The University of Texas at Dallas
Dept. of Electrical Engineering

EEDG/CE 6301: Advanced Digital Logic
EEDG/CE 6303: Testing and Testable Design
(Instructor: Mehrdad Nourani)

A Quick Tool Setup and Tutorial for Synthesis & Test Using SYNOPSYS Toolset

DISCLAIMER: *The following steps aim at your setting SYNOPSYS toolset and running a sample VHDL program. SYNOPSYS is a large commercial CAD tool suite with many additional options that are not explained here. What we explain here is the minimum to run few tools. Nothing is guaranteed here. With this new version of SYNOPSYS that we installed, there might be problems in this document and some commands may not work exactly as explained. Proceed cautiously and use it at your own risk. I encourage you to assign enough time to familiarize yourself with this tool to be able to use it efficiently. Please report problems, corrections and suggestions about this document to* nourani@utdallas.edu*.*

1 Setup

Most of the CAD tools, including SYNOPSYS, are accessible in various labs with UNIX workstations in EC building including EC 4.308 (VLSI CAD Lab) and EC 4.324 (Solarium Lab). More specifically, using workstations in EC 4.324 is recommended due to their faster speed and larger RAM. All CAD tools required for this course, run in Solarium Lab. Due to frequent upgrades, sometimes they may not run in other labs. Many of these tools, including SYNOPSYS, provide shell commands to allow user run them.

• You need to be familiar with the basic UNIX commands and one UNIX text editor (VI, EMACS, GVIM, EDIT, etc.) . The "Quick UNIX Guide" also posted in the course website can be a good start.

• In order to run CAD tools in your UTD UNIX account, make sure that your environment variables are set correctly: Remember that in UNIX the files whose names start with a "."(e.g. **.login**) are hidden, to view them type "ls -a".

– Set up the Nomachine and connect to "engnx" srever. Please follow the instructions posted in the link below: https://personal.utdallas.edu/~xxx110230/nx/

For off-campus connection, you need VPN (check https://www.utdallas.edu/oit/vpn/). You may also need to download the nomachine remote access software (check https://www.nomachine.com/download)

– In a new shell, type in the following command at the prompt in order to enable the environment variables(running on the "engnx" Nomachine server):
source /proj/cad/startup/profile.synopsys_2018_vcs_2021

Please note that if you source differently (i.e. the way that you source CAD files for other courses), you will need to close the console and open a new one to source as explained above. This would allow the environment variables to be taken into effect correctly.

• Make a directory in your home directory. For example, SYNOPSYS. Note that "~ " refers to your root directory.
mkdir ~/SYNOPSYS

• Within this directory you should have a directory called WORK. This directory is sometimes used by Synopsys tools to hold temporary values.
mkdir ~/SYNOPSYS/WORK

2 Preparation for Testing

2.1 Netlist Requirements

In order to stop tetramax from optimizing the circuit, your RTL description should adhere to the following guidelines:

1 The RTL description can be in either Verilog or VHDL. **Ensure that the coding style must be completely structural in nature.**

2 **It is necessary to describe all the primitives used in the top level, at the beginning of the file as shown in example 1.**

3 **Alternatively, you can use the gates which are predefined in the 'gtech_lib.v' library file as shown in example 2.** The 'gtech_lib.v' can be found in the path: /home/cad/synopsys_2007.12/syn/packages/gtech/src_ver/

4 In case of VHDL, bits and vectors must only use *std_logic* types. Other types such as *SIGNED* are not supported. Conversion functions are not supported.

Example 1: Using VHDL and Basic gates described within the same file

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_components.all;
---------------------------------------
entity and is
port ( A, B : in std_logic;
Z : out std_logic);
end and;
architecture beh_and of and is
begin -- beh_and
Z <= A and B;
end beh_and;
---------------------------------------
entity or is
port ( A, B : in std_logic;
Z : out std_logic);
end or;
architecture beh_or of or is
2
begin -- beh_or
Z <= A or B;
end beh_or;
---------------------------------------
entity not is
port ( A : in std_logic;
Z : out std_logic);
end not;
architecture beh_not of not is
begin -- beh_not
Z <= not A;
end beh_not;
---------------------------------------
entity nand is
port ( A, B : in std_logic;
Z : out std_logic);
end nand;
architecture beh_nand of nand is
begin -- beh_nand
Z <= A nand B;
end beh_nand;
---------------------------------------
entity nor is
port ( A, B : in std_logic;
Z          :out  std_logic);
end nor;
architecture beh_nor of nor is
begin  -- beh_nor
```

```
Z <= A nor B;
end beh_nor;
-------------------------------------

--TOP LEVEL MODULE

ENTITY example IS
PORT(A, B, C, D, E: IN std_logic;
F: OUT std_logic);
END;

ARCHITECTURE rtl OF example IS
component and
port ( A, B : in std_logic;Z : out std_logic);end component;

component or

port ( A, B : in std_logic;Z : out std_logic);end component;


component not
port ( A : in std_logic;Z : out std_logic);end component;


component nand
port ( A, B : in std_logic;Z : out std_logic);end component;


component nor
port ( A, B : in std_logic;Z : out std_logic);end component;


SIGNAL w, x, y, z: std_logic;

BEGIN

S_2 : not port map( A => E, Z => y);
S_3 : nor port map( A => w, B => x, Z => z);
S_4 : nand port map( A => y, B => z, Z => F);
S_0 : and port map( A => A, B => B, Z => w);
S_1 : or port map( A => C, B => D, Z => x);

END;
```

Save the file above as example.vhd under ~/SYNOPSYS/WORK/ directory.

Example 2: Using Verilog and basic gates from the 'gtech_lib.v' library file

```
module example_gtech(a,b,c,d,e,F);
input a,b,c,d,e;
```

```
output F;
wire w,x,y,z;

GTECH_NOT (e,y);
GTECH_NOR2 (w,x,z);
GTECH_NAND2 (y,z,F);
GTECH_AND2 (a,b,w);
GTECH_OR2 (c,d,x);
endmodule
```

Save the file above as example_gtech.v under ~/SYNOPSYS/WORK/ directory.

2.2 TetraMAX Invocation

<shell prompt> tmax&
This invokes the Graphical User Interface (GUI) for TetraMAX shown in Figure 1.

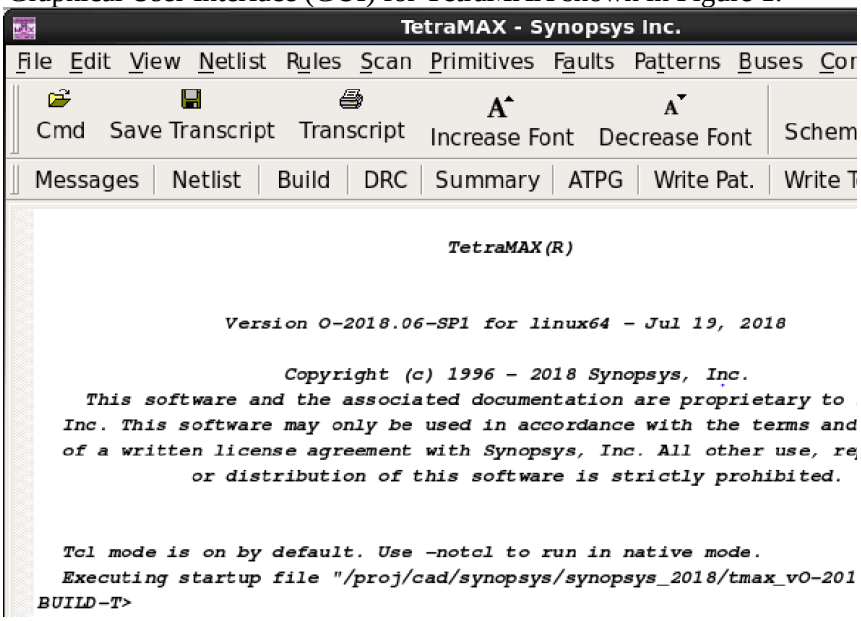


Figure 1: TetraMAX startup window.

2.3 Read Library Models

1 **Copy the file containing the verilog models of the components "gtech_lib.v' into your working directory.** It can be found in the path '/home/cad/synopsys_2007.12/syn/packages/gtech/src_ver/'.

2 **Click Netlist button in the command toolbar found at the top of the TetraMAX main window, shown in fig. 2.**

Figure 2: Command toolbar.

3 **Select the gtech.v file.**

4 **Check Library modules.**

5 **Click RUN.**

2.4 Read VHDL/Verilog File

You can read the netlist using the Read Netlist dialog box, or you can enter the *read netlist* command from the command line.

1 **Select the file, example.vhd/example_gtech.v**

2 **Click RUN.**

The following shows the *read netlist* command result.

```
------------------------------------------------------------------------
BUILD> read netlist /home/ee/n/nxa018600/SYNOPSYS/tmax/example.vhd
Begin reading netlist ( /home/ee/n/nxa018600/SYNOPSYS/tmax/example.vhd )...
External packages: ieee.std_logic_1164. Bit types: std_logic.
End parsing VHDL file ./example.vhd with 0 errors;
End reading netlist: #modules=1, top=example, #lines=20, CPU_time=0.02 sec,
Memory=0MB
------------------------------------------------------------------------
```

Note: After reading the file, if you see an error message like **Error: Module (...) referenced undefined module (...)**, it means that the file does not have the right components recognized by TMAX. Go back to design vision and do the optimization using **gtech** library. To do this, you can directly edit **.synopsys dc.setup** file and change the existing library (e.g. **class.db**) to **gtech.db** in link library, target library and symbol library lines. This step is explained in section "5.Synthesis (script based)" in the Synthesis Tutorial.

2.5 Building the ATPG Model

Building the ATPG design model takes those parts of the design that are to be part of the ATPG process and removes the hierarchy. You can build the ATPG model for your design using the Run Build Model dialog box, or you can use the *run build model* command from the command line.

1 **Click Build button in command toolbar.**

2 **Click RUN (The module you just imported should already be selected in the 'top module name' dropdown).**

The following shows the result of a build run.

```
------------------------------------------------------------
BUILD> run build_model example
```

```
---------------------------------------------------------------
Begin build model for topcut = example ...
---------------------------------------------------------------
There were 0 primitives and 0 faultable pins removed during model
6
optimizations
End build model: #primitives=11, CPU_time=0.02 sec, Memory=0MB
---------------------------------------------------------------
Begin learning analyses...
End learning analyses, total learning CPU time=0.02 sec.
---------------------------------------------------------------
```

The Graphical Schematic Viewer (GSV) toolbar is used to display the schematic view of the circuit. Click **SHOW → ALL**.It shows the schematic view of the circuit, as shown in fig.3.

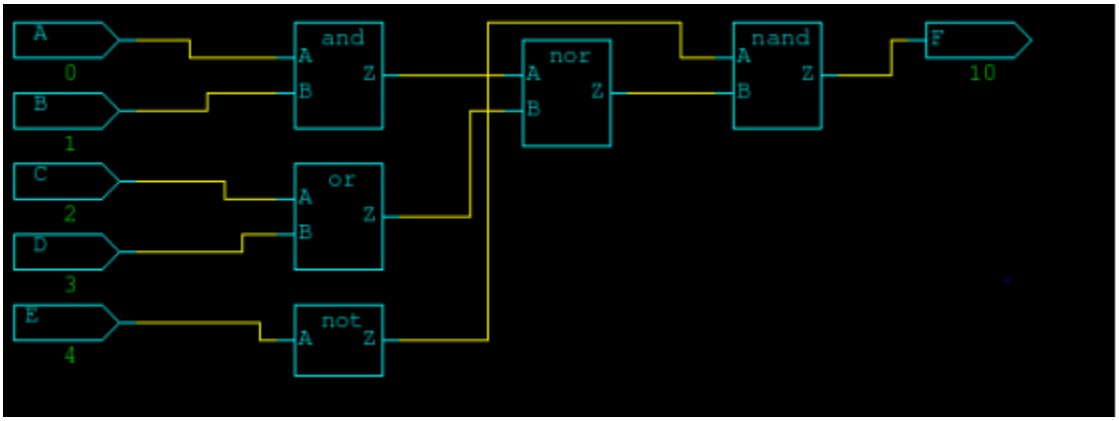


Figure 3: Schematic created by Tetramax

***** IMPORTANT : Please ensure that the schematic created by Tetramax matches the schematic you intend to create (or provided in the assignment question). If the schematic does not match with the hand-drawn schematic, further analysis will yield significantly different results. In some cases, it has been reported that, though the schematic looks like what is expected, the tool internally optimizes the circuit while running the ATPG. To avoid such issues, Please conform to the guidelines mentioned in section 2.1 *****

2.6 Perform Design Rule Check

You can perform DRC using the Run DRC dialog box, or you can execute the *run drc* command from the command line.

1 **Click the DRC button in the command toolbar.**

2 **Click RUN.**

The result of a typical DRC run is shown below. Make sure that your design reports no violations (bolded below).

```
DRC> run drc
---------------------------------------------------------------
Begin scan design rules checking...
```

```
-------------------------------------------------------------
Begin simulating test protocol procedures...
Test protocol simulation completed, CPU time=0.00 sec.
-------------------------------------------------------------
Begin scan chain operation checking...
Scan chain operation checking completed, CPU time=0.00 sec.
-------------------------------------------------------------
Begin nonscan rules checking...
Nonscan cell summary: #DFF=0 #DLAT=0 tla_usage_type=none
Nonscan rules checking completed, CPU time=0.00 sec.
-------------------------------------------------------------
Begin DRC dependent learning...
Fast-sequential depth results: control=0(0), observe=0(0),
detect=0(0), CPU time=0.00 sec
DRC dependent learning completed, CPU time=0.00 sec.
-------------------------------------------------------------
DRC Summary Report
-------------------------------------------------------------
```
No violations occurred during DRC process.
```
Design rules checking was successful, total CPU time=0.01 sec.
-------------------------------------------------------------
```

3 Perform ATPG

1 **Click the ATPG button.**
Make sure the pattern source is *internal* and *Add all faults* is selected for fault source.

2 **Click RUN.**
The result of an ATPG run is shown below.

Note: *When you add all faults, Tetramax adds all uncollapsed faults for every input and output pin of each gate, PI and PO, even if they share a wire. This means that you might not see the same number of faults as you do in your hand calculations. This is why you see a total of 40 faults in the reports below, which is more than you might expect for the example circuit.*

```
----------------------------------------------------------------
ATPG performed for stuck fault model using internal pattern source.
----------------------------------------------------------
#patterns #faults #ATPG faults test process
stored detect/active red/au/abort coverage CPU time
--------- ------------ ------------ -------- --------
Begin deterministic ATPG: #uncollapsed_faults=40, abort_limit=10...
6 40 0 0/0/0 100.00% 0.00
Uncollapsed Stuck Fault Summary Report
-----------------------------------------------
fault class code #faults
------------------------------ ---- ---------
Detected DT 40
Possibly detected PT 0
```

Undetectable UD 0
ATPG untestable AU 0
Not detected ND 0
---
total faults 40
8
test coverage 100.00%

Pattern Summary Report

#internal patterns 6
#basic_scan patterns 6

Note: In order to observe the **fault coverage** value, you need to select this feature in **Faults→Set Fault Options**. You might also want to select "collapsed" in Set Fault Options and then run ATPG.

3.1 Report Faults

1 **Faults → Report Faults...**

2 **Select Report Type as All.**

3 **Click OK.**

4 **You can refer Chapter 10, pages 6 to 9, of the tetramax user guide for more details about the reported fault types/categories.**

To report the collapsed faults, check the option **Report Collapsed**.

Note: there appears to be a bug in the GUI where Report Collapsed may not work unless you also choose the Verbose option.

3.2 Report Patterns

1 **Patterns → Report Patterns...**

2 **Select Report Type as All.**

3 **Click OK.** Reported patterns are shown in fig. 4 for the example.

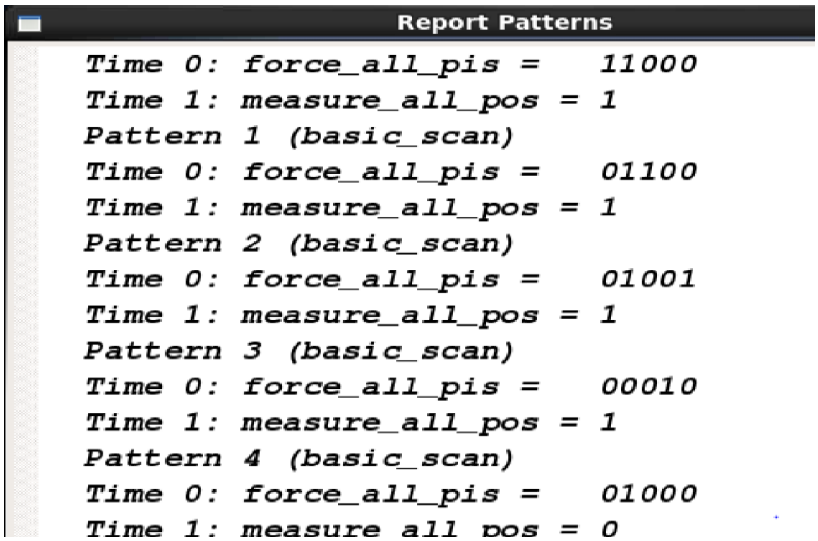


Figure 4: Reported Patterns for the given example

3.3 Pattern Compression

1 **ATPG → Basic Scan Settings...**

2 **Select 'Low' in Merge effort and 'Add all faults' in Fault source.**

3 **Click Run.**

The following shows the result of the test pattern compression.

```
ATPG performed for stuck fault model using internal pattern source.
-----------------------------------------------------------
#patterns     #faults     #ATPG faults  test      process
stored     detect/active  red/au/abort  coverage  CPU time
---------  -------------  ------------  --------  --------
Begin deterministic ATPG: #uncollapsed_faults=40, abort_limit=10...
6             40      0         0/0/0   100.00%      0.00
```

3.4 Format Vectors for ATPG

We must convert TetraMAX representation of the vectors to a readable format.

1 **Click the Write Pat button.**

2 **Fill the output file name.**

3 **Select the file format as Binary.**

4 **Click OK.**

3.5 Applying External or Random Vectors

When you run ATPG, by default Tetramax users "Internal" patterns using its internal test generation mechanism. However, the ATPG tool in TetraMAX has also options to accept external patterns (provided by user) or apply random patterns (using internal random pattern generator). The procedure is the same as generic running of ATPG except

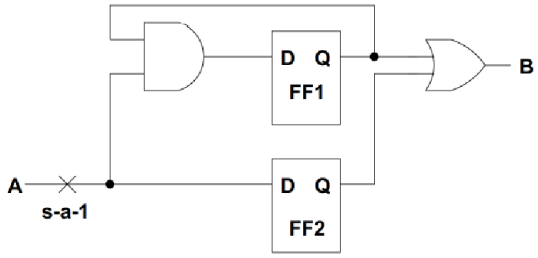
1 **Click the ATPG button → General ATPG Settings.**
Now you can choose *Random* (or *External* is you have other patterns of interest) and *Add all faults* is selected for fault source. **Uncheck** *Case sensitive*, **Check** *Read again* as shown in fig. 5.

Figure 5: options in the ATPG window

2 **Click RUN.**
You can also specify number of random patterns that you want Tetramax to apply. Note also that when you use random patterns in every run, the detected faults (and thus the fault coverage) will be different especially if you set the number of patterns to be small. If you intend to re-generate the same result, you better store the random patterns and in other runs use the *External* option.

4 ATPG for Sequential Designs

4.1 VHDL Netlist Requirements



Note: The D flip-flop entity does not have a reset pin.

Consider the following small sequential circuit example with two D flip-flops:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity and2 is
port ( A, B : in std_logic;
Z : out std_logic);
end and2;
architecture beh_and2 of and2 is
begin -- beh_and2
Z <= A and B;
end beh_and2;
---------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity or2 is
port ( A, B : in std_logic;
Z : out std_logic);
end or2;
architecture beh_or2 of or2 is
begin -- beh_or2
Z <= A or B;
end beh_or2;
---------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity dff is
port ( D, clk : in std_logic;
Q : out std_logic);
end dff;
architecture beh_dff of dff is
begin -- beh_dff
process(clk,D)
```

```
begin
if (clk'event and clk = '1') then Q <= D;
end if;
end process;
end beh_dff;
---------------------------------------
--TOP LEVEL MODULE
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY example IS
PORT(A, clk : IN std_logic;
B: OUT std_logic);
END;
ARCHITECTURE rtl OF example IS
component and2
port ( A, B : in std_logic;
Z : out std_logic);
end component;
component or2
port ( A, B : in std_logic;
Z : out std_logic);
end component;
component dff
port ( D, clk : in std_logic;
Q : out std_logic);
end component;
SIGNAL x, y, z: std_logic;
BEGIN
S_0 : and2 port map( A => A, B => y, Z => x);
S_1 : dff port map( D => x, clk => clk, Q => y);
S_2 : dff port map( D => A, clk => clk, Q => z);
S_3 : or2 port map( A => y, B => z, Z => B);
END;
```

0) **Ensure that you are logged on to the apache/giant server or are using nomachine. If you are not connected to this server, please use the command → ssh -Y apache.utdallas.edu**

1) **Enable environment setting: source /proj/cad/startup/profile.synopsys_2018_vcs_2021**

2) **Go to your working directory. For example: cd /ee6303/work/**

3) **Put the file ".synopsys dc.setup" in your working directory where you run design vision.**

4) **In the shell prompt, enter the command → design vision**

5) **Read the vhdl file into design vision → File → Read**

6) **Select the top-level design (in this case→example)**

7) **Click Design → Compile, check "Fix design rules only"! OK**

8) **Save the top-level design in VHDL format.**

9) **Save it as example rtl.vhd.**

This will save your design in complete structural model and non-optimized version.

4.2 ATPG Process

0) **Copy class.v file from "/home/cad/synopsys_2007.12/syn/doc/syn/dft_tutorial/LIB" into your working directory.**

1) **In the shell prompt, enter the command→ tmax**

2) **Read the verilog models of the components : Click Netlist → Read Netlist, Choose "class.v", check "library modules" → Run**

3) **Read the design file: Click Netlist →Read Netlist →Choose example_rtl.vhd, →Check "Enhanced Seq Model", → Run**

5) **Build the Model. Use the GSV schematic viewer to ensure that the circuit has been built as you intended. If it doesn't match your expectations, you may want to go back and play around with the "Set Build Options" and then rebuild the design.**

6) **Perform DRC.**

7) **Faults→Set Faults Options→Check Fault Coverage → Run**

8) **Click Open GSV → Show → All. Check if the schematic shown is as per your expectations.** ***Do not forget to include a snapshot of this schematic in your report.***

9) Click on ATPG, check "Add all faults", Click "Enable Full-Seq ATPG" in General ATPG settings frame, uncheck "Random Fill" in Full Sequential Settings frame → Run

For further analysis, follow the same procedure described in the ATPG process of combinational circuits earlier in the tutorial.

4.3 ATPG Results

```
TEST> run atpg
```

```
ATPG performed for stuck fault model using internal pattern source.
-----------------------------------------------------------
#patterns #faults #ATPG faults test process
stored detect/active red/au/abort coverage CPU time
--------- ------------ ------------ -------- --------
Begin deterministic ATPG: #uncollapsed_faults=30, abort_limit=10...
0 0 0 0/2/0 0.00% 0.00
-----------------------------------------------------------------
Begin Full-Sequential ATPG for 30 uncollapsed faults ...
--- abort limit : 10 seconds, NO BACKTRACK LIMIT
-----------------------------------------------------------------
#patterns #faults #ATPG faults test process
stored detect/active red/au/abort coverage CPU time
--------- ------------ ------------ -------- ----------
1 10 20 0/0/0 45.00% 0.01
2 6 14 0/0/0 65.00% 0.02
2 0 14 0/0/1 65.00% 10.07
2 0 2 0/9/1 65.00% 10.08
2 faults were identified as detected by implication,
TEST COVERAGE is now 68.33%.
Uncollapsed Stuck Fault Summary Report
-----------------------------------------------
fault class code #faults
------------------------------ ---- ---------
Detected DT 18
Possibly detected PT 5
Undetectable UD 0
ATPG untestable AU 6
Not detected ND 1
-----------------------------------------------
total faults 30
test coverage 68.33%
-----------------------------------------------
Pattern Summary Report
-----------------------------------------------
#internal patterns 2
#full_sequential patterns 2
```

4.4 Automatic Scan Cell Insertion

• Write the HDL code (e.g. **file_name.v**) describing your sequential circuit. It is recommended to use GTECH logic gates and GTECH_FD1 or similar for flip flops. Here is an example circuit suitable for Design for Test Scan cells:

```
module DFT_example(clk,a,b,c,d,e,F);
input clk; //Scan clk
input a,b,c,d,e;
output F;
wire u, v, w,x,y,z;

GTECH_NOT (e,y);
GTECH_NOR2 (w,x,z);
GTECH_FD1  (u, clk, F); //DFF which will become scan
GTECH_NAND2 (y,z,u);
GTECH_AND2 (a,b,v);
GTECH_FD1  (v, clk, w); //DFF which will become scan
GTECH_OR2 (c,d,x);
endmodule
```

• Open **design_vision** environment. In Setup, use **class.db, class.db**, and **class.sdb** as the link, target and symbol libraries, respectively.

• Analyze and elaborate your design.

• Click Design→ Compile, check "Fix design rules only" → OK

• View the schematic and check if all connections are correct.

• The following **dc_shell** commands can be used to replace normal flip-flops with scan flip-flop. Alternatively, you can use the command window in **design_vision** to not only execute the commands, but also view schematics when available.

– Select scan style:
set_scan_configuration -style multiplexed_flip_flop
Other styles include lssd, clocked_scan, etc.

– Choose the specifications of your desired scan architecture:
set_scan_configuration -chain_count 1
Other options include the number and length of scan chains.

– Specify the DFT clock for DRC and DFT insertion. Assume "clk" is the clock port used in your design:
set_dft_signal -view existing_dft -type ScanClock -port clk -timing {1 10}

– Create the test protocol:
create_test_protocol -capture_procedure single_clock

– Insert the scan chain:
insert_dft

Note: Try combining all the above commands into a TCL script! Click the "History" tab at the bottom of Design Vision. Your TCL can be run later using File > Execute Script...

– Check design rule for any violation:
current_design entity_name (Here, entity name refers to your top module name)

Ensure that there are no violations using:
dft_drc -verbose

– Check the circuit with inserted scan:
check_design -multiple_designs

– Generate test protocol (stil) file which keeps the information for interfacing and interaction with the circuit during scan test:
write_test_protocol -o <file_name_scan>.stil
-Open this file to see what information it has

– Save the new architecture with scan cells:
write -hierarchy -format verilog -output <file_name_scan>.v
-Open this file to see what has changed

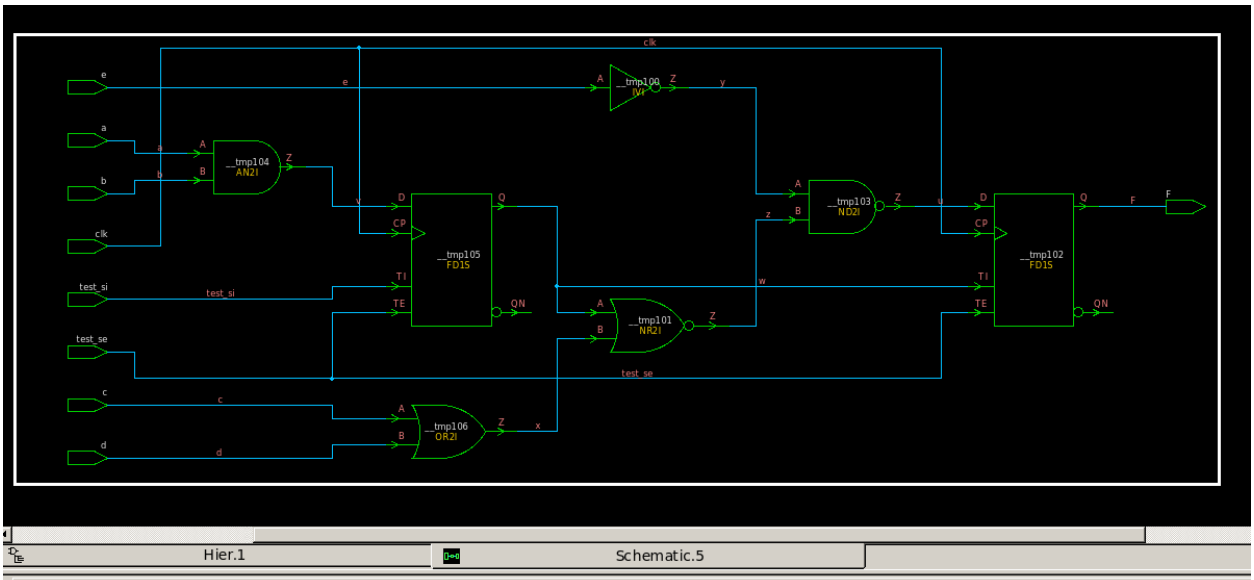
• You can generate reports on area and delay of the synthesized scan-inserted circuit and see the schematic of the circuit created by SYNOPSYS.

• Tetramax only accepts designs configured with class library. In order to use your DFT design with scan cells, you will need ***class.v*** library which is located:
/proj/cad/synopsys/synopsys_2018/syn_vO-2018.06-SP1/doc/syn/dft_tutorial/LIB/class.v

Now, when you go to Tetramax, you follow the same steps as for a combinational circuit. Only here, you need both HDL file of the new (with scan cells inserted) structure and protocol (stil) file for analysis. In DRC step of Tmax, you choose the test protocol (stil) file and get the fault coverage of the testable design. Note that if you have any violations in the design you won't be able to test it in Tetramax ATPG.

• The manual of each command can be found using the command:
man <command_name>

Try looking up a few commands to see their options! Or use the TetraMax ATPG User Guide.




```
  ...reading user specified asynchronous signals...

Information: Starting test design rule checking (TEST-222)
Test Design rule checking did not find violations
Information: Test design rule checking completed (TEST-123)
  Architecting Scan Chains
  Inserting Scan Cells
  Routing Scan Chains
  Routing Global Signals
  Mapping New Logic
Resetting current test mode

design_vision>
Current design is 'DFT_example'.
```

- Here is an example of what the circuit should look like after **insert_dft** has been successfully run.
- Notice that the DFFs have more pins than normal, and they form a scan chain.
- Notice that signals “test_si” (scan input) and “test_se” (scan enable) input signals have been added to the circuit automatically. In this case, the scan chain output is simply the output pin F.

Test Tutorial Addendum: Path Delay Test

5 Path Delay Test

The following procedure allows path-delay test analysis in TetraMAX:

- Just like any design for use in TetraMAX, write your HDL code, source properly, parse the design, compile it in **`design_vision`** using gtech libraries, and save it as an hdl file. Figure 1 shows an example circuit that implements a mux, and the Verilog code that can be used with the gtech library.

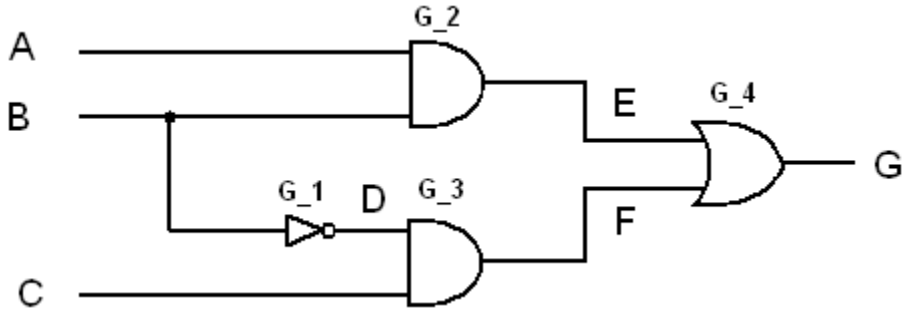


Figure 1: example_mux

```
module example_mux(A, B, C, G);
input A, B, C;
output G;

wire D, E, F;

GTECH_NOT  G_1 (B, D);
GTECH_AND2 G_2 (A, B, E);
GTECH_AND2 G_3 (C, D, F);
GTECH_OR2  G_4 (E, F, G);

endmodule
```

- In design_vision, you can get information on the gates and net names from the netlist that has been generated using the following command:

```
report_net -connections
```

You can use this information to write the path files.

- Manually write two path delay files in textual format (.txt), one for robust tests and the other for non-robust tests. In each file, have rising and falling transitions for all the paths in the circuit. For example if you have four paths in the circuit (see Figure 1), you would need to write a total of 16 paths, 8 robust paths (4 rising and 4 falling paths), and 8 nonrobust paths (4 rising and 4 falling paths).

- A sample of 8 robust paths for the circuit in Figure 1 is provided at the end of this document. Each path consists of a name, and the rising (or falling) transitions on the path (from a primary input to primary output using the net information you captured in the previous step). Rising transitions are

shown with symbol "^", and falling transitions are shown with "v". Also note that the same transition should be maintained if the path enters a non-inverting gate, and should be changed otherwise.

- Robust tests require conditions (constraints) on the off-path inputs, as can be seen in the sample path file.

- Non-robust tests do not require condition, since the tool generates the non-robust test automatically. (Exclude the conditions for the non-robust path file.)

- In Tmax&, just like before, run "Netlist" for gtech_lib.v library module, run "Netlist" for your synthesized vhdl design saved in design_vision, run "Build", and then run "DRC".

  Note: Tmax will attempt to optimize circuits during the build stage. If the resulting circuit diagram is not as expected, you can prevent these optimizations by modifying the **"Set Build Options"** *dialog before running the build model. For this example, you may need to set the MUX optimizations to "None" to get the circuit as shown below.*

- Under "Faults"-> "Set Fault Options", check the "Path delay", "Collapsed", "Fault coverage", and "Verbose" options, and then OK.

- Under "Faults"-> "Add Delay Paths", browse to choose your path file (robust or nonrobust). *Make sure you proceed through this step without any errors.*

- In "ATPG", under "Full Sequential Settings", set the "Merge effort" to "Medium". Also, check the "Add all faults" option, and then "Run". This results in the path delay ATPG.

- Under "Faults"-> "Report Delay Paths", select "All", and then OK. This would report the paths used for test.

- Under "Faults"-> "Report Delay Paths", select "Path Name", and enter the name of a path in the path file(s) (e.g. "path1r" which is the AEG robust rising path), check the "GSV Display" and "Set pindata to delay data" options, and then OK. This command would display the particular path with all the transitions on the lines, indicating that the transitions on the input would propagate to the output and would cause a proper transition at the output with a permitted delay.

- Repeat the previous step for all paths in the design.

- Under "Patterns"-> "Report Patterns", select "All", and run it with and without the "Path delay" option checked. This second report tells you which test patterns correspond to each path, the type of path (rising or falling) and the Fault Class (Detected or detected_robustly).

- Compare the patterns output by TetraMax with the patterns obtained by hand.

Sample Path File (don't copy the page numbers!):

```
//---Path AEG Robust Rising
$path {
$name path1r ;
$transition {
A ^;
G_2/U1/Z ^;
G_4/U1/Z ^;
G ^ ;
}
$condition{ //conditions for robust test case
B 11;
}
}
//---Path AEG Robust Falling
$path { $name path1f ;
$transition {
A v;
G_2/U1/Z v;
G_4/U1/Z v;
G v ;
}
$condition{ //conditions for robust test case
B 11;
}
}
//---Path BEG Robust Rising
$path { $name path2r ;
$transition {
B ^;
G_2/U1/A ^;
G_2/U1/Z ^;
G_4/U1/Z ^;
G ^ ;
}
$condition{ //conditions for robust test case
A 11;
C 00;
}
}
//---Path BEG Robust Falling
$path { $name path2f ;
$transition {
B v;
G_2/U1/A v;
G_2/U1/Z v;
G_4/U1/Z v;
G v ;
}
$condition{ //conditions for robust test case
A 11;
C 00;
}
}

//---Path BDFG Robust Rising
$path { $name path3r ;
$transition {
```

```
B ^;
G_1/U1/A ^;
G_3/U1/B v; //transition changes after inverting gate
G_4/U1/Z v;
G v ;
}
$condition{ //conditions for robust test case
A 00;
C 11;
}
}
//---Path BDFG Robust Falling
$path { $name path3f ;
$transition {
B v;
G_1/U1/A v;
G_3/U1/B ^; //transition changes after inverting gate
G_4/U1/Z ^;
G ^ ;
}
$condition{
A 00;
C 11;
}
}
//---Path CFG Robust Rising
$path { $name path4r ;
$transition {
C ^;
G_3/U1/Z ^;
G_4/U1/Z ^;
G ^ ;
}
$condition{ //conditions for robust test case
B 00;
A 11;
}
}
//---Path CFG Robust Falling
$path { $name path4f ;
$transition {
C v;
G_3/U1/Z v;
G_4/U1/Z v;
G v ;
}
$condition{ //conditions for robust test case
B 00;
A 11;
}
}
```

Tool Command Language "TCL" scripts

- Notice that whenever you select an option or run a command in the Tmax GUI, all it is doing is inserting text into the command line. As you are running commands, try copying these commands into a separate text file and save it as "setup.tcl" (your previous commands are also available in the "history" tab in Tmax). This allows you to repeat everything that you have done quickly and also keeps a record of the steps you have taken. To run a .tcl script, you can call it from within Tmax using the "Cmd" button. Or call it from the terminal:

```
tmax <filepath>/setup.tcl
```

- Using TCL scripts is entirely optional, but it is a powerful tool for increasing productivity. In general it is more efficient to use tcl scripts than the GUI, and it should be your goal to use them as much as possible once you are comfortable with a tool. Advanced users might never use the GUI at all.

- Here is an example TCL script which performs all the steps you would normally do in the GUI for path delay testing. Try going through this script line by line and notice that each command directly corresponds to something you could do in the GUI.

  NOTE: this script might not work if you don't pay attention to filepaths! You may have to modify either the script or your file structure so that they match. While not required, you are encouraged to generate your own .tcl scripts from scratch.

```
read_netlist ./gtech_lib.v
read_netlist ./delay_test/example_mux.v
set_build -merge nomux_from_gate
run_build_model example_mux
run_drc
set_faults -model path_delay -report collapsed -fault_coverage -
summary verbose
add_delay_paths ./delay_test/non_robust_path.txt
add_faults -all
run_atpg -ndetects 1
report_patterns -all -internal -path_delay
report_patterns -all -internal
write_patterns  ./delay_test/output_patterns  -internal  -format
stil99 -cycle_count -nopatinfo -parallel 0 -cellnames parallel
```

- Running this script should read in a design "example_mux.v", perform setup, run the atpg, report the results in the TetraMax log, and write some results to a file.

Example output from TCL script:

```
    Collapsed Path_delay Fault Summary Report
-----------------------------------------------
fault class                     code   #faults
------------------------------  ----  ---------
Detected                         DT          8
  detected_by_simulation         DS        (1)
  detected_robustly              DR        (7)
Possibly detected                PT          0
Undetectable                     UD          0
ATPG untestable                  AU          0
Not detected                     ND          0
-----------------------------------------------
total faults                                 8
test coverage                           100.00%
fault coverage                          100.00%
ATPG effectiveness                      100.00%
-----------------------------------------------
           Pattern Summary Report
-----------------------------------------------
#internal patterns                           8
    #fast_sequential patterns                8
-----------------------------------------------
report_patterns -all -internal -path_delay
Pattern  Fault            Fault  Fault  Launch
num      name             type   class  clock
-------  ---------------  -----  -----  --------
0        path1r           str    DR     -
1        path4f           stf    DR     -
2        path4r           str    DR     -
3        path1f           stf    DR     -
4        path2r           str    DS     -
5        path3f           stf    DR     -
6        path3r           str    DR     -
7        path2f           stf    DR     -
```

```
report_patterns -all -internal
 Pattern 0 (fast_sequential)
 Time 0: force_all_pis =   010
 Time 1: force_all_pis =   110
 Time 2: measure_all_pos = 1
 Pattern 1 (fast_sequential)
 Time 0: force_all_pis =   101
 Time 1: force_all_pis =   100
 Time 2: measure_all_pos = 0
 Pattern 2 (fast_sequential)
 Time 0: force_all_pis =   000
 Time 1: force_all_pis =   101
 Time 2: measure_all_pos = 1
 Pattern 3 (fast_sequential)
 Time 0: force_all_pis =   110
 Time 1: force_all_pis =   010
 Time 2: measure_all_pos = 0
 Pattern 4 (fast_sequential)
 Time 0: force_all_pis =   100
 Time 1: force_all_pis =   111
 Time 2: measure_all_pos = 1
 Pattern 5 (fast_sequential)
 Time 0: force_all_pis =   011
 Time 1: force_all_pis =   001
 Time 2: measure_all_pos = 1
 Pattern 6 (fast_sequential)
 Time 0: force_all_pis =   101
 Time 1: force_all_pis =   011
 Time 2: measure_all_pos = 0
 Pattern 7 (fast_sequential)
 Time 0: force_all_pis =   110
 Time 1: force_all_pis =   100
 Time 2: measure_all_pos = 0
```