VLSI Test Technology and Reliability

ET4076-11

Lab Course
2011-2012

Computer Engineering Lab
Delft University of Technology
The Netherlands

Ir. J. Verbree
Ir. M. Taouil
Dr. Ir. S. Hamdioui

1. Introduction

This lab course introduces you (students or perhaps future test engineers) to some practical aspects of testing integrated circuits and systems. Every manufactured chip has to be (extensively) tested to make sure that no faulty chip will be shipped to customers; this is because chips may suffer from design errors and/or defects due to manufacturing process.

The manufacturing process for computer chips is not perfect. No matter how perfectly clean the clean room is, there will always be variation between production batches and even neighboring chips. This is due to the fact that the manufacturing process is very complex and highly sensitive to environmental conditions. Furthermore the base materials from which the chip is produced are slightly polluted and even the silicon wafers themselves will have some defects. Furthermore the transistors are continuously scaled smaller and become increasingly sensitive for these defects; the importance of individual chip testing is paramount.

Design-for-Test or DfT is the process of modifying and adding additional hardware to a chip's design in order to aid the testing process. DfT makes it possible to perform individual chip testing within a reasonable amount of time per unit while achieving high test coverage. Furthermore, DfT can help diagnose which components are failing in order to target those areas in a subsequent redesign or manufacturing of the chip.

DfT is integrated in Electronic Design Automation (EDA) software. There are three major players in this field: Cadence, Synopsys and Mentor Graphics. EDA software helps creating a chip in various facets of the design and production process, from the basic requirements through designing, synthesis, floor planning, optimization, power- , area-, speed-analysis and testing.

For this lab, we will be using Cadence software; in particular the tools RTL Compiler, Test Encounter and NC-Sim. All three of these tools are part of the Cadence Encounter toolset, specifically targeting digital integrated circuits.The choice for Cadence software as opposed to other software providers stems not only from advanced state of the software but also of the high penetration of Cadence software in the field chip design, and therefore the likelihood for you as future chip designers using of Cadence Software in your field of work.
Additionally, it is highly beneficial for the University and therefore you as student that through the Euro Practice license this software is available. Note that on its own a full license for EDA software can run into hundreds of thousands of Euros per user basis. The Computer Engineering section of TUDelft is also a leading member of the Cadence Academic Network, ensuring that the section get access and support to Cadence software tools.

1.1. Lab goals

The lectures of the course VLSI Test Technology and Reliability provide a solid background in all aspects of IC testing including fault modeling, test pattern generation, design-for-testability, scan design, etc. The course shows that testability is one of the most important requirements which should be considered along with other essential constraints such as performance or cost when designing a circuit. A circuit with poor testability may cause time and/or cost losses during post-fabrication testing and testing for reliability. The latter aspect might be very important because testing of some operation-critical devices is done during the whole lifetime of such a device.

The purpose of the lab is get student familiar with design-for-testability and with using EDA software tools. The learned concepts during the lectures will be used to solve the assignments and thereafter EDA tools will be used to experimentally evaluate the proposed solutions and changes designs to make them better testable. .

This lab course will thus provide students with insides on the fundamentals of DfT software tools and to provide first-hand experience on the usage of these tools. After finishing the lab, students will be:

- Familiar with EDA testing software.
- Able to perform test pattern generation for combinational and sequential circuits.
- Able to modify circuits to make them better testable (e.g., scan chain insertion).
- Able to use JTAG for board-level testing.
- Evaluate the impact of testability on circuit design.

1.2. Lab materials

To perform the activities of the lab properly and efficiently, the following material has to be available:

- This lab course manual.
- VLSI Testability and Reliability course materials (i.e., slides and book).
- Cadence product manuals and reference guides.

In addition, source files needed to perform the lab are zipped together and can be found in the public directory of the user '*said'.*

1.3. Assignments

There are in total three assignments. Each assignment has two parts: one theoretical part that has to be worked out at home and one practical part that has to be performed using Cadence tools.

1. Combinational Circuit Testing: In this assignment, you will lean and practice concepts such as stuck-at-fault models, fault coverage, ATPG for combinational circuits, etc.
2. Sequential Circuit Testing: : In this assignment, the focus will be mainly on scan design, balancing Scan Chains and ATPG for sequential circuits.
3. Board Testing and Boundary Scan: In this assignment, you will learn and practice to add Boundary Scan to a design to aid board testing; i.e., how to test both interconnects and chips on a board are tested.

Table 1 summarizes the required knowledge for each of the assignments together with the targeted objectives.

Table 1: Overview over the assignments

#	Required knowledge	Objectives
1	• Defects, fault models, fault equivalence and fault collapsing (Chapters 1, 2 and 4) • Functional versus structural testing (Chapter 4) • Fault simulation (Chapter 5) • Testability measures and SCOAP (Chapter 6) • ATPG for combinational circuits, D and PODEM algorithms (Chapter 7) • Movement through the file system • Opening files and starting application • Editing files (we recommend to use a graphical interface) • Moving files • Please read the first 2 chapters from a Guide to Linux handbook: http://tldp.org/LDP/intro-linux/intro-linux.pdf	• Define fault lists of combinational circuits and optimize them • Develop test patterns for stuck-at-faults • Use SCOAP for testability measures and analysis • Better understand the theory and have good background for the Lab assignment 1. • Become familiar with the Cadence environment • Learn to use RC to get a visual representation of the design and of the changes that have been made to it • Learn to use ET to perform ATPG • Be able to compare and analyze the obtained results with those from the software, and explain the possible differences.
2	• Defects, fault models, fault equivalence and fault collapsing (Chapters 1, 2 and 4) • Testability measures and SCOAP (Chapter 6) • ATPG for combinational circuits, D and PODEM algorithms (Chapter 7) • Automatic Test Pattern Generation for sequential circuits (Chapter 8) • Scan design (Chapter 14) • Basic knowledge of LINUX and the Cadence tools: RTL compiler (ET), Encounter Test (ET) and NC-sim.	• Insert scan chains in a sequential circuits. • Develop test patterns for scan based sequential circuits. • Put Scan Design into practice • Adapt ATPG to Scan Design • Use scan design as DFT to test sequential circuits
3	• … • .. • .. • .. • .. • . •	• …..

1.4. Assessment

There is no written exam. The assignments should be completed and approved in order to get "pass" for the lab. The mark that you will get in the oral exam of the course VLSI Test Technology and Reliability is valid only if you pass the lab.

2. Simulation Setup environment and access

This section first explains how to access the servers which support the tools to be used. Thereafter, it describes how to set up the simulation environment. Finally, it briefly describes how to use the tools and start them.

It is worth noting that it is very important to read and understand this section in order to be able to setup the simulation environment and perform your assignments in an efficient way and without errors.

2.1. Access to CE servers

It is crucial to get the-to-be used tools up and running on the machine that you will be using. The tools and licenses are very expensive; they are installed and available on only three Computer Engineering (CE) compile servers on the 15th floor of the EWI building. Fortunately, these servers can be accessed by Linux-workstations on the 15th floor for which you are required to have a CE-account. These machines have a synchronized file system with the servers, allowing for local editing and remote execution. Alternatively, the servers can also be reached by any PC regardless of the operating system within the TU Delft IP-range provided that you have a CE-account. You can use a VPN-client to obtain access from home.

Before continuing, some basic understanding of Linux is required. Please, read the first 2 chapters from the following manual: 'a Guide to Linux handbook':
http://tldp.org/LDP/intro-linux/intro-linux.pdf

2.1.1. Access using Linux

In order to make use of this facility, you need an account (i.e., *username*) within CE section. For this, you have to contact the system manager and/or the professor.

Start a console/terminal window in Linux; then type one of the following commands to connect to a server:

$> ssh –X username@ce-comp01.et.tudelft.nl
$> ssh –X username@ce-comp02.et.tudelft.nl

Use your CE-account to gain access to the servers; your general TUDelft NET-id account will not work. In case you have problems accessing the server, please contact the Teaching Assistant. You may start two consoles/terminals at the same time such that you can use one for running EDA tools and one for editing the input files/scripts.

It is worth noting that only very basic editors (such as 'vi') are available on the server. It is therefore preferable to use the second console/terminal to connect to one of the CE Linux desktop where you can use more advanced editors such as 'Kwrite'. You can use one of the following commands to connect to one of the nine CE desktops:

$> ssh –X username@ce-ws001.et.tudelft.nl

$> ssh –X username@ce-ws002.et.tudelft.nl
……
$> ssh –X username@ce-ws008.et.tudelft.nl
$> ssh –X username@ce-ws009.et.tudelft.nl

Alternatively you can edit the scripts on you own machine and transfer the files afterwards to the server via SFTP (same addresses as for SSH, port 22).

2.1.2. Access using Windows

You can alternatively use a non-Linux machine such as Windows. However, in this case specific programs should be used in order to make the access to CE-servers feasible There are two options:

- MobaXterm: this is the preferable program as it does not require any additional configuration. You can download this application form:
  http://mobaxterm.mobatek.net/MobaXterm_v4.0.zip
  additional information about MobaXterm can be found at:
  http://mobaxterm.mobatek.net/download-home-edition.html
- A *combination* of Putty (SSH connection) and Xming which provides Graphic extension for Putty so you can actually see the GUI of the tools. This option needs additional configuration steps. Putty can be downloaded at:
  http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
  and Xming at: http://sourceforge.net/projects/xming
  In case you use this option, make sure to enable R11 forwarding in Putty to support GUI.

2.1.3. Access from home

It is also possible to access CE-servers from your home computer; however, this requires an IP within the TUDelft network. For those outside the TUDelft network, they need to use VPN for the connection. TUDelft already has a support for VPN; this can be found at:

https://intranet.tudelft.nl/live/pagina.jsp?id=fd98130a-0893-4b2b-91d1-f5ba8181ff76&lang=en

2.2. Simulation environment set up

Once you succeed to have access to CE-server, you have to set up your own working directory. We recommend to create a new directory in the default directory “Documents”. First, go to this directory using the following command:

$> cd ~\Documents

Next, create your working directory where the assignments will be placed.

$> mkdir cad_assignments

Next, enter this directory by using the following command:

$> cd cad_assignments

This directory is empty. The following command lists the content of the directory.

$> ls

Nothing should appear as the directory is empty.

A zip-file, which contains all the designs, libraries and scripts you need to start working on the assignments, has been made available to you.
This zip-file is located in the public directory of the user '*said*', that can be found in same home directory as you own account. When you start a file manager of a command prompt the default displayed directory is your own directory (i.e. */home/<user name>*), from there you can travel to "*/home/said/public",* do note that "*/home/said"* is not accessible, you will have to travel to "*/home/said/public"* directly.

Now, we will copy the zip file to your working directory and unzip it. Use the following command **while in your working directory**:

$> cp /home/said/public/Student_package_VLSI_Testing.tar.gz .

It is assumed here that you are in your working directory; otherwise you have to replace the "dot" at the end of the command with the following:

/home/yourname/documents/cad_assignments

Next, the files will be extracted from *Student_package_VLSI_Testing.tar.gz;* again it is assumed here that you are in your working directory. Use the command:

$> tar –xzvf Student package_VLSI_Testing.tar.gz

In order to see all the extracted files, type again the following command:

$> ls

Now, you should see several files that have been extracted from the original compressed file. The contents of working directory should have a structure as depicted in Figure 1.

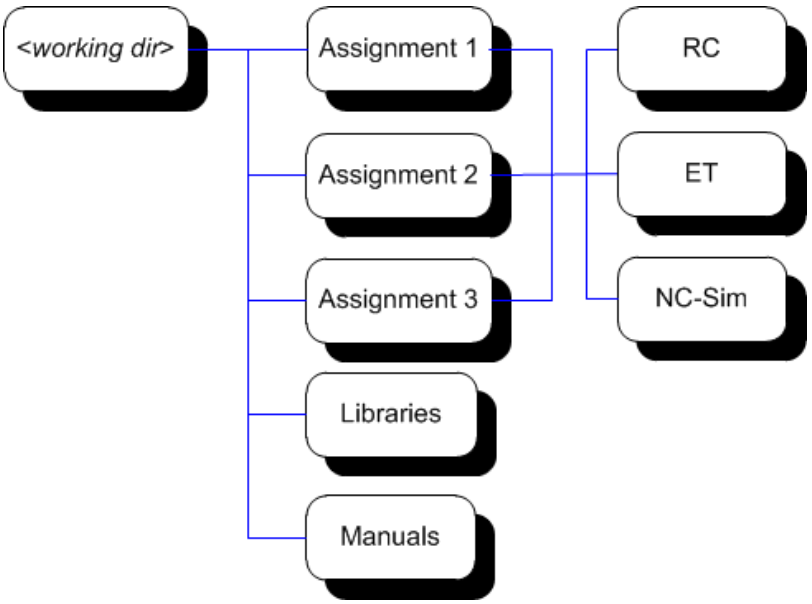

Figure 1: Directory structure, VLSI Test and Reliability lab course

2.3. EDA/ cadence tools

This section describes first the tools to be used and thereafter discusses how to start/run them.

2.3.1. Cadence tool set

In order to be able to change the designs and insert DFT features during this Lab, three software tools will be used. They are part of the Encounter software package, which specifically targets digital circuits. The three tools are briefly explained next. For this lab course you will be focusing on three Cadence software tools.

- **RTL Compiler (RC):** This is a general tool able to read a description of a digital circuit given in a vhdl or verilog format. The tool provide the ability to review, modify and optimize the different component of the design. RTL compiler requires a full specification of all components within the design; hence, they have to mapped to a pre-defined library. This library contains the logical and physical properties of standard cells, such as the one that are being used in the netlists. Our cell library for instance contains many variants of AND-gates, varying not just in the number of inputs, but also size, voltage requirements, delay, energy consumption, heat tolerance, drive-strength and so on. The tool is able to automatically optimize the design and provide the most efficient solutions to meet the design requirements. In this lab course, the tool will be used to view, insert DFT features (such as scan insertion) and compile the designs under consideration. The RTL compiler provides feedback on each performed step; e.g., the impact of added DfT features to the design in terms of area, latency and power consumption. It worth noting that this tool will be primarily used for scan design and JTAG. Once the design is modified to support DFT infrastructure, it can be saved as a verilog netlist and used as input for Encounter Test tool.

- **Encounter Test (ET):** it is an Automated Test Pattern Generation (ATPG) tool. ET used the logical functions of the design component to generate appropriate test patterns. It identifies all possible faults within the design, and provides test patterns for their detection together with a test bench that can be used for fault simulation. The faults are stored in a fault list which may contain static faults (such as stuck-at faults) as well as non-static faults (such as delay faults). Note that the test bench consists of the test patterns, setup requirements (if any) to put the design in the desired state to receive test pattern, and the fault-free expected responses. ET is trying to use the least amount of test vectors to cover as many faults as possible; it therefore maintains lists of all possible fault sites and whether a fault has been covered by a test or not. Ultimately these statistics also yield the so called test-coverage, the percentage of faults covered by a particular set of test set or Testbench.

- **NC-Sim**: it is a tool used to perform circuit/fault simulation. In order for the tool to perform an accurate simulation, it needs the specification of all design components, the design itself and the test bench created by ET. The NC-Sim runs the test bench and compares the produced outputs with the expected values. This software tool provides also visual feedback to what is actually happing inside the DUT (Device Under Test) during the test. NC-Sim is a simulation tool which is similar to Modelsim from Mentor Graphics (with which you may be familiar with).

Figure 2 will be used to explain the overall flow of these tools. Initially, we start a hdl design description (e.g., verilog and vhdl) such as “Design.v” in the figure. Thereafter, the file will be compiled using RC to check the correctness of the input file. Next, DFT infrastructures are inserted in the design resulting in a new file “DFT_Design.v”. ET is then applied to generate test

patters and test bench. The latter is used with "DFT_Design.v" file and required libraries to perform fault injection and circuit simulation using NC-Sim in order to identify the fault coverage.

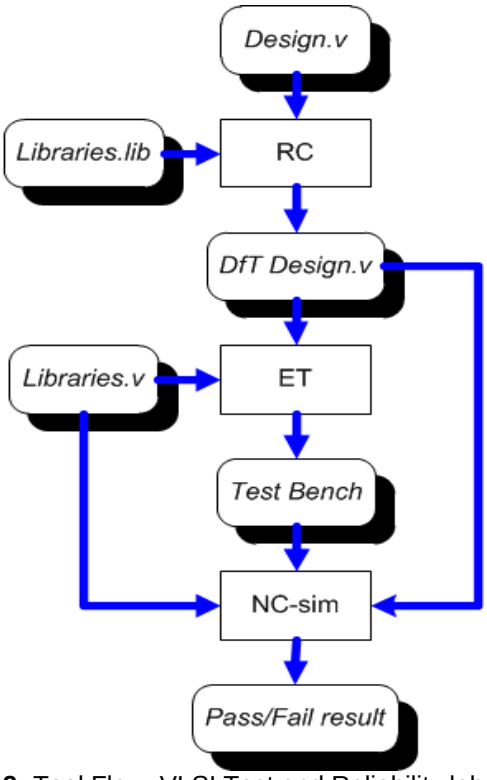

Figure 2: Tool Flow, VLSI Test and Reliability lab course

2.3.2 Staring the simulation environment

Assuming that you already have the access to **appropriate CE-server** (see Section 2.1.1) , you can prepare the environment to run the tools. For this purpose, a source script '*cds-6.1.5-mmsim10.1.sh'* will be used; it is placed in the following location:

/opt/applics/bin/

Note that the script you will be running depends on the shell you are using. For instance, if you are using C shell, then the name of the script will be '*cds-6.1.5-mmsim10.1.csh'.* To find out which kind of shell you are using, use the command:

$> echo $0

In case the output is similar to "bash", then source the following file:

$> source /opt/applics/bin/ cds-6.1.5-mmsim10.1.sh

In case the output is similar to "tcsh" , then source the following file:

$> source /opt/applics/bin/cds-6.1.5-mmsim10.1.csh

Once the Cadence toolset has been sourced, the three software tools that will be used during the lab course are ready for use. You can start the programs using the following commands:

$> rc –gui

Exit the program by typing exit command in the 'rc' command line

rc:/> *exit*

Start now the ET using the following command:

$> et –gui &

Exit the program by closing the ET GUI.

The NC-sim will be started later when we will generate a netlist of a design to play with.

2.3.4. Confidentiality

It is worth noting that for this course the UMC cell library for 90nm technology is made available for students (UMC is a Taiwanese foundry). It should be seen as a confidential material that should not be further distributed and/or copied.

It is therefore strictly forbidden to copy this library for personal use.

3. Assignment 1: Combinational Circuit Testing

Each assignment consists of two parts; one theoretical part to be prepared at home, and practical part that is to be performed in the Cadence environment. You will not be allowed to start the lab assignment unless you have shown that you have successfully completed the homework. A teaching assistant will then approve you homework.

3.1. Homework assignment

Before providing the description of the homework, the required knowledge and the learning goals of the assignment will be given.

3.1.1. Required knowledge

In order to work out the homework, you need the following knowledge:

- Defects, fault models, fault equivalence and fault collapsing (Chapters 1, 2 and 4)
- Functional versus structural testing (Chapter 4)
- Fault simulation (Chapter 5)
- Testability measures and SCOAP (Chapter 6)
- ATPG for combinational circuits, D and PODEM algorithms (Chapter 7)

3.1.2. Learning Goals

After finishing the assignment you will be able to:

- Define fault lists of combinational circuits and optimize them
- Develop test patterns for stuck-at-faults
- Use SCOAP for testability measures and analysis
- Better understand the theory and have good background for the Lab assignment 1.

3.1.3. Assignment description

Given the circuit of Figure 3; it is a combinational circuit consisting of eight logic gates. Answer the following questions.

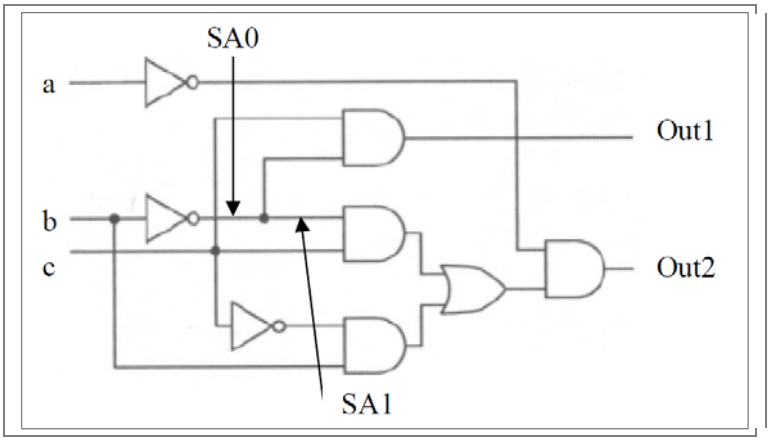

Figure 3: Combinational circuit

1. What is the difference between a defect and a fault model? Give an example.
2. How many test vectors will be needed if functional testing will be used to test the above circuit?
3. What is the number of potential fault sites in the above circuit?
4. Derive the equivalence collapsed set. What is the collapsed ratio? (optional: you may also use dominant fault collapsing)
5. Enumerate the minimal set of single SAF that must be tested according to Checkpoint Theorem. Notice that a single input gate (inverter) has to be treated as continuation of its input line. Use also the concept of equivalence faults to further reduce the number of faults that have to be tested?
6. Compute the combinational SCOAP testability measures (both controllability and observability).
7. Use D algorithm to generate a test pattern for SA0 shown in the circuit? If applicable, how does D algorithm use testability measurements for decision-making?
8. Use D algorithm to generate a test pattern for SA1 shown in the circuit? If applicable, how does D algorithm use testability measurements for decision-making?
9. Use now PODEM algorithm to generate the test pattern for the SA1.
10. (Optional). Given the minimal fault set of question 5, determine (e.g., using the fault simulator concept or ATPG) the required test set for such a fault set.

3.2. Lab assignment

This assignment will introduce you to the Cadence simulation environment and the main tools that will be used for the lab. In addition, the tools will be used to practice ATPG for the circuit of Figure 3. Before describing the lab assignment, the required knowledge and the learning goals will be listed.

3.2.1 Required knowledge:

In addition to the required knowledge described in Section 3.1.1, you need some basic knowledge of Linux such as the following:

- Movement through the file system; moving files
- Opening files and starting applications
- Editing files (we recommend using a graphical interface)
- Read the first *three chapters* from Guide to Linux handbook: http://tldp.org/LDP/intro-linux/intro-linux.pdf

You are encouraged to use a separate directory for each of the assignments.

3.2.2. Learning Goals

- Become familiar with the Cadence environment
- Learn to use RC to get a visual representation of the design and that of changes that have been made to it.
- Learn to use ET to perform ATPG
- Perform circuit simulation using NC-Sim
- Compare and analyze the obtained results and explain any possible differences.

3.2.3. Assignment description

In this assignment, the circuit of Figure 4 will be used; it is the same as the one used for the homework. The purpose of the assignment is to use the Cadence tools in order to generate a set of test patterns for the detection of all static stuck-at-faults.

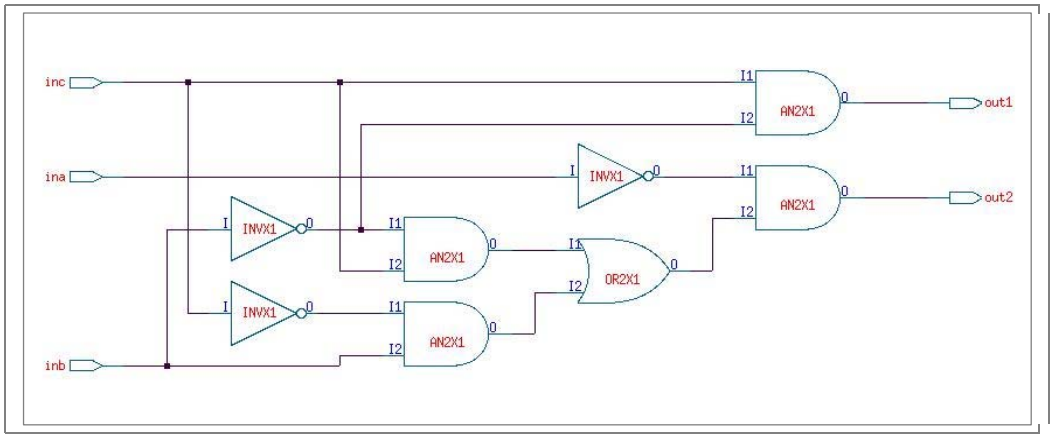

Figure 4: Logic circuit of Lab assignment 1

This lab assignment consists of three main parts:

A. **Reading in the netlist**
B. **Test Pattern Generation**
C. **Circuit Simulation**

First the netlist design of Figure 4 has to be read by RTL Compiler in order to produce the visual schematic of the design/circuit. Thereafter, Encounter Test (i.e. the ATPG tool) will be used to generate the test bench (i..e., test patterns, test setup and the expected results) for stuck-at-faults. Finally, NC-Sim will be used to perform circuit simulation using the generated test bench and compare the obtained results with the expected ones.

A. Reading the netlist:

Make sure that your working environment is "*<work dir>/assignment1/rc/* "

1. Start the RTL compiler. Prior to reading the HDL description of the circuit, RTL compiler requires a technology library that contains the description of the cells used in the circuit. These library cells (such as AND/OR-gates and flip flops) are made in a certain technology, in our case 90nm technology in a UMC facility. The library can be set up (after RC is started) by typing the following command in the rc command prompt:

   &rc:/> *set_attribute library ../../libraries/fsd0a_a_generic_core_ss0p9v125c.lib*

   The "../../" will let you go two levels up in directory hierarchy starting from your working directory, and accessing the directory "*libraries"* which consists of the technology library that we are targeting to set up (*fsd0a_a_generic_core_ss0p9v125c.lib)*

2. Open a second Linux terminal and go to the same directory as above (i.e., rc); open the file ‘ *assignme*nt1.v ‘which contains the netlist of the design; use the following command
   &> kwrite assignment1.v &

   Inspect this Verilog *netlist* file and verify that it indeed describes the circuit as given in Figure 3.

3. The next step consists of reading this file using RC compiler. What are the different commands that can be used to read the HDL file in general? Explain the *difference* between them? Specifically, why is “read_hdl” not suitable in our case? Hint: (see document ‘RC_user_10.1.pdf “ in manual directory, pages 78 to 81 and 100 in particular).

……..
……..
……..
……
. …….
……

   Assume that we have a vhdl description of the circuit of Figure 3 and we want to generate its netlist using RC. Is RC going to produce the same netlist as that in the file *assignment1.v*? Why?

……..
……..
……..
……
. …….
……

   You can close the Verilog file.

4. Use the appropriate command as found in task 3 and execute it. After the execution, the circuit should appear in the GUI. Verify again that this circuit indeed is the same as that of Figure 3.

   You can e.g., zoom in and out by clicking and dragging on the schematic. Note that the way you drag a selected box determines the result of the action. Create a selection box from left to right and then from bottom to top to experiment with dragging actions.

5. Next we will use a TCL script file in order to automatically set up the library, read the input file and generate the schematic.

   Read pages 2 to 6 from the RC User Manual ‘RC_user_10.1.pdf’(**pdf page 22 to 26**) in order to get more insight about how to use script files in the general RC work flow.

   Open the script file *view_assignment1.tcl*. Make sure again you are in ‘rc’ directory of assignment 1. The file has the following content:

   #####Assignment 1
   #####Read Libraries

set_attribute library ../../fsd0a_a_generic_core_ss0p9v125c.lib
#####Read Design
#<put your read_in command here to read the netlist as found in task 3>

The last sentence in the above file means that you have to add a command that will allow the reading of the netlist.

Now the script has to be run. To do that, we restart RC compiler by closing it and opening it again. RC compiler has to be closed by typing the following command at the prompt.

&rc:/> exit

Run the script from the RC prompt, or alternatively through the GUI window:

- From the RC prompt: run the command*: &rc:/> source <file_name>*
- Through the GUI window: Use the "source script" functionality (as depicted in Figure 5); select the appropriate file and hit OK to run the script.

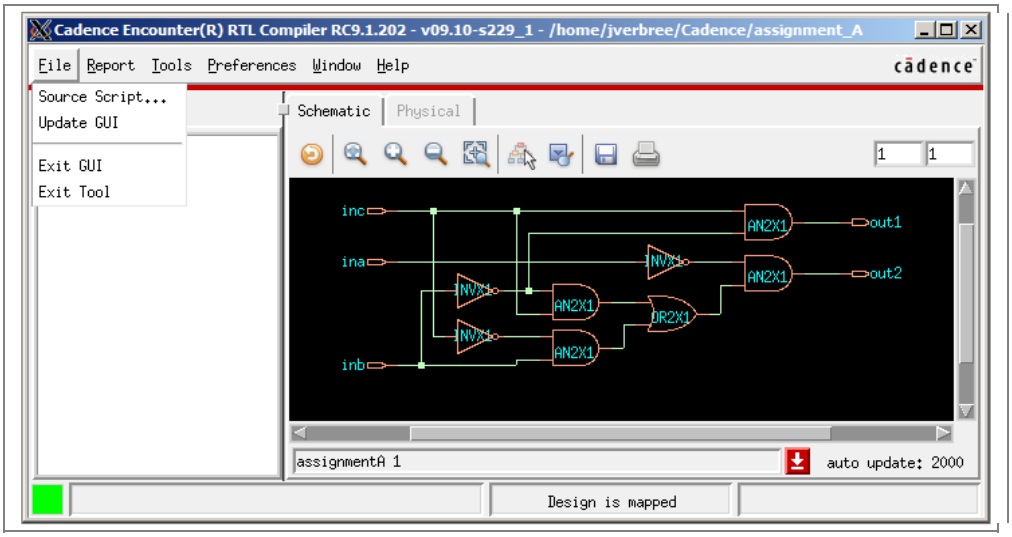

Figure 5: RC Screenshot of assignment 1 design

6. It is worth noting that RC is able to predict the maximal attainable clock speed, power usage and area by using the specified technology library.

   Use RC to estimate the latency, power consumption and area [hint: use GUI] .

Latency:……..
Area:……..
Power:……..

You can now exit RC and start ET.

&rc:/> exit

B. Test pattern generation

It is useful to have the "Encounter Test Modeling User Guide" (filename: "ET_models_10.1.pdf"), and the "Encounter Test Command Reference Guide" (ET_cmdref_10.1.pdf) available. The first document gives a general view about ET and how it can be used, while the second one specifies the commands and their meanings.

1. Start ET (*$> et –gui&)* and create a new project in your working director. We recommend creating the project in the folder <work_dir>/assignment1/et/ (see menu → new)

The next step is to generate ATPG for our circuit. To realizes this, the following steps have to be completed in ET:

- Specify the model, i.e., design file and test library.
- Specify the test mode, e.g., scan, JTAG, etc.
- Specify the fault model, e.g., static faults, dynamic faults, etc.

All these steps can be performed using the "Build Models" menu depicted in Figure 6. (ET_models_10.1.pdf, p23-25).

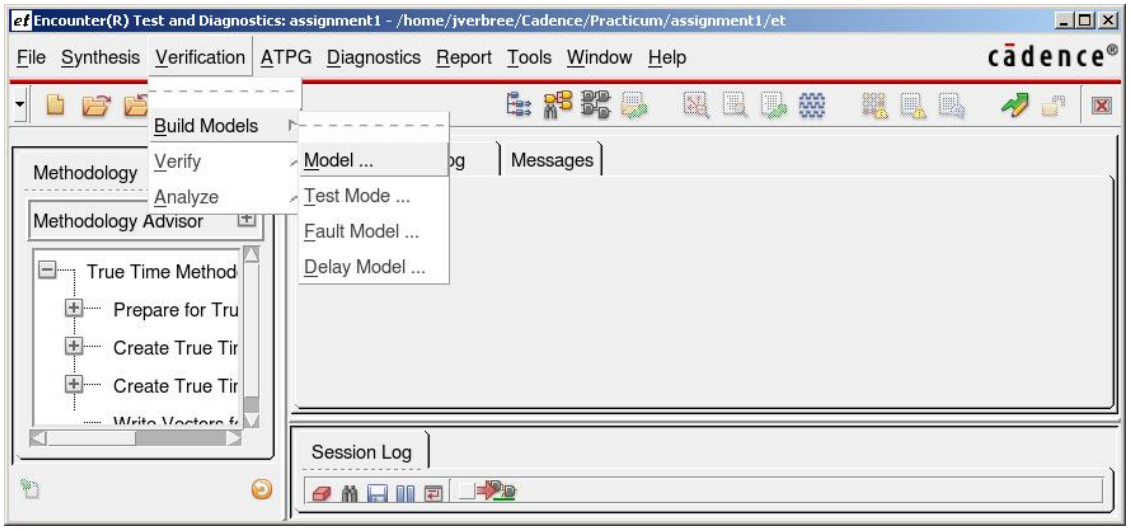

Figure 6: ET Screenshot, Verification → Build Models

Specifying the model

2. Use the "Verification → Build Models → Test Model" menu (see Figure 6) to specify the synthesized netlist and cell-library; browse to select the desired netlist. Before hitting "run", push the "setup" button and select the simplified cell-library "fsd0a_a_generic_core_30_tud.lib.v" in the new windows, hit ". Read also the Model Guide (file: "ET_models_10.1.pdf", pages 23-30).

Specifying the test mode

3. In the "Test Mode" you need to specify the targeted DFT feature that will be used; e.g. Scan Design, compression, JTAG etc. (ET_models_10.1.pdf, pages 77-83).

   In this assignment, you have to use the 'FULLSCAN' test mode. FULLSCAN is normally used for sequential circuit testing. Since ET does not provide a special mode for combinational circuit testing, we will use FULLSCAN.

   Set the 'test mode name' to FULLSCAN and hit run.

We have used the FULLSCAN test mode; however, the ET supports other modes. Name some of the other modes? When they useful?

…… …… …… ……

Specifying the fault model

4. In this step, the set of targeted fault models has to be defined for fault coverage evaluation. For example, static faults (i.e., Stuck-at-faults), dynamic faults, etc. In this assignment, we will target only static faults.

   Select via the menu: "Verification → Build Models → Fault Model", make sure to disable dynamic faults. For now, only static faults are considered.

   Check the session log file to answer the following questions:
   - How many total statistic faults were found?
   - How many collapsed static faults were found?

…… …… …… ……

5. Before generating the test bench / test vectors, ET has to verify the correctness of the chosen DFT features; this can be performed using Verify option under Verification menu:

   Verification → Verify → Test Structures

   Make sure to select the correct Test Mode; Select FULLSCAN.

6. The ATPG process is next; see Figure 7. Test benches can be created for different type of tests such as Static Tests, Delay Tests, Scan Chain, etc (see ATPG→ Create Tests). This assignment targets Static Tests.

   Run ATPG for static Test for *Logic*
   ATPG → Create Tests → Specific Static Faults → logic
   and report about the following:
   - How many patterns are generated?
   - Are there any undetectable faults?
   - How many simulated patterns and how many patterns are needed to realized the maximum fault coverage? Explain the difference.

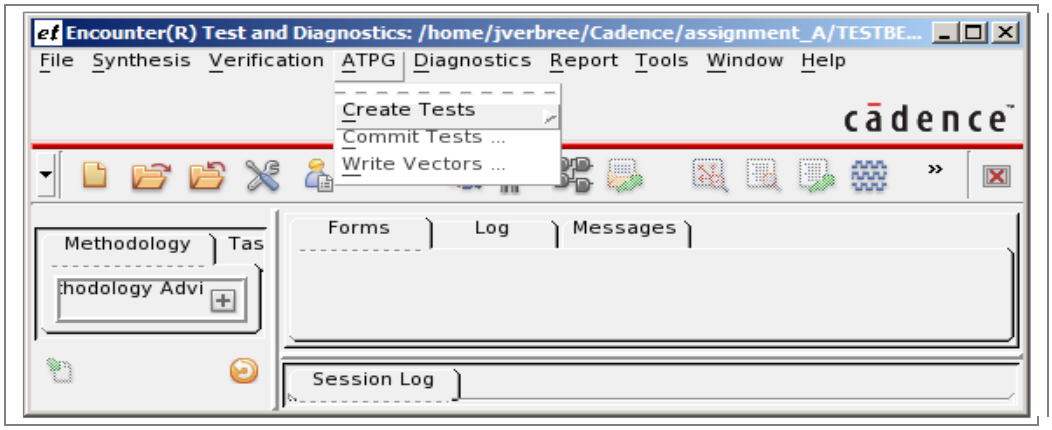

Figure 7: ET screenshot, ATPG

It is useful to play with the tool by e.g.,

- Targeting both static and dynamic faults (Verification → Build Models → Fault Model)
- Running ATPG for delay tests.

7. The ATPG can be run many times with many options till the minimum set of tests/vectors to realize the targeted fault coverage is created. Once this is done, the function "Commit Tests" under ATPG menu has to be used to select the tests to be put in the test bench.

   Add the generated test set for static faults (created for logic in task 6) to the test bench using "Commit Tests".

8. After the test sets to be included in test bench are selected, the test bench has to be created. This is done using the function "Write Vectors" under ATPG menu.

   Create the test bench for the selected test set. While doing this, you will be requested to provide many input options:

   - Vectors to write in test bench: you can choose the committed vectors, but also the uncommitted ones if you already gave them specific names.
   - Provide the output file name and the directory where it will be stored
   - Language: this specifies the language in which the test bench has to be created; examples are STIL, WGL and Verilog. You have to choose Verilog.
   - Scan format: for this assignment, this input is optional as we are not using scan.

9. The test bench (.verilog file) and the setup file (mainsim.v file) are now created. Open <.verilog file> with Kwrite to review its content. How many test vectors are there? What are they? And what are the expected responses of each pattern?

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

10. Given the two stuck-at-faults given in Figure 3, which ones of the test patterns generated in previous step can detect them?

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

Close ET now, by file->exit.

You are now ready for the next step which is circuit simulation using the generated test bench.

C. Circuit simulation

The final step for this assignment is to simulate the test bench using NC-Sim. To make navigation a bit easier, copy the design file and the two test bench files into the NC-sim directory.

We will simulate the design using the test bench.

The test bench contains mainly three items: (a) the setup instructions to put the chip in the desired test state, (b) the test patterns and (c) the expected results. NC-Sim will simulate the circuit using the patterns from the test bench and compare the produced results with the expected ones (which are also part of the test bench).

To start the NC-Sim, you have to read the manual "Simulating_ Encounter_Test_Verilog_Patterns_App_Note", and page 5 in particular; you will then see how the executing command is structured. It is worth noting that the optional and recommended arguments of the command are not needed.

1. Complete the ncverilog command below to start NC-Sim:
   - Check that the provided path to the library file is correct
   - Add the correct file name for "<our_design_netlist>.v ". That is the netlist that will be simulated.
   - Add the correct name for the setup file (mainsim.v file) which was created with ET .
   - Add the correct filename (.verilog file) consisting of test sets.
   - Add an "–access rwc" argument to allow NCsim to access the external pins directly.
   - Add the argument "–timescale 1ns/1ps" to force the time unit/simulation precision.
   - Add the argument "–gui" to activate the GUI environment.

```
ncverilog  -v ../../libraries/fsd0a_a_generic_core_30.lib_tud.v    \
                  <our_design_netlist>.v  \
                  <mainsim_file>.v
                  + TESTFILE1 = <Test_Bench_file>.verilog \
                  ……
```

2. Use your compiled command to start NC-sim; the results is shown in Figure 8

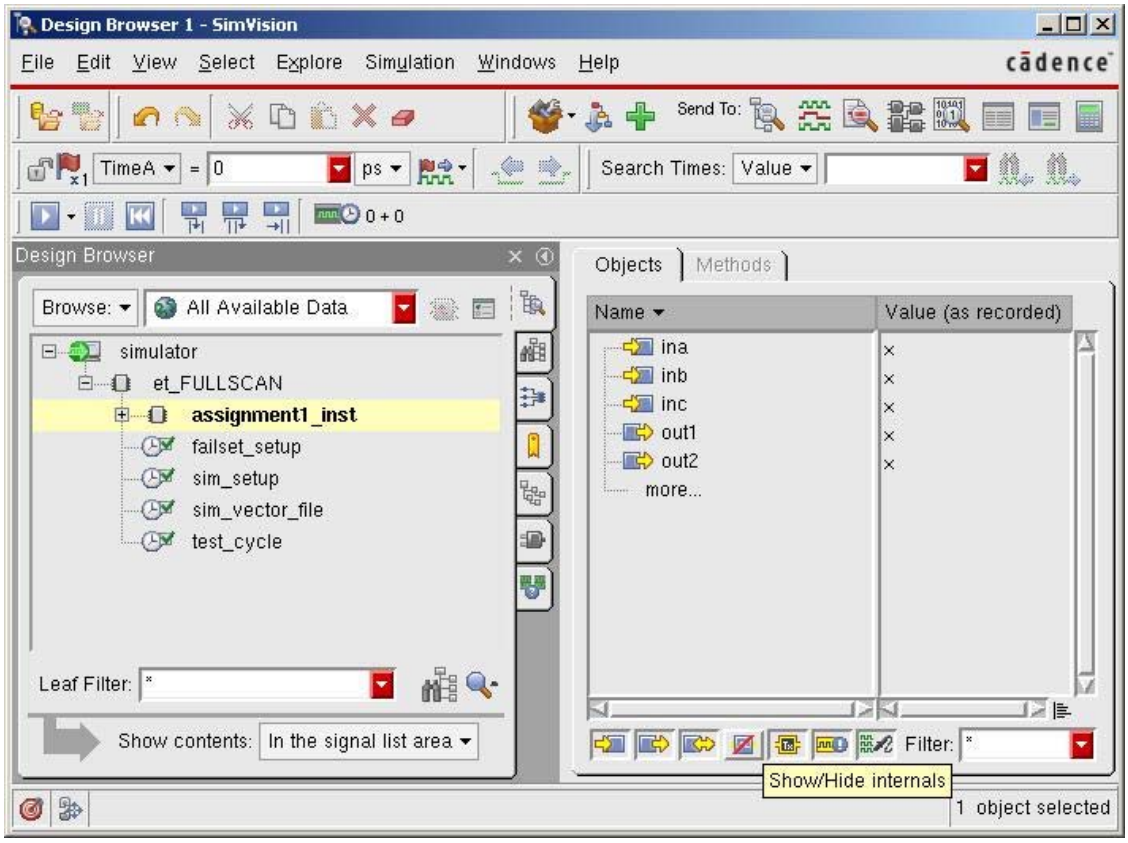

Figure 8: NC-Sim screenshot, signal selection

Browse to select the design under consideration in order to select the inputs and the outputs that will be visualized. Right click on the selected signals, and hit “send to waveforms window”.

Run the simulation: “simulation” menu → run.

The results will be something like that shown in Figure 9. You can zoom out along the x-axis.

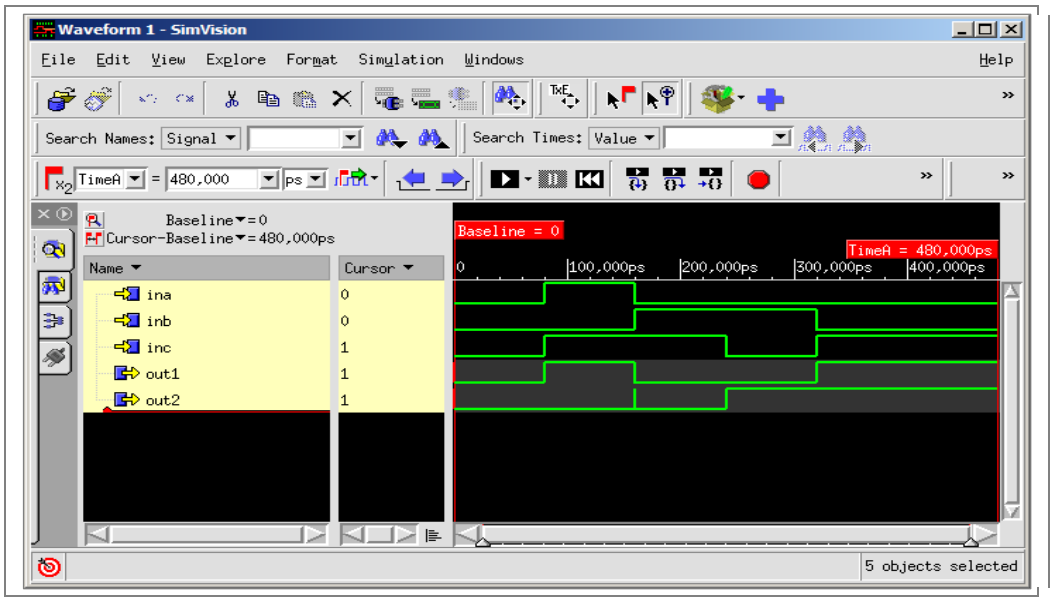

Figure 9: NC-Sim screenshot, waveform after simulation

3. Inspect the waveforms and report about the test patterns and the produced outputs, do they match your expected results?

……
……
:……
:……

4. Inspect the Console window for a log of the simulation run, write down required simulation time and number of good comparing vectors. Can you explain why this number is different from the number of generated vectors?

……
……
……

4. Assignment 2: Sequential Circuit Testing

This assignment deals with sequential circuit testing and scan chain design. The homework will provide you with an opportunity to analytically exercise with the concepts. In the lab assignment you will be using Cadence tools for scan insertion and test vector generation.

4.1. Homework assignment

Before providing the description of the homework, the required knowledge and the learning goals of the assignment will be given.

4.1.1. Required knowledge

In order to work out the homework, you need the following knowledge:

- Defects, fault models, fault equivalence and fault collapsing (Chapters 1, 2 and 4)
- Testability measures and SCOAP (Chapter 6)
- ATPG for combinational circuits, D and PODEM algorithms (Chapter 7)
- Automatic Test Pattern Generation for sequential circuits (Chapter 8)
- Scan design (Chapter 14)

4.1.2. Learning Goals

After finishing the assignment you will be able to:

- Insert scan chains in a sequential circuits.
- Develop test patterns for scan based sequential circuits.

4.1.3. Assignment description

Given the sequential circuit of Figure 10 which consists of two combinational blocks and three D flip-flops. Combinational circuit 1 consists of 150 gates while combinational circuit 2 consists of 200 gates.

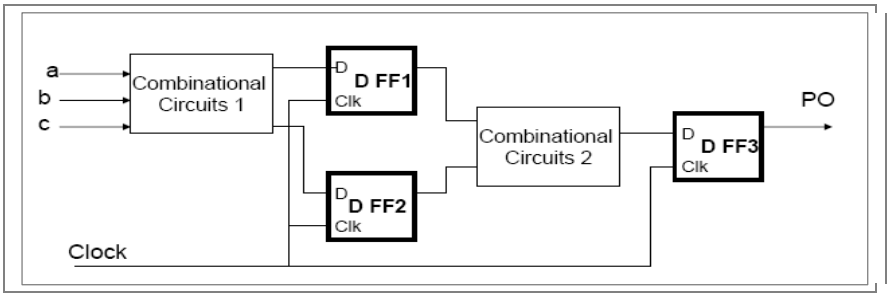

Figure 10: Sequential circuit

1. Convert the circuit above into full scan circuit. Redraw the circuit with scan design and show all the signals and modifications done in the original circuit. Describe clearly the steps you followed to realize the scan design.

2. Explain how you can test the original circuit using the developed scan design.
3. Assume that we want to test SAF in combinational circuit 2. Describe how you can do this.
4. What is the benefit of Scan design? what are the associated cost? How much is the area overhead?

4.2. Lab assignment

This lab assignment Cadence tools will be used to make the sequential circuit of Figure 11 (which is a counter) scan testable. Before describing the assignment further the required knowledge and learning goals are given.

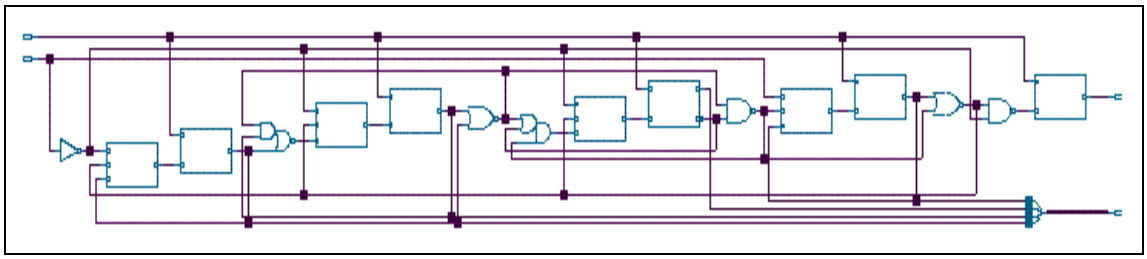

Figure 11: Logic circuit of Assignment2

4.2.1 Required knowledge

In addition to the required knowledge described in Section 4.1.1, you need the following in order perform the assignment in an efficient way:

- Understanding the concept of scan design, how the test is performed and the impact of scan design on the original circuit
- Impact of scan design on area and latency
- Basic knowledge of LINUX and the Cadence tools: RTL compiler (ET), Encounter Test (ET) and NC-sim.

4.2.2. Learning Goals

After finishing the lab you will be able to:

- Put Scan Design into practice
- Adapt ATPG to Scan Design
- Use scan design as DFT to test sequential circuits

4.2.3. Assignment description

The purpose of the assignment is to use Cadence tools to add scan design to the circuit and generate a test bench for the modified circuit utilizing the scan chains to detect the static stuck-at-faults.

This lab assignment consists of three main parts:

A. Scan chain insertion
B. Test Pattern Generation
C. Circuit Simulation

A. Scan chain insertion:

Make sure that your working environment is now "<work dir>/assignment2/rc/"

In this part of the assignment your task is to add DfT hardware to a design. In particular we will replace the existing Flip-Flops (FFs) with scan cells, which offers the same functionality plus the ability to scan values in and out of the circuit.

1. As with the previous lab assignment, you will have to load the design and library into RC. Before doing that, open the Verilog netlist of the counter in kwrite and inspect the file. What is the module/entity name of assignment2? What is the number of inputs and outputs of the circuit?

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

Start 'RC" and read in the Verilog netlist of the counter. Remember to first setup the library prior to reading the HDL file. You can do that by running a command line or creating a script file (e.g. "view_assignment2.tcl") like the one created in the first assignment.

2. What are the area and latency of this circuit?

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

This time RC will not only be used to view the design, but will be also used to make changes to the design and implement Scan Design; this requires several additional steps.
The generalized view is depicted in Figure 12, for us however it requires performing 7 additional steps.

- Add a test pin
- Select Muxed-Scan design
- Check dft_rules
- Replace Flip flops with Scan Cells.
- Define Scan chains
- Put Scan Cells into scan chains
- Write out the new augmented netlist.

Each of the above steps requires a command as will be shown next. You have to run each of the commands in the RC prompt. Once they work correctly, you can include them in a script (e.g., scan_insert_assignment2.tcl script); use the script created in assignment 1 as a template to start with. Make sure that the script file is stored in "<work_dir>/assignment2/rc" folder.

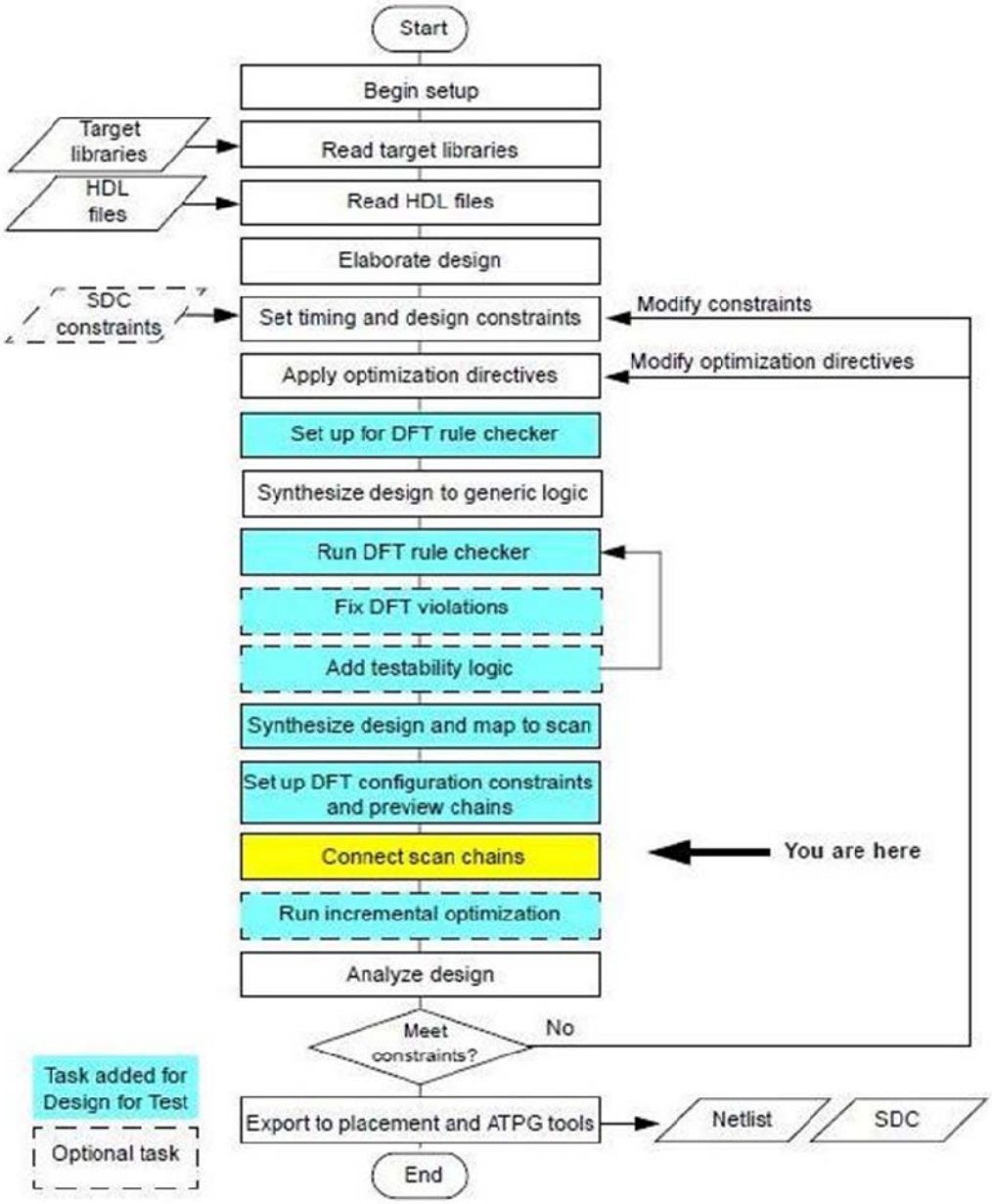

Figure 12: Top-down test synthesis flow

3. **Add a test pin**. A new test pin must be added; this pin is used to switch between the functional and test modes (for e.g. scan-in and scan-out). The test pin is called scan_enable (SE). In order to add this pin, you can use the 'define_dft shift_enable" command (see RC_cmdref_10.1.pdf P647). What is the full define_dft shift_enable command you have to use?

define_dft shift_enable.....
.....
….....

4. **Select Muxed-Scan design**. This command selects the MUX to be used with scan design. Each FF will get then a MUX at its input; the MUX will have two inputs: the original FF functional input and the Scan-in test input; the MUX will be controlled by the scan-enable signal (SE). This is known to RC as the 'muxed_scan' style of scan design.

   What is the command to be used in order to set up this style? It is a "set-attribute" command, see the RC_attref_10.1.pdf manual, search for "muxed_scan".

set-attribute.....
.....
.....

5. Next, **dft check rules** will be applied on the circuit in order to check whether the FF can be converted into SFF.

   The command we are going to use is: 'check_dft_rules' this command and its effects are explained in the RC_cmdref.10.1.pdf p553 manual.

check_dft_rules.....

6. Now the **FFs** can be **replaced** with SFFs. To maximize the fault coverage, all FFs should be converted. (see RC_comref.pdf p739)

   What is the command we are going to use?

.....
.....

7. Next the **Scan chains** can be **defined**. (HINT: See p635, RC_cmdref_10.1.pdf),

   Define the name of the chain, the sdi- and sdo-ports, and create the ports as non-shared ports.
   Name: chainx replace the “x” with the scan chain’s number.
   sdi : sdx
   sdo: sdx
   What is the command that we are going to use?

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

8. **Concatenate** the DFFs into the newly defines **chain**. (HINT: See p583, RC_cmdref_10.1.pdf), are there any attributes required for our usage?

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

9. **Write out** the new scan enabled netlist to the file ‘assignment2_scn.v‘ in the current directory. Use the command write_hdl for this. What is the full command that you are going to use?

   write_hdl.....
   .....
   .....

10. Run all 7 commands you have compiled from task 3 to 9, and verify that you have a new netlist with all FFs replaced with scan cells, a scan chain and addition pins (scan_enable, sdi0 ans sdo0). With the 7 commands you can now write your own script file, i.e. “scan_insertion” script. This script must start with the 2 commands from the “view_assignment2.tcl” script.

    Create a “scan_insertition” script that automatically reads in a cell library, “assignment2.v” and writes out a scan enabled netlist.

B. Test Pattern Generation:

Automatic Test Pattern Generation for Scan Enabled designs

In the <work dir>/assignment2/et directory you will find a pin assignment file. It specifies to ET which pins to use to control the scan chain that was inserted with RC.
You will have to modify this file to match the pin names you used in “assignment2_scn.v”.
The full syntax of a pin assignment file is given in ET_models_10.1.pdf page 81 to 83, and includes helpful examples.

The ATPG procedure is similar to the procedure in the first exercise. In case you forgot the precise steps you took we request that you revisit the steps in lab assignments1 as you can take the same steps here.

We start in exactly the same way:

- Go to the “<work dir>/assignment2/et” directory
- Start ET
- Start a new project
- Build the model
- Build the test mode, once more “FULLSCAN” **and**
  under “Additional Options” specify the “input pin assignment file”.
- Build the fault model
- Verify test structures

The major difference between generating a test bench for assignment1 and this one is the fact that assignment2/counter is a scan enabled design, i.e. it has a scan chain present in the circuit.

1. Instead of creating a logic test, we will make several tests and combine them to make a test bench. The logical first test to prepare is the “Scan Chain flush” test,
   it is logical to perform this test on a chip first as this test tests the scan chains functionality. If the Scan Chain fails this test it is pointless to try any other scan related tests, as a broken scan chain will corrupt all of those tests aswell.

   Prepare a “Scan Chain flush” test, and comment on the coverage. Now do the same with the “Logic” test. Why is the coverage obtained by the logic test so high even when the faults inside the Scan Chains are not targeted?

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

2. Commit both tests sets, and write out the Verilog test bench.
   Why are there now 2 test files with test vectors?

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

C. Circuit Simulation:

Circuit simulation of this assignment is almost identical to the simulation for assignment1, with only a minor tweak in the nc-verilog execute command to account to the third test file in the test bench generated by Encounter Test.

1. The via ATPG acquired patterns can now be simulated as before, do note there are now two test vectors files that both should be included in the nc-sim command.

   Write down this new ncverilog command.

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

2. Execute the command compiled at task 13.

   Inspect the Console window for a log of the simulation run, write down required simulation time and number of good comparing vectors.

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