

# Lab1

## Scan Chain Insertion and ATPG Using Design Compiler and TetraMAX

---

Pro: Chia-Tso Chao

TA: Ho-Jie Hsu

2024-12-09

# Server Usage (1/4)

---

## □ Policy

- Only 30GB storage is provided for each student
- New servers ee21~30 are encouraged to use
- ee31~35 is for TA, please don't use

# Server Usage (2/4)

---

## □ Tips

- Use “scheck” command to get all available server’s info
- Use “htop” command to get this server’s info
- Use “quota” command to get usage of storage

# Server Usage (3/4)

---

## □ Rules

- You would be **suspended right away** and **be in blacklist** if you:
  1. Borrow your account or borrow other's account
  2. Unusual usage of server, especially not related to course
  3. Destroy equipment or system on purpose
  4. Stealing other's account or invading the system

# Server Usage (4/4)

---

## □ Rules

- You would be **suspended for 2 weeks**, or **permanently** if you:
  1. Shut down or reboot server without permission
  2. Over-usage of computational resources
- Other serious violation would be punishment

# Log in to Work Stations

---

- Usage:

- \$ ssh [Account]@[Host name]

- Host name: [ee21~ee30].si2.iee.nycu.edu.tw

- Account: **vtlab001~vtlab033**

- Example (if your account is **vtlab000**):

- In terminal

- \$ ssh vtlab000@ee21.si2.iee.nycu.edu.tw

# Notification

---

- Port 415 is used for ssh connection to prevent network attack, only NYCU IP could be connected to server, e.g. 140.113.xxx.xxx

# Outline

---

- ☐ Introduction
- ☐ Design Compiler
- ☐ TetraMax
- ☐ Lab

# Outline

---

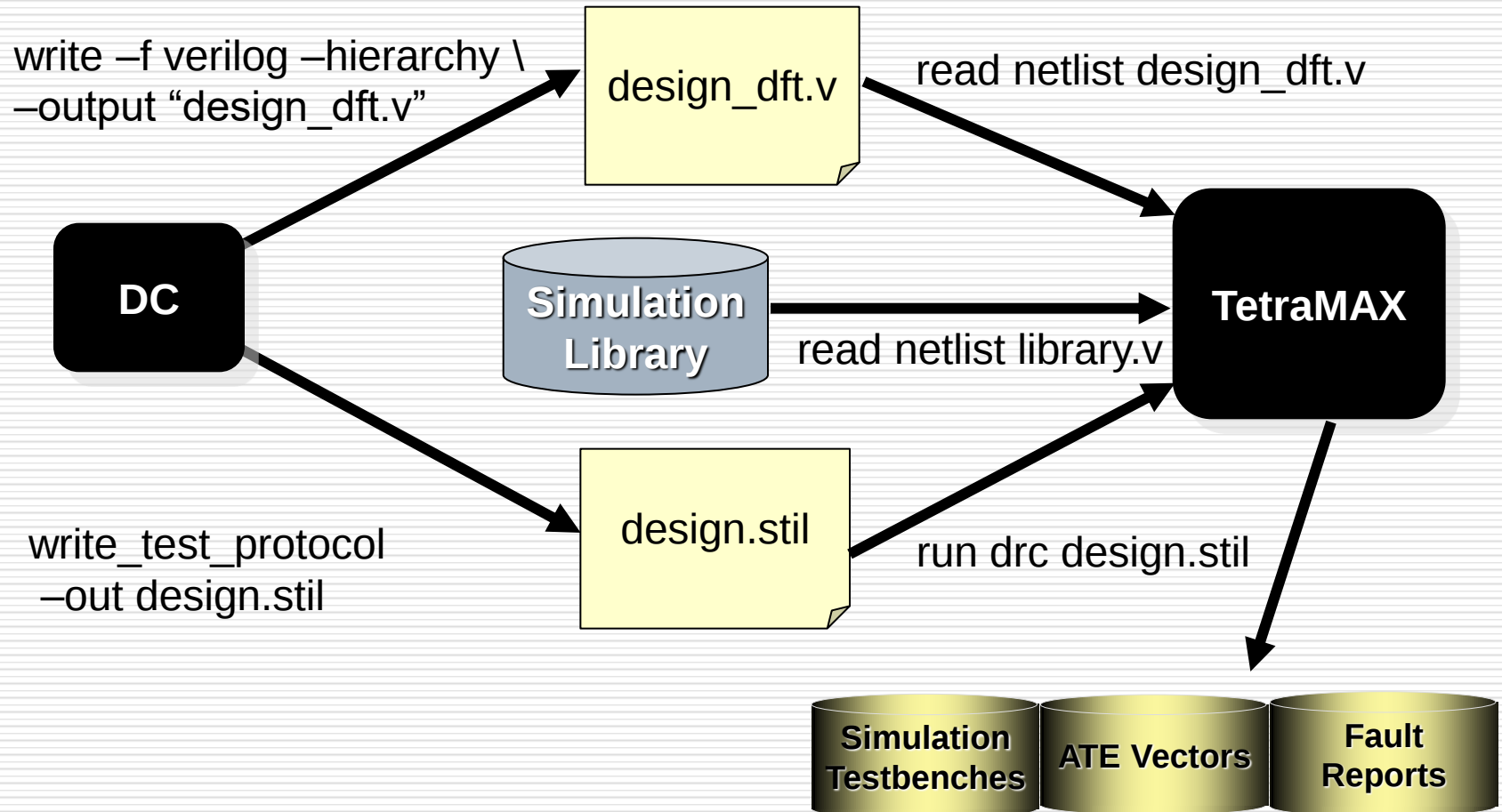
- Introduction
- Design Compiler
- TetraMax
- Lab

# Introduction

---

- ❑ This lab compares **impact on circuit after scan-chain insertion**.
- ❑ Items to be compared include area, power, test coverage and pattern count.
- ❑ **Synopsys Design Compiler** is the most common synthesis tool.
- ❑ **Synopsys TetraMax** is used to perform ATPG (Automatic Test Pattern Generation) and fault simulation.

# DFT compiler to TetraMAX



# Outline

---

- ☐ Introduction
- ☐ Design Compiler
- ☐ TetraMax
- ☐ Lab

# Invoke Design Compiler

---

- ❑ Extract files from TA's directory

```
$ tar -xvf ~vtlabTA01/lab.tar
```

- ❑ Change working directory

```
$ cd lab1
```

- ❑ Tool environment

Defined in .synopsys\_dc.setup

(it's a hidden file, so use command `$ ls -al` to find it)

- ❑ Invoke design compiler

```
$ dc_shell-t
```

# Read File, Link, Uniquify

---

- ❑ Read in RTL verilog source files  
`dc_shell> read_file -format verilog pre_norm.v`
- ❑ Show library details  
`dc_shell> list_libs`
- ❑ Specify the current module to synthesize  
`dc_shell> current_design pre_norm`
  - `pre_norm : top module`
- ❑ Link
  - Resolve the design reference based on reference names
  - Locate all design and library components, and connect them`dc_shell> link`
- ❑ Uniquify
  - Remove multiply-instantiated hierarchy in the current design by creating a unique module for each instance`dc_shell> uniquify`

# Wire Model, Scan Style, Clock

---

- ❑ Set up wire load model define in library
  - `dc_shell> set_wire_load_model -name wl10 -library l90sprvt_typ`
    - ❑ Use ``report_lib l90sprvt_typ`` to view library information
- ❑ Specify the scan style. Three styles are supported
  - Multiplexed flip-flop (multiplexed\_flip\_flop)
  - Clocked scan (clocked\_scan)
  - Level-sensitive scan design (lssd)
    - ❑ `dc_shell> set_scan_configuration -style multiplexed_flip_flop`
- ❑ Specify clock
  - `dc_shell> create_clock clk -period 10`
    - ❑ `clk` : the signal name defined in top module

# Compile(1/2)

---

- Use command "**compile**" to perform logic level and gate level synthesis and optimization on current design
  - **-map\_effort**  
specify the relative amount of CPU time spent during the mapping phase of "compile"

# Compile(2/2)

---

- **-scan**

Specify command to consider the impact of scan insertion on mission mode constraints during optimization. **This option causes the command to implement all flip-flops with scan flip-flops.**

- **Example:**

- `dc_shell> compile -scan -map_effort medium`

# Identify Scan-Chain Count, Generate Test Protocol (Method 1)

---

- ❑ Set scan-chain count considering the limitation of ATE or software, multiple clock domain, test time limitation  
`dc_shell> set_scan_configuration -chain_count 10`
- ❑ Define clocks in your design, then generate a test protocol
  - `-infer_clock`: infer *test* clocks in design
  - `-infer_async`: infer asynchronous set/reset signals in design`dc_shell> create_test_protocol -infer_clock -infer_async`

# Identify Scan-Chain Count, Generate Test Protocol (Method 2)

---

- ❑ If you want to specify some PI/POs to be normal inputs at operation mode and scan inputs during test mode use following commands
  - `dc_shell> set_scan_configuration -chain_count 1`
  - `dc_shell> set_dft_signal -port add -type scandatain`
  - `dc_shell> set_dft_signal -port sign -type scandataout`
  - `dc_shell> create_test_protocol -infer_clock -infer_asynch`

# Identify Scan-Chain Count, Generate Test Protocol (Method 3)

---

□ If you want to specify scan-chain order, use the following command

- `dc_shell> set_scan_configuration  
-chain_count 1`
- `dc_shell> set_scan_path ch1 -ordered_elements  
{ DFF_1 DFF_2 ... DFF_50 } -complete true`
- `dc>shell> create_test_protocol -infer_clock -  
infer_async`
- **-complete**  
indicate whether a specified scan chain is  
complete so that Design Compiler cannot add  
components to it

# Preview Design, Scan-Chain Synthesis

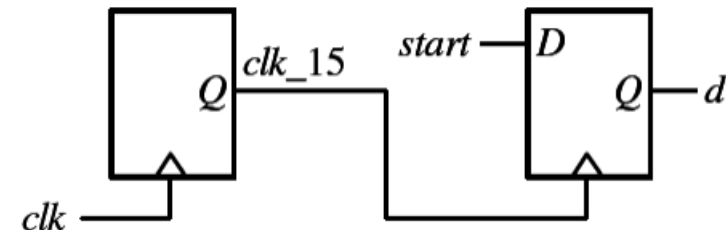
---

- ❑ Preview the scanned design for scan chain information
  - `dc_shell> preview_dft`
- ❑ Check test design rules according to the scan style chosen
  - `dc_shell> dft_drc`
- ❑ Insert scan chain
  - `dc_shell> insert_dft`

# If clock is gated (DRC violation)

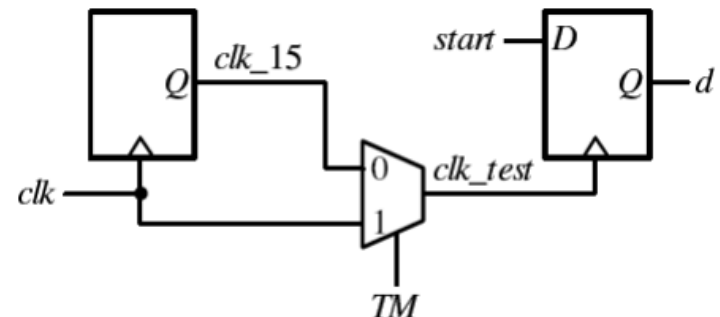
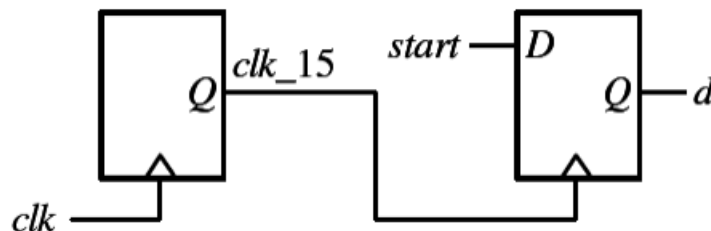
---

- -----
- DRC Report
- Total violations: 1
- -----
- 1 PRE-DFT VIOLATION
- 1 Uncontrollable clock input of flip-flop violation (D1)
- 
- Warning: Violations occurred during test design rule checking. (TEST-124)
- -----
- Sequential Cell Report
- 1 out of 71 sequential cells have violations
- -----
- SEQUENTIAL CELLS WITH VIOLATIONS
- \* 1 cell has test design rule violations
- SEQUENTIAL CELLS WITHOUT VIOLATIONS
- \* 70 cells are valid scan cells



# If clock is gated (DRC violation)

- ❑ Add additional signal TM (test mode) for testability
  - `dc_shell> create_port -direction "in" {TM}`
  - `dc_shell> set_dft_configuration -fix_clock enable`
  - `dc_shell> set_dft_signal -view exist -type ScanClock -timing {50 100} -port clk`
  - `dc_shell> set_dft_signal -view spec -type TestData -port clk`
  - `dc_shell> set_dft_signal -view spec -type TestMode -port TM`
  - `dc_shell> set_autofix_configuration -type clock -control TM -test_data clk`



# Report Area, Time, and Power

---

## □ Report area, timing, and power

- `dc_shell> report_area`
- `dc_shell> report_timing`
- `dc_shell> report_power`

# Result (1/2)

---

## □ Area

- Number of ports: 147
- Number of nets: 594
- Number of cells: 474
- Number of references: 52
  
- Combinational area: 2765.914043
- Noncombinational area: 1302.566048
- Net Interconnect area: 103180.518768
  
- Total cell area: 4068.480091
- Total area: 107248.998859

# Result (2/2)

---

## □ Power

- Global Operating Voltage = 1
- Power-specific unit information :
  - Voltage Units = 1V
  - Capacitance Units = 1.000000pf
  - Time Units = 1ns
  - Dynamic Power Units = 1mW (derived from V,C,T units)
  - Leakage Power Units = 1uW
  
- Cell Internal Power = 92.3638 uW (38%)
- Net Switching Power = 151.7164 uW (62%)
- -----
- Total Dynamic Power = 244.0803 uW (100%)
  
- Cell Leakage Power = 4.9543 uW

# Post Scan Check, Report Scan Path

---

- ❑ Recheck a design against the design rules of a chosen scan style
  - `dc_shell> dft_drc`
- ❑ Report the configuration of scan paths
  - `dc_shell> report_scan_path`

# Write Out Synthesized Verilog And STIL Files

---

## □ Save the scanned gate level netlist

- `dc_shell> write -hierarchy -format verilog -output pre_norm_scan.v`

## □ Save scan chain configuration

- `dc_shell> write_test_protocol -output pre_norm_scan.stil`

- `dc_shell> write_sdc pre_norm_scan.sdc`

- `dc_shell> exit`

# Outline

---

- ☐ Introduction
- ☐ Design Compiler
- ☐ TetraMax
- ☐ Lab

# Invoke TetraMax

---

- ❑ In tsch, invoke TetraMax  
\$ tmax64 -s
- ❑ Initial state is "BUILD"  
BUILD>

# Read Netlist and Library

---

- Read verilog netlist file

```
BUILD> read_netlist pre_norm_scan.v
```

- Read library file

```
BUILD> read_netlist l90sprvt.v -library
```

# Reporting Modules

---

- -summary

Generate a summary report on all modules

- -error

Report all modules that have at least one violation of a rule with **severity** of "error"

- -undefined

report all modules that are referenced but not defined

- Example:

- BUILD> report\_modules -summary
- BUILD> report\_modules -error
- BUILD> report\_modules -undefined

# Building ATPG Design Model

---

- Builds the in-memory **simulation model** from the design modules that have been read in
  - **BUILD> run\_build\_model pre\_norm**
    - It will change into **DRC command** mode

# Set DRC Parameters And Run

---

- ❑ Set the parameters that control DRC process. You can display the current settings with "**report\_settings**" command
- ❑ Perform Design Rule Checking, which is **required** to enter the **TEST command mode**, where test generation and fault simulation can be performed
  - **DRC> run\_drc pre\_norm\_scan.stil**
    - ❑ pre\_norm\_scan.stil : scan chain configuration file

# Add Faults

---

- ❑ Select the fault model for ATPG
  - `TEST > set_faults -model stuck`
  - TetraMAX supports test pattern generation for five fault models
    - ❑ Stuck-at
    - ❑ Transition
    - ❑ Path delay
    - ❑ IDDQ
    - ❑ Bridging
- ❑ Create a list of faults for fault simulation and test generation
  - `TEST > add_faults -all`

# ATPG (1/3)

---

- ❑ Set the parameters that control the ATPG processes
  - **-merge**  
Specify whether to perform pattern merging during ATPG. The arguments indicates **how much effort to spend doing merging (default: none)**
  - **-verbose**  
With -verbose enabled, extra messages are displayed during the pattern merge operation
  - **-abort\_limit**  
Specify the max number of remade decisions before terminating a test generation effort during ATPG (**default: 10**)

# ATPG (2/3)

---

- **-coverage**  
Specify a test coverage limit at which to terminate the ATPG effort. Ranging from 0~100 (**default: 100**)
- **-decision**  
When backtracking, using specific way to determine (**default: norandom**)
- **-time**  
Specify the maximum CPU time, in seconds, allowed per fault or per run. The time limit can be turned off again by specifying a 0 for the time values.
- **Note for Tcl mode:**  
Multiple values specified by the -time option must appear as a list and be enclosed by braces "{}".

# ATPG (3/3)

---

- **-full\_seq\_time**

Similar to “-time” option, but applies to the Full-Sequential ATPG algorithm. (default: 0)

- Example for scan chain design:

- TEST> set\_atpg -merge high -verbose -abort\_limit 250  
-coverage 100 -decision random  
TEST> run\_atpg

- Example for non-scanned design:

- TEST> set\_atpg -merge high -verbose  
-full\_seq\_time {3600 86400} -full\_seq\_atpg  
TEST> run\_atpg

# Understanding ATPG Output

```
TEST> run_atpg
# ATPG performed for stuck fault model using internal pattern source.
# -----
# #patterns  #patterns      #faults      #ATPG faults  test      process
# simulated  eff/total    detect/active  red/au/abort  coverage  CPU time
# -----
# Begin deterministic ATPG: #collapsed_faults=3803, abort_limit=100...
# 32          29      29      2671    1132          0/0/4      73.68%      0.02
# 64          31      60       578     552          2/0/14     87.16%      0.07
# 96          29      89       152     379          23/0/90    91.14%      0.30
# ...
```

- ❑ ATPG progress is reported pass by pass (32 simulated patterns)
- ❑ #patterns eff: number of patterns that can detect faults
- ❑ #patterns total: cumulative #patterns eff
- ❑ #faults detect: number of faults that are detected in a pass
- ❑ #faults active: number of faults in target fault list
- ❑ #ATPG faults red/au/abort:
  - Cumulative number of redundant/ATPG untestable/aborted faults

# Fault Summary Report

## Collapsed Stuck Fault Summary Report

| fault class       | code | #faults |
|-------------------|------|---------|
| Detected          | DT   | 4132    |
| Possibly detected | PT   | 0       |
| Undetectable      | UD   | 43      |
| ATPG untestable   | AU   | 0       |
| Not detected      | ND   | 126     |
| -----             |      |         |
| total faults      |      | 4301    |
| test coverage     |      | 97.04%  |

## Pattern Summary Report

|                      |     |
|----------------------|-----|
| #internal patterns   | 212 |
| #basic_scan patterns | 212 |

## □ ATPG metrics

- Total fault number
- Test coverage
- Pattern count
- CPU time  
(Not shown in summary report)

# Fault Class

---

- Detected (DT)
  - Guarantee a detectable difference between the expected value and the fault effect value
- Possibly Detected (PT)
  - A faulty machine response will simulate an "X" rather than a 1 or 0
- Undetectable (UD)
  - Cannot be tested by any means
- ATPG Untestable (AU)
  - Cannot be found using ATPG, but may be detected by other methods(functional tests)
- Not Detected (ND)
  - Cannot be found due to **ATPG iterations limits** or **designs too complex**

# Test Coverage

---

$$\text{Test Coverage} = \frac{DT + (PT * \text{posdet\_credit})}{\text{all faults} - (UD + (AU * \text{au\_credit}))}$$

- Posdet\_credit: default 50%
- Au\_credit: default 0%

# Reporting Faults

---

- ❑ Sets the parameters that control the fault manager
  - `TEST > set_faults -summary verbose`
- ❑ Set which kind of faults you want to see collapsed/uncollapsed
  - `TEST > set_faults -report collapsed`
  - `TEST > report_summaries`
- ❑ Display fault data
  - `"-class"`: Specifies a specific fault class to be reported
    - ❑ `TEST > report_faults -class UD`

# Writing Faults & Patterns

---

- Writes fault data to external file
  - TEST > write\_faults pre\_norm\_faults.rpt  
-all -replace
- Writes patterns to external file
  - TEST > write\_patterns  
pre\_norm\_test\_patterns.stil -format stil  
-replace
  - TEST > exit

# Outline

---

- ☐ Introduction
- ☐ Design Compiler
- ☐ TetraMax
- ☐ Lab

# Goal

---

- Compare the following for circuit with and without inserting scan chain
  - Area
  - Power
  - Fault count
  - Pattern count
  - Test coverage
  - ATPG runtime

# Notice – DC (w/o Scan-chain)

---

- ❑ For circuit without scan-chain, don't set any command related to scan in design compiler, including: `compile -scan`, `preview_dft`, `insert_dft`, `set_scan_configuration`, `report_scan_path`, `create_test_protocol`, `write_test_protocol`, `write_scan_def`

# Notice – Tmax (w/o Scan-chain)

---

- ❑ For circuit without scan-chain running ATPG, only use the following command: (page 34)
  - `DRC> run_drc`
- ❑ For ATPG without scan-chain, please remember to add an ATPG constraint.
  - `TEST> set_atpg -full_seq_time {600 36000}`
- ❑ For circuit without scan-chain doing ATPG, use option: `-full_seq_atpg`
  - `TEST> set_atpg -full_seq_atpg`

# Result

---

| pre_norm                               | Area   | Power | Fault count | Coverage<br>(collapsed) | ATPG<br>Run<br>Time(s) | Pattern |
|----------------------------------------|--------|-------|-------------|-------------------------|------------------------|---------|
| Non-<br>Scanned<br>(-full seq<br>atpg) | 98072  | 260uw | 4011        | 97.88%                  | 9689.6                 | 264     |
| Scanned                                | 105072 | 271uw | 4415        | 99.34%                  | 1.57                   | 159     |

# Homework

---

- Run `pre_norm.v` and `s38584_seq.v`
  - Generate a result table like last slide.
-

# Reference

---

- ❑ [1] SynopsysInc., “Design Compiler User Guide”, Dec. 2004.
- ❑ [2] SynopsysInc., “Design Compiler Command-Line Interface Guide”
- ❑ [3] SynopsysInc., “Design CompilerReference Manual”
- ❑ [4] IPCORE Lab Slide 2006, Tian-Sheuan Chang
- ❑ [5] VLSI Testing Course Slide, Jing-Jia Liou
- ❑ [6] CIC Training Center Slide, Hsin-Jung Huang