17 October 2024

TESTING AND FAULT TOLERANCE

Laboratory Session 2: "ATPG & Fault Simulation"

Student Name	Student Surname	Student ID

SAVE THIS FILE IN PDF AND SUBMIT IT ON "EXERCISE"

Politecnico di Torino

Riccardo Cantoro
(riccardo.cantoro@polito.it)

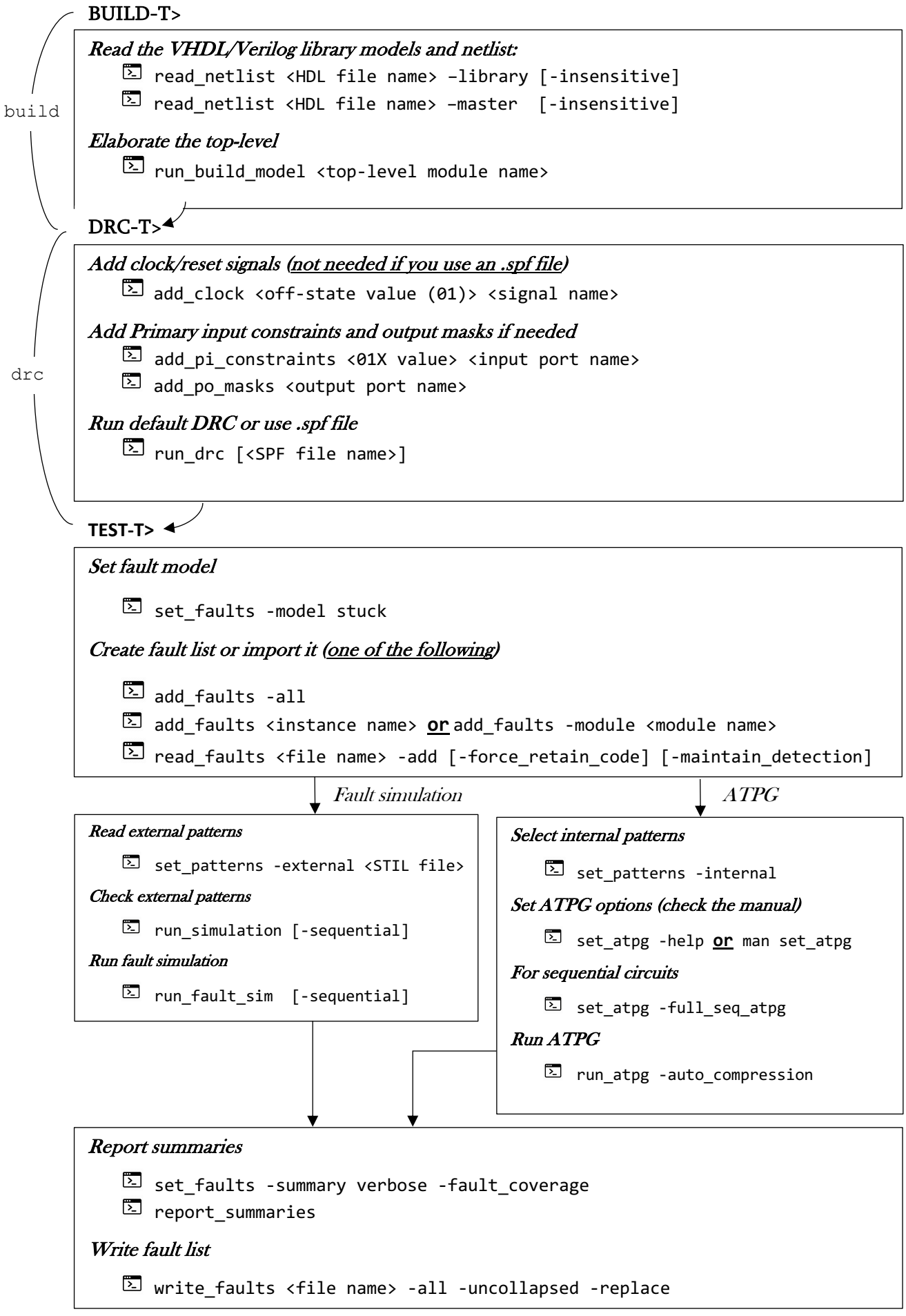
Michelangelo Bartolomucci
(michelangelo.bartolomucci@polito.it)

Files of LAB2


All files listed here are included in your remote /home directory under lab2 folder.

Filename	Description
`b10.v`	ITC'99 `b10` netlist (sequential circuit)
`b12.spf`	ITC'99 `b10` `STIL` procedure file
`b12.v`	ITC'99 `b12` netlist (sequential circuit)
`b12.spf`	ITC'99 `b12` `STIL` procedure file
`c6288.vhd`	ISCAS'85 `c6288` netlist (combinational circuit)
`c6288.stil`	ISCAS'85 `STIL` procedure file and test patterns
`pt2002.v`	Technology library models

TestMAX Flow



[A] Fault Simulation of External Patterns

This exercise aims to implement a TCL script that applies a fault simulation flow for the stuck-at fault model. `TestMAX` can then execute the script as follows:

```
tmax your_script.tcl -shell
```

`TestMAX` can simulate test patterns defined in different formats, such as `STIL` (Standard Test Interface Language). Patterns stored in `STIL` files comply with the test procedures, which are also described in the same file and read by `TestMAX` during the Design Rule Checking (DRC) process. As an example, the following code snippet shows two combinational patterns stored in a `STIL` file:

```
"pattern 0": Call "capture" { "_pi"=10100; "_po"=HLLHLHLHL; }
"pattern 1": Call "capture" { "_pi"=11001; "_po"=LHHHLHLHL; }
```

In the example, `_pi` is followed by the values to be applied to the Primary Inputs, while _po is followed by the values expected on the Primary Outputs when the patterns are applied.

[A.1] Tasks:

1. Check the contents of the `STIL` files in the folder and complete the table below
2. Create a `TestMAX` script for the fault simulation of the `C6288` benchmark circuit by using the external test patterns stored in the `STIL` file. Add all stuck-at faults in the circuit while allowing the masking of some of the primary outputs (see TestMAX Flow).

Circuit	PIs	POs	Clock signals	Test patterns
C6288				
b10				
b12				

In the DRC command mode, some of the primary outputs can be masked. This means that a fault effect propagated to such an output is not considered as detected by the fault simulator. PO masks are added during the DRC mode of the flow. For instance, if you want to mask all possible ports but two, you can do it via the following flow:

```
DRC-T> add_po_masks -all
```

```
DRC-T> remove_po_masks <output port 1 name>
DRC-T> remove_po_masks <output port 2 name>
```

It is possible to get the list of primary output pins and the current PO masks with the following commands:

```
DRC-T> report_primitives -pos
DRC-T> report_po_masks
```

[A.2] Tasks:

1. Complete the following table by applying the proper PO masks to the `C6288` circuit:

PO Masks	Test patterns	Test coverage	Not controlled faults	Not observed faults
none				
`datao(1)`				
`datao(2)`				
`datao(3)`				
`datao(1)` to `datao(3)`				
all but `datao(1)` to datao(3)				
all				

[B] Combinational ATPG

In this exercise, `TestMAX` generates the patterns internally using the Automatic Test Pattern Generation (`ATPG`) process. As for the fault simulation, you can create a TCL script by following the guidelines in the figure. The script shall generate test patterns for a combinational circuit by using all stuck-at faults, allowing the masking of some Primary Outputs (as for the fault simulation script) and applying constraints to some Primary Inputs (PIs).

You can use `STIL` procedures to force a constrained port to a state other than the requested constrained value for a limited number of tester cycles and then return the port to its constrained value. For example, you might want to hold a global reset port to an off state for general `ATPG` patterns but then allow it to be asserted to initialize the design. You can reproduce this behavior using `TestMAX` during the DRC phase, where it is possible to define constraints on the PIs as follows:

```
DRC-T> add_pi_constraints 0 <port name>     # the pin is tied'0
DRC-T> add_pi_constraints 1 <port name>     # the pin is tied'1
DRC-T> add_pi_constraints x <port name>     # the pin is not controllable
```

The command can be repeated for each primary input pin to be constrained. The list of primary inputs can be obtained as follows:

```
DRC-T> report_primitives -pis
```

You can also define restrictions on internal signals, which cannot be constrained with the `add_pi_constraints` command. Such features are not needed for the purposes of this exercise and you can check the user manual if you are interested.

[B.1] Tasks:

1. Complete the following table after running the `ATPG` on the `C6288` circuit, each time applying the requested primary input constraints and primary output masks while gathering the results from the `TestMAX` reports.

PI Constraints	PO Masks	Test Patterns	Test coverage	ATPG Untestable faults	Aborted faults
none	*none*				
`datai(0)` = 0	*none*				
`datai(0)` = 1	*none*				
`datai(1)` = X	*none*				
`datai(0)` to `datai(2)` = 0	*none*				
`datai(0)` to `datai(2)` = 1	*none*				
`datai(0)` to `datai(2)` = X	*none*				
`datai(0)` to `datai(2)` = X	`datao(1)` to `datao(3)`				
`datai(0)` to `datai(2)` = X	All but `datao(1)` to `datao(3)`				
none	*all*				

[C] Full Sequential ATPG

In the case of sequential circuits, the ATPG process using the functional input and output pins of the circuit is performed in the so-called *Full Sequential Mode* (which is activated using the proper setting, as in the figure). Moreover, you need to define clocks and reset signals, either using a STIL file or explicitly in the DRC mode.

[C.1] Tasks:

1. Complete the following table after running the full sequential ATPG on the b10 and b12 benchmarks, which are sequential circuits.

Circuit	Stuck-at faults	Test Patterns	Test coverage	Aborted faults	CPU time
b10					
b12					

Since the process may require a lot of time (high complexity), it is possible to run it in background using **byobu**, an enhanced terminal multiplexer SW as follows:

1. Connect to the remote server and run byobu on the terminal
2. Run the TestMAX script in the byobu shell
3. Press **F6** to *detach* the byobu shell. Now you can disconnect from the remote server, the work will continue in background
4. When you connect again to the remote server, run again byobu and you will *attach* to the shell.

Appendix: Advanced Techniques

[i] Fault-free simulation

A fault-free (or *gold*) machine simulation using the external patterns should be performed before running a fault simulation to compare the TestMAX simulation responses with the expected responses found in the patterns. If the gold machine simulation reports errors, there is little value in proceeding to run fault simulation.

The gold machine simulation can be performed with the following command:

```
TEST-T> run_simulation -sequential
```

In case of error, TestMAX reports a list of messages indicating the simulation mismatches, as in the following example:

```
TEST-T> run_simulation -sequential
Begin sequential simulation of 36 external patterns.
 10 o_y2  (exp=1, got=0)
 20 o_y10 (exp=1, got=0)
Simulation completed: #patterns=36/102, #fail_pats=2(0), #failing_meas=2(0)
```

[ii] Fault Injection using TestMAX

The run_simulation command can be used to perform a logic simulation of the current pattern source determined by the set_patterns command and it reports any differences between simulated and expected values. This simulation can be performed in the presence of a selected fault and will report all failures that result from the fault.

After performing a fault-simulation, one fault marked as Detected by Simulation (DS) can be selected from the fault list and injected in the circuit.

For a logic defect, there are four types of fault behavior that you can select by means of the -stuck option: `stuck-at-0 (0), stuck-at-1 (1), stuck-at-01 (01),` and `stuck-at-X (X).` For the stuck-at-01 fault type, the value is fixed either to 0 or 1 for each pattern for all affected fanout branches. For the stuck-at-X fault type, the value is fixed either to 0 or 1 for each time frame for all affected fanout branches. All affected fanout branches have the same value.

The -stuck option supports multiple arguments of the same type. You can specify a maximum of ten logic faults for different pins, however only one injection is supported for a `stuck-at-01` logic defect.

The following example simulates a single stuck-at-01 fault:

```
TEST-T> run_simulation -stuck { 01 ucore/freg/u540 }
```

The following example simulates a stuck-at-0 and a stuck-at-1 fault:

```
TEST-T> run_simulation -stuck { 0 ucore/freg/u540 1 ucore/alu/u27 }
```

Transition delay faults can also be injected in the circuit. The following example simulates a rising transition delay fault in functional logic:

```
TEST-T>  run_simulation -slow { r ucore/freg/u540 }
```

[iii] Modify STIL Patterns

The patterns stored in the `STIL` file contain both primary input and primary output values. If the primary input values are modified, the primary output values must be adjusted accordingly. You can use `TestMAX` to simulate fault-free circuit patterns and update the primary output values. The updated patterns can be written back into a `STIL` file, as in the following example:

```
TEST-T> set_patterns -external patterns.stil
TEST-T> run_simulation -sequential -update
TEST-T> write_patterns patterns_update.stil -external –format stil
```

As an exercise, you can duplicate the C6288 STIL file and replace the _pi signals of the original patterns with the following values:

```
01010101010101011111111111111111
10101010101010101111111111111111
01010101010101011101111111111111
10101010101010101101111111111111
11011011011011011111111111111111
01101101101101101111111111111111
10110110110110111111111111111111
11111111111111111010101010101010
11111111111111110110101010101010
11111111111111101101010101010101
11111111111111100110101010101010
11111111111111110000000000000000
```

Then, you can run the TestMAX script to update the POs accordingly, followed by a fault simulation to evaluate the new fault coverage.

[iv] N-times Detection

When you run the `ATPG` and the fault simulation processes with the default options, `TestMAX` considers a fault as detected when a difference is propagated to an observable primary output. By using the N-times detection option (`ndetects`), you can specify how many times a fault must be excited and propagated to an observable primary output before being considered as detected, as in the following examples:

```
TEST-T> run_fault_sim –sequential –ndetects 5
TEST-T> run_atpg –auto_compression –ndetects 5
```

As an exercise, you can play with the N-times detection option during the `C6288` fault simulation and see how the fault coverage changes concerning the one obtained in the first exercise. Moreover, you can do the same for the `ATPG` and compare the results with the previous exercises.

[v] Incremental ATPG

The ATPG process may typically be iterated until sufficient fault coverage is obtained. Especially for large designs, the first execution of the ATPG process is relatively ineffective. This generally is due to the default ATPG options or specific testing strategies.

For example, the first run may be executed by limiting the number of patterns to generate and trying to compact them as well as possible. This can be done as in the following example:

```
TEST-T> set_atpg -patterns 500 -merge high
TEST-T> run_atpg
```

At the end of an ATPG process, the newly generated patterns are added to the internal pattern set, while the current fault list is updated. A new ATPG process can be started on the current fault list.

Moreover, you can run an ATPG process after a fault simulation as follows:

```
TEST-T> set_patterns –external patterns.stil
TEST-T> run_fault_sim -sequential
TEST-T> set_patterns -internal
TEST-T> run_atpg
```

Another parameter that may impact the effectiveness of the ATPG process is the *abort limit*. The search for a pattern by the ATPG algorithm involves making a decision and certain assumptions, setting input values, and determining whether controllability and observability can be attained. When an assumption is proved false, or some restriction or blockage is encountered, the algorithm backs up, remakes the decision, and proceeds until the abort limit is reached, or a pattern is found to detect the fault. The value of the *abort limit* parameter can be set as in the following example:

```
TEST-T> set_atpg –abort_limit 20
```

By increasing the value of such a parameter, the process will require higher levels of effort. Determining an appropriate setting for the abort limit is an iterative process. The following example sequence shows how to play with this feature:

```
TEST-T> set_atpg -abort_limit 10 -merge off
TEST-T> run_atpg
TEST-T> set_atpg -abort 50
TEST-T> run_atpg
TEST-T> set_atpg -abort 250
TEST-T> run_atpg
```

As an exercise, you can try to improve the fault coverage obtained in the ATPG of the C6288, especially with many PI constraints and PO masks, by acting on the abort limit value.