Verification Training plan

# Lý thuyết (1 tuần)

* Verification concept (<http://testbench.in/>)
  + **ASIC DESIGN**
  + [**FUNCTIONAL VERIFICATION NEED**](http://testbench.in/TS_03_FUNCTIONAL_VERIFICATION_NEED.html)
  + [**TESTBENCH**](http://testbench.in/TS_04_TESTBENCH.html)
  + [**LINEAR TESTBENCH**](http://testbench.in/TS_05_LINEAR_TESTBENCH.html)
  + [**LINEAR RANDOM TESTBENCH**](http://testbench.in/TS_06_LINEAR_RANDOM_TESTBENCH.html)
  + [**HOW TO CHECK THE RESULTS**](http://testbench.in/TS_07_HOW_TO_CHECK_THE_RESULTS.html)
  + [**SELF CHECKING TESTBENCHS**](http://testbench.in/TS_08_SELF_CHECKING_TESTBENCHS.html)
  + [**HOW TO GET SCENARIOS WHICH WE NEVER THOUGHT**](http://testbench.in/TS_09_HOW_TO_GET_SCENARIOS_WHICH_WE_NEVER_THOUGHT.html)
  + [**HOW TO CHECK WHETHER THE TESTBENCH HAS SATISFACTORILY EXERCISED THE DESIGN**](http://testbench.in/TS_10_HOW_TO_CHECK_WHETHER_THE_TESTBENCH_HAS_SATISFACTORILY_EXERCISED_THE_DESIGN.html)
  + [**TYPES OF CODE COVERAGE**](http://testbench.in/TS_11_TYPES_OF_CODE_COVERAGE.html)
  + [**STATEMENT COVERAGE**](http://testbench.in/TS_12_STATEMENT_COVERAGE.html)
  + [**BLOCK COVERAGE**](http://testbench.in/TS_13_BLOCK_COVERAGE.html)
  + [**CONDITIONAL COVERAGE**](http://testbench.in/TS_14_CONDITIONAL_COVERAGE.html)
  + [**BRANCH COVERAGE**](http://testbench.in/TS_15_BRANCH_COVERAGE.html)
  + [**PATH COVERAGE**](http://testbench.in/TS_16_PATH_COVERAGE.html)
  + [**TOGGLE COVERAGE**](http://testbench.in/TS_17_TOGGLE_COVERAGE.html)
  + [**FSM COVERAGE**](http://testbench.in/TS_18_FSM_COVERAGE.html)
  + [**MAKE YOUR GOAL 100 PERCENT CODE COVERAGE NOTHING LESS**](http://testbench.in/TS_19_MAKE_YOUR_GOAL_100_PERCENT_CODE_COVERAGE_NOTHING_LESS.html)
  + [**FUNCTIONAL COVERAGE**](http://testbench.in/TS_20_FUNCTIONAL_COVERAGE.html)
  + [**COVERAGE DRIVEN CONSTRAINT RANDOM VERIFICATION ARCHITECTURE**](http://testbench.in/TS_21_COVERAGE_DRIVEN_CONSTRAINT_RANDOM_VERIFICATION_ARCHITECTURE.html)
  + [**PHASES OF VERIFICATION**](http://testbench.in/TS_22_PHASES_OF_VERIFICATION.html) **(hết tuần này – t2 26/09 review)**
  + [**ONES COUNTER EXAMPLE**](http://testbench.in/TS_23_ONES_COUNTER_EXAMPLE.html)
  + [**VERIFICATION PLAN**](http://testbench.in/TS_24_VERIFICATION_PLAN.html)
* OOP basic

# Bài tập (1 tuần)

* Test rtl QSPI core, yêu cầu:
  + Viết testplan trước khi bắt đầu code
  + Có function coverage 100%, code coverage 100%
  + Có viết assertion check (bắt buộc), monitor check (có thể viết hoặc không)
  + Chia các test case cụ thể
  + Tạo report (html)
  + Chạy mô phỏng chạy bằng script (command lines)
  + Tổ chức thư mục: rtl, testbench, model để riêng từng thư mục. các file trong testbench chia cụ thể rành mạch theo từng chức năng, không viết chung thành 1 file (stimulus, driver, top testbench, assertion, coverage…)

# Tài liệu tham khảo:

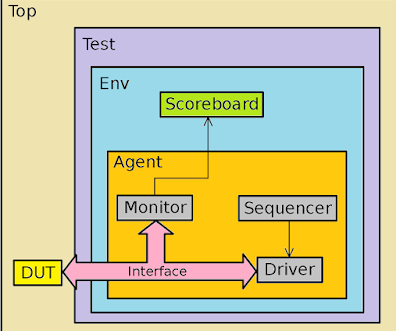
* + <https://verificationguide.com/systemverilog/systemverilog-tutorial/>
  + <http://testbench.in/TS_00_INDEX.html>
  + Sách Systemverilog for Verification
  + <https://nguyenquanicd.blogspot.com/2019/11/uvm-bai-8-tao-bao-cao-coverage-va-phan.html>
  + https://www.chipverify.com/systemverilog/systemverilog-testbench-example-1

1. Lý thuyết

* Một thiết kế ASIC mất khoảng 50-70% thời gian chỉ để verification
* Linear TestBench thực hiện các thao tác sau :
  + - Khởi tạo thiết kế đang được kiểm tra (DUT)
    - Kích thích DUT bằng cách áp dụng các vector kiểm tra
    - Xuất kết quả của sổ dạng song hoặc đến một thiết bị đầu cuối để kiểm tra trực quan bằng tay
* Linear Random TestBench giống với Line TestBench nhưng giá trị kích thích đầu vào DUT sử dụng hàm $random để tạo giá trị ngẫu nhiên 32bit( có thể lặp lại)
* Check result : là việc kỹ sư thẩm định kết quả từ quan sát dạng song và tín hiệu được gửi đến thiết bị đầu cuối
* Selt check result : hệ thống sẽ chỉ ra nơi hệ thống lỗi
* Giải pháp cho vấn đề lỗi tiềm ẩn là xác minh ngẫu nhiên ràng buộc. Kích thích sẽ được tạo tự động, kỹ sư chỉ cần thông số kỹ thuật và Testbench sẽ tự tạo không gian giải pháp và chọn các kịch bản từ không gian giải pháp
* Xác định mức độ mã nguồn của DUT đã được kiểm tra( structural coverage) bằng cách kiểm tra:
  + Tất cả các lệnh rẽ nhánh if else hoặc case
  + Các biến đã có sự thay đổi chưa
  + Tất cả các lệnh RTL đã thực thi chưa
  + Chức năng của FSM đã thực hiện đúng chưa
  + Các đừng dẫn trong khối đã được liên kết chưa
* Coverage là mức độ bao phủ, code coverage là các thông tin coverage có thể được trích xuất tự động từ RTL code của thiết kế. function coverage là loại coverage được định nghĩa bởi người kiểm tra gắn liền với mục đích và chức năng của thiết kế.
  + Code coverage gồm:
    - Statement coverage : là mức độ bao phủ của các phát biểu , xác định 1 phát biểu trong code đã thực thi chưa
    - Block coverage : nhóm các câu lệnh begin end , if else , vòng lặp .. thành 1 khối. xem khối đó đã được thực thi chưa
    - Branch coverage : là mức độ bao phủ của các lệnh rẽ nhánh, độ bao phủ đạt 100% khi tất cả các nhánh đều đã đc thực thi
    - Condition converage : là mức độ bao phủ của các điều kiện, dùng để đánh giá, phân tích các điều kiện có trong các lệnh rẽ nhánh và toán tử điều kiện . Độ bao phủ 100% khi tất cả các tổ hợp giá trị của biến trong biểu thức đã xuất hiện
    - Topple coverage : là mức độ bao phủ của sự thay đổi trạng thái. Độ bao phủ đạt 100% khi có sự thay đổi logic 0,1,z
    - FSM coverage : là độ bao phủ của FSM khi tất cả các trạng thái đều đc sử dụng
  + Function coverage :
    - Người kiểm tra phải định nghĩa điểm bao phủ
    - Điểm bao phủ dựa trên spec và đặc điểm mong muốn
    - Có thể đánh giá tính đẩy đủ và đúng đắn của thiết kế
    - Systemverilog hỗ trợ covergroup, coverpoint, cross, task, funsction để mỗ tả function coverage
* Code coverage cần đạt 100% để tránh gặp sai sót trong thiết kế. Khi kĩ sư thực hiện được tất cả các trường hợp có thể xảy ra thì khi đó việc xác minh đã hoàn tất . Test done = 100% code coverage + 100% function coverage

UVM TestBench Architechture:

Phương pháp xác minh tổng quát – Universal Verification methodology (UVM) là một phương pháp xác minh thiết kế



Các thành phần của UVM :

* Evn : chứa các thành phần tác nhân hoặc scoreboard
* Scoreboard : nhận các dữ liệu từ màn hình và so sáng chúng với giá trị mong muốn
* Agent :

SVA- SystemVerilog Assertion :

là một cấu trúc ngôn ngữ cung cấp 1 cách thay thế để viết các dàng buộc, bộ kiểm tra và các điểm che khuất cho thiết kế

phân loại assertion:

* Immediate assertion : giống như 1 assert if else đơn giản, không phụ thuộc vào clock và reset.

Immediate\_assertion\_name :

Assert (Q !=0)

Else

$error(“%m checker failed”);

Mã trên giống với :

Always\_comb begin

If(assertion statement)

// do nothing

Else

%error(“fail”);

End

* Concurrent assertion : cho phép mô tả các biểu thức phức tạp hơn và phụ thuộc vào clock và reset. Từ khóa property để phân biệt 2 loại assertion

// If inputs vld=1 and dat=8'h55, then ack is high 3 cycles later.

// ack sẽ bằng 1 sau 3 chu kì kể từ khi điều kiện trong ngoặc đúng

valid\_gnt\_chk: assert property (

@posdege (clk) disable iff (rst)

(vld && dat == 8'h55) |-> ##3 ack);

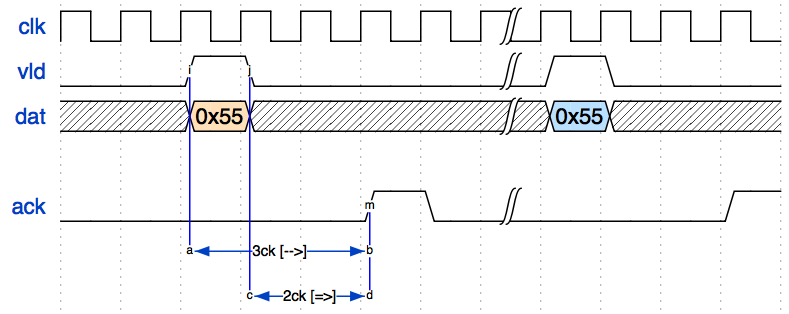
// ack sẽ bằng 1 sau n-1 chu kì kể từ khi điều kiện trong ngoặc đúng

valid\_gnt\_chk: assert property (

@posdege (clk) disable iff (rst)

(vld && dat == 8'h55) |=> ##2 ack);

// “ |-> ##1 “ = “ |=>” , dissble iff thiết lập tín hiệu không đồng bộ - thường là reset



Một số hàm sử dụng cho SVA:

* $rose : trả về true nếu bit LSB của biểu thức thay đổi thành 1 và ngược lại
* $fell : trả về true nếu bit LSB của biểu thức thay đổi thành 0 và ngược lại
* $stable : trả về true nếu giá trị của biểu thức không thay đổi và ngược lại
* $past(expression , num\_cycles) : trả về giá trị của biểu thức từ trước num\_cycles
* $countones: trả vè số 1 trong 1 biểu thức
* $onehot: trả vè true nếu chính xác 1 bit , neesuko có bit nào hoặc nhiều hơn 1 bit 1 thì là false
* $onehot0: trả về true là không có bit nào hoặc chỉ 1 bit 1 trong biểu thức

Một số toán tử sử dụng trong SVA :

* ##n ; ##[m:n] : toán tử độ trẽ cố định; khoảng thời gian tương tác
* |->: hàm chồng chéo
* |=> : hàm không trùng lặp
* [\*n];[\*m:n] : toán tử lặp liên tục; lặp liên tục trong phạm vi chu kì đc chỉ định
* [->n];[->m:n]: cho biết có 1 hay nhiều chu kì trẽ giữa một lần lặp lại của biểu thức

Các lệnh dùng trong verify

* Rand : cách để khai báo 1 biến để dung lệnh randum giá trị. Rand bit [1:0] cnt ;
* Mailbox : giao tiếp các thư giữa các tiến trình.
  + Có 2 loại mailbox là
    - Chung : có thể gửi và nhận bất kì loại nào . mailbox mailbox\_name
    - không giới hạn : chỉ gửi và nhận loại dữ liệu cụ thể. Mailbox#(type) mail\_name
  + 1 số lệnh thao tác với mailbox
    - New() tạo 1 mailbox , trans = new() ;
    - Put() nơi lưu trữ mailbox , gen2driv.put(trans);
    - Num() , trả về giá trị tin nhắn có trong mailbox
* Event : là 1 kiểu đối tượng tĩnh để đồng bộ hóa giữa các tiến trình hoạt động đồng thời. Để kích hoạt các sự kiện đã đc khai báo ta dùng -> hoặc ->> , các tiến trình có thể đợi 1 sự kiện bằng cách dung @ hoặc .triggered