methodology, the placement of the lockup latches is probably not important. If the ORCA design will be placed and routed in a bottom-up fashion, however, you may have preferred locations for them. For example, you may want to avoid any lockup latches at the top-level.

11. Obtain an estimate of the design's test coverage in DFTC.

Question 6. What is the estimated test coverage for ORCA?

.....

12. **Exit** Design Compiler.

Task 2. Bottom-Up Scan Insertion using Test Models

Two scripts are used for bottom-up scan insertion, scan_blocks.tcl and scan_top.tcl. These scripts currently run using full gate-level netlists.

In the subsequent steps you will first run **scan_blocks.tcl** to insert scan into the blocks of the ORCA design generating test models for the blocks. Later you will run the **scan_top.tcl** to complete the top-level scan chain stitching in ORCA using the test models created for the blocks instead of the full gate-level netlists.

The ORCA top-level design contains all the I/O pad instantiations, a CLOCK_GEN block that contains the PLLs and clock multipliers, and an ORCA_TOP block that serves as the core of the design and contains all the blocks for which you will perform scan insertion.

The foreach loop in the scan_blocks.tcl script iterates through a TCL list of the blocks within ORCA_TOP. The loop first reads in the appropriate .ddc file from the correct directory and then proceeds with block-level scan insertion. The results are saved at the end of the loop and all design files are removed before proceeding to the next block.

1. Locate the foreach loop within the scan_blocks.tcl script and modify it to save the test model for each block in a file separate from the DDC file in the test_models directory (put these commands in the script immediately before the command that removes each block).

2. Modify the scan_top.tcl to read in the test models only from the proper directory (these test models were saved by the scan_blocks.tcl script).

Read the test models first, and then read in the block that contains them, ORCA_TOP. Make sure the **RESET_BLOCK** is **not** a test model.

```
. . .
Your commands go here
. . .

read_ddc mapped/ORCA.ddc
current_design ORCA
link
. . .
```

3. Examine the scan_blocks.tcl script and answer the following questions.

Question 7. Why specify –`clock_mixing no_mix` for block level scan insertion?

.....

.....

Question 8. What is the result of `–insert_terminal_lockup` for block level scan insertion?

.....

.....

Question 9. If lockup latches will be inserted at the end of all block level scan chains, do you need to insert any lockup latches between block level chains at the top-level?

.....

4. Run the bottom_up.tcl script, which first runs scan_blocks.tcl on the blocks within ORCA_TOP and then runs scan_top.tcl on the ORCA design.

```
UNIX% dc_shell –f scripts/bottom_up.tcl | tee
                                          bottom_up.log
```

5. Examine the top-level `preview_dft` report for ORCA.

```
vi reports/ORCA_preview_dft.rpt
```

Question 10. Are the scan chains balanced? If not, why?

.....

6. Use the approach outlined in lecture to revise the block level scan architectures so the top-level scan chains are better balanced (stop when all scan chain lengths are within 20 scan cells of each other).

Question 11. Which command and setting produced this better-balanced top-level scan architecture?

.....

Question 12. In the context of a design that contains test models, why can you NOT use the same method for counting the number of lockup latches in the design as used in Task 1?

.....

.....

Note: Since the resulting Verilog gate-level netlists of both the block level scan insertion and the top-level scan stitching are saved to the same UNIX directory, you can use UNIX commands to count the lockup latches.

7. Determine the number of LOCKUP latches inserted throughout ORCA.

```
UNIX%  fgrep LOCKUP tmax/*.v | wc -l
```

Note: `fgrep` searches for fixed string matches and `wc -l` outputs the number of matched lines. In this case, that number is equal to the number of lockup latches.

Question 13. How many lock-up latches were inserted?

.....

Question 14. Where in the ORCA hierarchy were they placed (*Hint*: do the `fgrep` without piping it to `wc`)?

.....

.....

.....

Note: If the ORCA design will be placed and routed using a top-down methodology, the placement of the lockup latches is probably not important. If the ORCA design will be placed and routed in a bottom-up fashion, however, you may have preferred locations for them. For example you may want to avoid any lockup latches at the top-level.

Question 15. How does this compare to the number of lock-up latches inserted in Task 1 by the top-down script?

.....

8. Examine the estimated coverage report for ORCA and answer the following questions.

```
UNIX%  cat reports/ORCA_coverage.rpt
```

Question 16. How does this coverage compare to the estimated coverage obtained when inserting scan top-down into ORCA?

.....

.....

Question 17. Why is the estimated coverage obtained using test models not meaningful?

.....

.....

Note: In order to obtain a more meaningful estimate for the test coverage, you must read the entire design into TetraMAX.

9. Run a script to obtain the actual test coverage in TetraMAX and then answer the following questions.

```
UNIX%  cd tmax
UNIX%  tmax –shell run.tmax
```

Question 18. Why is the total number of faults in TetraMAX much greater than the number of faults in DFTC?

.....

.....

Question 19. What test coverage does TetraMAX obtain for ORCA?

.....

10. Exit Design Compiler.

Task 3. Optional: Bottom-up Scan Insertion using ILMs with Test Models Attached

1. Repeat Task 2 but instead use ILMs with test models attached.

- Modify scripts/ilm_scan_blocks.tcl to generate and save the ILMs.
- Modify scripts/ilm_scan_top.tcl to read in the ILMs.

If you get stuck, check the .solutions directory.

Congratulations! You have completed the lab.

Answers / Solutions

Task 1. Top-Down Scan Insertion using RSS

3. Modify this script to use the RSS features you learned about in lecture. Compared to the original script in the lab directory, your revised script should:

a. Modify the original design as little as possible

b. Run scan insertion faster

c. Consume less memory

```
set_dft_insertion_configuration  \
     –synthesis none  \
     –preserve_design_name true
```

Question 1. Are the scan chains well balanced? If not, why?

No. The scan chains are very unbalanced because clock mixing is not enabled.

7. Open the top_down.tcl file again and change it so that you can achieve balanced top-level scan chains.

```
set_scan_configuration –clock_mixing mix_clocks
```

Question 2. Are the scan chains more balanced now?

Yes. All 6 chains are either 488 or 487 elements long.

Question 3. What DFT structure did DFTC have to add to achieve these balanced scan chains?

Lockup latches.

Question 4. How many lockup latches were added?

2

Question 5. Where in the ORCA hierarchy were they placed?

```
dc_shell-xg-t> get_cells LOCKUP* -hier
{I_ORCA_TOP/I_RESET_BLOCK/LOCKUP I_ORCA_TOP/I_SDRAM_WRITE_FIFO/LOCKUP}
```

Question 6. What is the estimated test coverage for ORCA?

97.67%

Task 2. Bottom-Up Scan Insertion using Test Models

1. Locate the foreach loop within the scan_blocks.tcl script and modify it to save the test models only for each block:

```
foreach design $designs {
    . . .
   # Save the Test Model HERE
   write_test_model -format ddc \
      -output test_models/${design}.ddc
    . . .
}
```

2. Modify the scan_top.tcl to read in the test models only from the proper directory (these test models were saved by the scan_blocks.tcl script.)

Read the test models first, and then read in the block that contains them, ORCA_TOP.

```
read_ddc [glob test_models/*.ddc]
use_test_model -true [get_designs *]
remove_design RESET_BLOCK
read_ddc mapped_scan/RESET_BLOCK.ddc
use_test_model -false RESET_BLOCK

read_ddc mapped/ORCA_TOP.ddc
change_names –rule verilog
read_ddc mapped/ORCA.ddc
current_design ORCA
link
. . .
```

Question 7. Why specify `–clock_mixing no_mix` for block level scan insertion?

Mixing clocks in block level scan chains would restrict top-level scan chain balancing. It is therefore better to postpone scan chain clock mixing until you perform the final top-level scan stitching.

Question 8. For block-level scan insertion, what is the result of using `–insert_terminal_lockup`?

Each block level scan chain ends with a lockup latch, eliminating the need for lockup latches at the top-level.

Question 9. If lockup latches will be inserted at the end of all block-level scan chains, do you need to insert any lockup latches between block-level chains at the top-level?

No, doing so would create back-to-back lock-up latches in the scan chains.

Question 10. Are the scan chains balanced? If not, why?

No. Some block level scan chains are much longer than others. This affects the balance of the top-level scan chains; they are very unbalanced.

Question 11. Which command and setting produced this better-balanced top-level scan architecture?

In scan_blocks.tcl:

```
set_scan_configuration –max_length 100
```

Question 12. In the context of a design that contains test models, why can you NOT use the same method for counting the number of lockup latches in the design as used in Task 1?

You cannot count the latches within a test model because the test model is an empty design in DFTC.

Question 13. How many lock-up latches were inserted?

34

The number of lock-up latches seen will be dependent on the setting for –max_length. The number provided is for a setting of “-max_length 100”.

Question 14. Where in the ORCA hierarchy were they placed (*Hint*: do the fgrep without piping it to wc)?

```
unix% fgrep LOCKUP tmax/*.v | awk '{printf ("%s %s %s\n", $1, $2, $3)}'

      BLENDER_gates.v:  lanlq1 LOCKUP
      BLENDER_gates.v:  lanlq1 LOCKUP1
      BLENDER_gates.v:  lanlq1
      BLENDER_gates.v:  lanlq1
      CONTEXT_MEM_gates.v:  lanlq1 LOCKUP
      ORCA_scan.v:  lanlq1 LOCKUP
      ORCA_scan.v:  lanlq1 LOCKUP1
      ORCA_scan.v:  lanlq1 LOCKUP2
      ORCA_scan.v:  lanlq1 LOCKUP3
      PARSER_gates.v:  lanlq1 LOCKUP
      PARSER_gates.v:  lanlq1 LOCKUP1
      PCI_CORE_gates.v:  lanlq1 LOCKUP
      PCI_CORE_gates.v:  lanlq1 LOCKUP1
      PCI_CORE_gates.v:  lanlq1 LOCKUP2
      PCI_CORE_gates.v:  lanlq1 LOCKUP3
      PCI_CORE_gates.v:  lanlq1 LOCKUP4
      PCI_CORE_gates.v:  lanlq1 LOCKUP5
      PCI_CORE_gates.v:  lanlq1 LOCKUP6
      PCI_CORE_gates.v:  lanlq1 LOCKUP7
      PCI_CORE_gates.v:  lanlq1 LOCKUP8
      PCI_CORE_gates.v:  lanlq1 LOCKUP9
      PCI_CORE_gates.v:  lanlq1 LOCKUP10
      PCI_RFIFO_gates.v:  lanlq1 LOCKUP
      PCI_RFIFO_gates.v:  lanlq1 LOCKUP1
      PCI_WFIFO_gates.v:  lanlq1 LOCKUP
      PCI_WFIFO_gates.v:  lanlq1 LOCKUP1
      RESET_BLOCK_gates.v:  lanlq1 LOCKUP3
      RISC_CORE_gates.v:  lanlq1 LOCKUP
      RISC_CORE_gates.v:  lanlq1 LOCKUP1
      RISC_CORE_gates.v:  lanlq1 LOCKUP2
      SDRAM_RFIFO_gates.v:  lanlq1 LOCKUP
      SDRAM_RFIFO_gates.v:  lanlq1 LOCKUP1
      SDRAM_WFIFO_gates.v:  lanlq1 LOCKUP
      SDRAM_WFIFO_gates.v:  lanlq1 LOCKUP1
```

Question 15. How does this compare to the number of lock-up latches inserted in Task 1 by the top-down script?

There are many more lockup latches, one for each block level chain.

Question 16. How does this coverage compare to the estimate coverage obtained when inserting scan top-down into ORCA?

The ~45% estimated test coverage is much lower than that obtained for the full gate-level top-down scan inserted design.

Question 17. Why is the estimated coverage obtained using test models not meaningful?

A test model in DFTC is an empty design. When obtaining the estimated test coverage for a design in DFTC that uses test models, these empty designs will also be passed to TetraMAX.

Question 18. Why is the total number of faults in TetraMAX much greater than the number of faults in DFTC?

TetraMAX has the full gate-level netlist for the entire design hierarchy. DFTC had the gate-level netlist for ORCA_TOP and the designs above it (CLOCK_GEN and ORCA); but all the blocks within ORCA_TOP except for RESET_BLOCK were test models (empty designs) in DFTC.

Question 19. What test coverage does TetraMAX obtain for ORCA?

98.50%

12 DFT MAX flow

Learning Objectives

After completing this lab, you should be able to:

- Insert Adaptive Scan using top down flow
- Understand needed components for Adaptive Scan
- Run TetraMAX flow on Adaptive Scan desgin

Lab Duration:
45 minutes

Overview

During this lab you will:

1. Perform the Mapped DFT flow for a test-ready design through scan insertion while enabling Adaptive Scan insertion.
2. Identify needed files to export to TetraMAX.

Mapped Flow:

1. Read the gate-level design and test protocol into DFTC.
2. Specify scan constraints and preview the scan architecture that will result from applying them.
3. Add needed option to enable Adaptive Scan Flow.
4. Insert the scan chains and Adaptive Scan logic.
5. Understand post scan insertion dft_drc checks.
6. Hand off the design and related files for use by downstream tools; exit.

File Locations

All files for this lab except for designs and libraries are located in the directory **`lab12_dftmax`**.

Directory Structure

`lab12_dftmax`	Current working directory
`logs`	Session log files
`mapped`	Gate-level netlist
`mapped_scan`	Gate-level scan design netlist
`reports`	DFTC reports
`tmax`	Files for downstream tools
`scripts`	Constraint and run scripts
`../ref/rtl/ORCA/vhdl`	Design files
`../ref/db`	Library files
`.solutions`	Solution scripts

Instructions

Your goal is to complete and run the four steps of the Mapped Flow portion of the DFT Insertion Flow. The scripts you will modify are for previewing the scan architecture, inserting scan chains and finally, handing off the needed files.

The four needed scripts:

```
Mapped Flow
  4read_gates_and_protocol.tcl
  5preview_dft.tcl
  6insert_dft.tcl
  7handoff.tcl
```

Task 1. Read gate-level Design and its Test Protocol

1. Invoke DFTC.

```
cd lab12_dftmax
dc_shell | tee logs/mylab12.log
```

2. Examine and execute the script that performs step 4 of the DFT flow: reading in the gate-level design and creating the protocol.

```
exec cat scripts/4read_gate_and_protocol.tcl


s scripts/4read_gate_and_protocol.tcl
```

Task 2. Specify and Preview Scan Architectures

1. Investigate side effects of specifying long chains.

```
set_scan_configuration \
    -clock_mixing mix_clocks -chain_count 2
preview_dft -show scan_summary
```

Question 1. How many scan chains will be inserted, and how will this affect the number of tester cycles on the tester?

.....

Question 2. What is one way to reduce Test Data Volume and Test Time?

.....

2. Change the **scripts/settings_insert_dft.tcl** script to enable compression with a compression ratio of 5.

```
set_dft_configuration -scan_compression enable
set_scan_compression_configuration –minimum_compression 5
set_scan_configuration –chain_count 5
```

Note: Since the design is so small for lab purposes, setting a compression ratio of 5 is enough to see the benefits.

Question 3. Do you need to specify the compression ratio?

.....

3. Do another preview_dft run and compare what was architected now.

```
preview_dft -show scan_summary
```

Note: For the class, a Tcl script is provided to display content in a separate window. Try "`v preview_dft -show scan_summary`"

Question 4. What do you see now in the preview report?

.....

4. Investigate the use of the Test_mode pin.

Question 5. Why was there a test_modea pin added to the design?

.....

Question 6. Can you share the existing test_mode pin that might be selecting Test Clocks?

.....

Question 7. Which pins are used for scan in, scan out and scan enable?

.....

5. Edit the file **scripts/settings_insert_dft.tcl** again and specify your own top level Test Mode pin for compression. Then rerun the script.

```
create_port –direction in TM_COMP
set_dft_signal –view spec –type TestMode –port TM_COMP
```

6. Since a new port was added you need to recreate the test protocol file and rerun dft_drc.

```
remove_test_protocol
create_test_protocol –capture_procedure multi_clock
dft_drc
preview_dft –show scan_summary
```

Question 8. Which port is now being used to select between modes?

.....

7. Now insert_dft to insert the chains and compression logic.

```
insert_dft
```

Question 9. What new messages do you see with `insert_dft`?

.....

Task 3. Hand-off files for ATPG for two modes

Question 10. How many STIL Test Protocol files do you need to write out with an Adaptive Scan flow?

.....

Question 11. How many netlists are needed?

.....

1. Check drc rule violations on Internal_scan mode:

```
current_test_mode Internal_scan
dft_drc
```

2. Check drc rule violations for ScanCompression_mode:

```
current_test_mode ScanCompression_mode
dft_drc
```

Question 12. What are the differences that you see?

.....

Note: Internal_scan mode is also known as reconfigured scan mode.

3. Edit the file **scripts/7handoff.tcl** and add the command to write out the ScanCompression_mode protocol.

The ScanCompression_mode protocol file should be named "scancompress.spf" and written to the "tmax" directory.

4. Run the handoff script:

```
s scripts/7handoff.tcl
```

Question 13. Look at both test protocol files, do you see a difference?

.....

Task 4. Run ATPG in Compression mode

1. Run the TetraMAX command file ./tmax/compression.cmd to run ATPG in Adaptive Scan mode:

```
cd tmax
tmax  compression.cmd &
```

Question 14. What is the test coverage reported by TetraMAX ?

.....

Congratulations! This completes the DFTMAX lab.

Answers / Solution

For help with the scripts/commands, see the **.solutions** directory.

Question 1. How many scan chains will be inserted, and how will this affect the number of tester cycles on the tester?

2 scan chains. This will make the scan chains longer, which will affect the number of tester cycles needed per pattern hence increasing the Test Data Volume and Test Time.

Question 2. What is one way to reduce Test Data Volume?

Use Adaptive Scan technology to insert compression.

Question 3. Do you need to specify the compression ratio?

Yes. The compression ratio defaults to 10. So you only need to specify it if you are reducing or increasing this number.

Question 4. What do you see now in the preview report?

You see two modes specified. One for Internal_scan with 5 chains and one for ScanCompresion_mode with 30 much shorter chains. You also see a new Test Mode Controller Information section.

Question 5. Why was there a test_modea pin added to the design?

A control signal is need to select between the two modes created by Adaptive Scan, Internal_scan and ScanCompression_mode.

Question 6. Can you share the existing test_mode pin that might be selecting Test Clocks?

No. The test_mode port was specified as a Constant type signal. It could be a signal used to fixed resets or clocks. We need a dedicated Test Mode pin for Adaptive Scan.

Question 7. Which pins are used for scan in, scan out and scan enable?

Same pins as specified for Normal scan

ScanDataIn Ports:

pad[0], pad[1], pad[2], pad[3], pad[4]

ScanDataOut Ports:

sd_A[0], sd_A[1], sd_A[2], sd_A[3], sd_A[4]

ScanEnable Port:
scan_en

Question 8. Which port is now being used to select between modes?

TM_COMP.

Question 9. What new messages do you see with insert_dft?

```
Architecting Scan Compression structures
Architecting Scan Chains
Architecting Load Compressor (version 1.3)
Number of inputs/chains/internal modes = 5/30/4
Architecting Load Decompressor (version 1.3)
Number of outputs/chains = 5/30
Combinational initialization successfully completed.
```

Question 10. How many STIL Test Protocol files do you need to write out with an Adaptive Scan flow?

Two Test Protocol files, one for normal scan, and one for adaptive scan.

Question 11. How many netlists are needed?

Only one.

Question 12. What are the differences that you see?

There are new scan compression violations:

```
517 SCAN COMPRESSION VIOLATIONS
150 X on chain affects observe ability of other chains violations (R11)
367 X source gate is propagatable to other scancells violations (R14)
```

Question 13. Look at both test protocol files, do you see a difference?

Visually inspect the files to see added sections for Adaptive Scan.

Question 14. What is the test coverage reported by TetraMAX?

99.56%