SYSTEMVERILOG FOR VERIFICATION

Phần 1: Cấu trúc cơ bản của Verify

1. Data types
2. Queues

* Là 1 kiểu dữ liệu giống hàng đợi

1. Function coverage

* Covergroup là một kiểu do người dùng định nghĩa ,cho phép lấy mẫu tất cả variable, transition,cross tại cùng 1 thời điểm cạnh clock hoặc lấy mẫu
  + Có thể khai báo convergroup trong packet, module, program, interface, class
  + Được sử dụng nhiều trong test random
* Các thành phần bên trong covergroup:
  + Clock đồng bộ hóa việc lấy mẫu các điểm cần cover
  + Tập hợp các coverpoint
  + Phạm vi cover của các coverpoint
  + Các cross coverage giữa các coverpoint hoặc các biến
  + Các đối số tùy chọn
* Coverpoint là 1 giá trị hoặc biểu thức có chức năng bao hàm thông số kỹ thuật
  + Mỗi Coverpoint bao gồm các bins được liên kết với giá trị được lấy mẫu của nó hoặc chuyển đổi giá trị
* Sử dụng covergroup để có thể kiểm soát xem các giá trị đầu vào đã randomize có được bao phủ hết các giá trị cần bao phủ chưa, khác với việc randomize giá trị đầu vào là tạo ra giá trị đầu vào

Ví dụ:

g1 là tên của covergroup

oc , ac là nhãn của coverpoint

offset , addr là giá trị của coverpoint

tại sườn lên của clock thì coverpoint được lấy mẫu, nếu xung clock bị bỏ qua thì ta phải kích hoạt thủ công bằng hàm sample() …..

bit [1:0] offset ;

logic [7:0] addr ;

covergroup g1 @(posedge clk);

oc : coverpoint offset ;

ac: coverpoint addr ;

endgroup

g1 g1\_inst = new ;

* Bins : khởi tạo giá trị cho từng biến trong phạm vi có thể có của một biến coverage . Bins binname = { range in coverpoint varible};

Ví dụ : <https://www.youtube.com/watch?v=ogKqZrwBv6Q>

Cp\_b : coverpoint b{

Bins b0={0,2,3};

Bins b1[]={[2:8]};

}

Bins b1 ={0,2,7}; bin b1 thì cover addr nhận giá trị từ 0,2 hoặc 7

Bins b1[]= { [2:9] };mỗi bins b1 sẽ có 1 báo cáo riêng của từng phần tử [2:9]

Bins b1[3] = { [2:9] }; chia bin b1 thành 3 đoạn báo cáo riêng trong khoảng [2:9]

Bins b1 = {10=>20=>30}; bin b1 thì cover addr sẽ chuyển giá trị 20, 30 sau mỗi lần clock

Ignore\_bins b1={6,7,10}; các điểm cover của addr sẽ bị loại bỏ, không cần cover

Illegal\_bins b1 = { 4,2,7}; các điểm cover của addr đánh dấu là bất hợp pháp

* Option.member\_name = expression : định nghĩa các tùy chọn bên trong covergroup , có thể là 1 biến hoặc tất cả phạm vi của covergroup

Các member\_name được định nghĩa trong sách trang 578, table 19-1 systemverilog\_hardware\_design\_specific

Vd : option.at\_least=2; số lần truy cập tối thiểu cho mỗi biến khai báo bins

1. Processes

* Fork – foin : chạy các chương trình theo cách song song
* Begin- end : chạy các trường trình theo cách lần lượt
* Fork – join : bắt đầu chạy tất cả các processes song song có bên trong nó và chờ đến khi tất cả đã chạy xong thì thoát chặn
* Fork – join\_any : sẽ bỏ thoát khỏi chặn ngay khi có 1 process hoàn thành
* Fork – join\_none : không có chặn
* Wait fork : đợi cho đến khi hoàn thành fork
* Mailbox : trao đổi dữ liệu giữa các tiến trình
  + Lệnh name\_mailbox = new(3) : khởi tạo 1 mailbox có chứa 3 giá trị
  + Lệnh name\_mailbox.put() : để đẩy giá trị vào mailbox
  + Lệnh name\_mailox.get() : để lấy giá trị bên trong maibox để sử dụng

1. Class

* New() : hàm khởi tạo, để khởi tạo các biến bên trong lớp với 1 giá trị. Nếu hàm new() chưa được truyền giá trị thì mặc định là 0
* Static : khai báo 1 biến là static thì biến đó sẽ là duy nhất ko có bản sao trong tất cả các thành phần khác có trong class :
  + truy cập biến static bên ngoài class bằng cách dùng toán tử :: kể cả khi ko có class khởi tạo
* Class assignment – chuyển nhượng : cả 2 con trỏ cùng trỏ vào cùng 1 vị trí bộ nhớ, thay đổi pkt2 thì pkt1 cũng thay đổi

Vd :

packet pkt­1;

packet pkt2;

pkt1 = new(10,20);

pkt2 = pkt1 ;

* Shallow copy – sao chép nông : pkt2 chỉ sao chép giá trị của pkt1, pkt2 thay đổi thì pkt1 ko thay đổi

Vd :

packet pkt­1;

packet pkt2;

pkt1 = new(10,20);

pkt2 = new pkt1 ;

* Deep copy – sao chép sâu : giống như 1 bản sao chép đầy đủ của các đối tượng nhưng sẽ là sao chép vào vị trí bộ nhớ khác

Vd:

* Tính kế thừa : xây dựng lớp mới trên các lớp đã có, lớp mới kế thừa thuộc tính và các phương thức , quyền truy cập

Vd : class driver extends scoreboard

* Lớp dẫn xuất là class được xây dựng thừa kế 1 class khác
* Virtual – lớp ảo : không được khởi tạo trực tiếp mà chỉ được kế thừa, dùng cho việc kết nối giữa thành phần phần cứng vs phần mềm

Vd : <https://www.vlsiforu.com/abstract-class-or-virtual-class-in-system-verilog/>

Ta thấy rằng virtual class basepacket chỉ đc khai báo ngoài và được khởi tạo khi là lớp dẫn xuất của các class khác

1. Randomization and constraints
2. Randomization

* Là việc tạo giá trị ngẫu nhiên với từ khóa rand và randomize()
* Rand : giá trị đc phân bố đồng đều trên phạm vi
* Randc: giá trị đc ngẫu nhiên theo chu kì, giá trị ko lặp lại cho đến khi tất cả các giá trị đã đc gán
* Randomize() : dùng để ngẫu nhiên hóa đối tượng vd: object.randomize();
* Rand\_mode(0) : 0 -> vô hiệu hóa random , 1-> bật random vd : pkt.addr.rand\_mode(0); // vô hiệu hóa random cho biến addr của pkt.

1. Constraint

* Các giá trị ngẫu nhiên sẽ bị ràng buộc
* Constrain <constraint\_block\_name> { addr expression;} : addr nhận giá trị ngẫu nhiên có ràng buộc với từng expression:
  + - Inside { [a:b] }: biến addr nhận giá trị ngẫu nhiên trong khoảng [a:b], có thể gán phạm vị bằng giá trị, khoảng,hoặc ngoài khoảng “!”
    - Addr > 5: biến addr nhận giá trị ngẫu nhiên lớn hơn 5
    - Ràng buộc hàm ý : expression -> constraint: nếu expression đúng thì addr nhận giá trị theo constrain, có thể dùng if else để cho trường hợp khác expression
* Name dist { value := weight or value :/ weight }; : weight là trọng số cho sự xuất hiện của value “:=” trọng số cho mỗi giá trị còn “/=” là cho toàn bộ giá trị

Vd : <https://www.theartofverification.com/weighted-distribution-in-system-verilog/>

* Foreach constraint : ràng buộc giá trị ngẫu nhiên cho mảng

Vd : <https://verificationguide.com/systemverilog/systemverilog-foreach-loop-constraint-blocks/>

* Constraint\_mode(0): vô hiệu hóa constraint, muốn xem trạng thái của constraint thì gọi hàm pkt.constraint\_mode(); giá trị trả vê là 0 or 1
* Static constraint: giống như static class
  + Sự thay đổi chế độ ràng buộc của 1 biến sẽ làm thay đổi tất cả các biến còn lại của class

Vd : <https://verificationguide.com/systemverilog/systemverilog-static-constraints/>

Khi tắt ràng buộc của pkt1.addr nhưng pkt2.addr cũng bị tắt theo thì biến addr là static constraint sẽ làm thay đổi thuộc tính của addr bên trong class

1. 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);

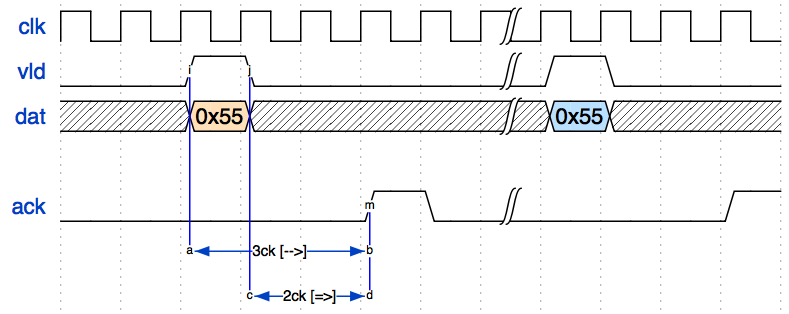
// ack sẽ bằng 1 sau n-1 chu kì kể từ khi điều kiện trong ngoặc đúng

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:

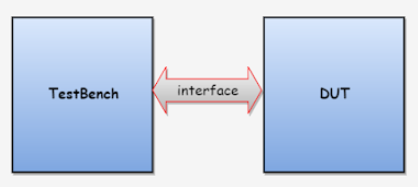
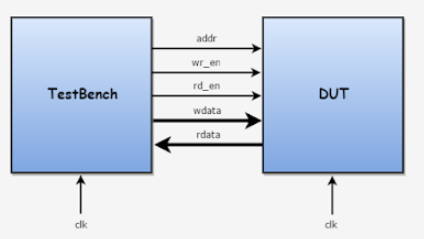
* $rose(expression) : trả về true nếu bit LSB của biểu thức thay đổi thành 1 và ngược lại
* $fell(expression) : trả về true nếu bit LSB của biểu thức thay đổi thành 0 và ngược lại
* $stable(expression) : trả về true nếu giá trị của biểu thức không thay đổi và ngược lại
* $past(expression , num\_cycles) : trả về giá trị của biểu thức từ trước num\_cycles
* $countones(expression): trả vè số bit 1 trong 1 biểu thức
* $onehot(expression): trả về true nếu chỉ có 1 bit 1 , có nhiều hơn 1 bit 1 thì là false
* $onehot0(expression): trả về true nếu tất cả các bit là 0 , nếu là 1 thì là false
* $isunknown(expression) : trả về true nếu có bất kì bit x hoặc z nào , nếu ko là false
* $Countones(expression) : đếm số lượng bit 1
* Còn vài hàm nữa .

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

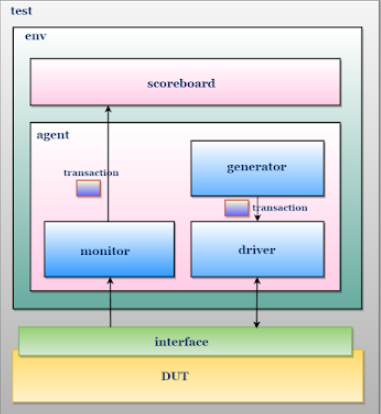
1. Interface

* Interface dùng để đóng gói các tín hiệu giao tiếp thành 1 giao diện.
* Modport : định nghĩa kiểu in/out cho tín hiệu



1. Cấu trúc verify

* Transaction:
* Generator: khởi tạo giá trị( ngẫu nhiên hoặc constrain) gửi đến driver
* Driver: nhận giá trị khởi tạo và truyền đến DUT
* Monitor: quan sát hoạt động của các tín hiệu và chuyển dữ liệu đến scoreboard
* Agent: chứa các class generator, driver, monitor
* Scoreboard: nhận các dữ liệu từ monitor và so sánh với các giá trị mong muốn
* Environment: là nơi để chứa các thành phần cấp cao như agent và scoreboard
* Interface: tạo liên kết giữa các thành phần
* Test:



1. Kiến thức ngoài lề

* Để hiểu hơn về các khối có trong VIP <https://www.chipverify.com/uvm/uvm-tutorial>
* Cấu trúc của ngôn ngữ perl sử dụng để tự động hóa công việc cho text còn bash-script là dạng ngôn ngữ thông dịch cũng là tự động hóa công việc cho các tiến trình

Phần 2: thư viện UVM

* Các thành phần của uvm: