TÓM TẮT BÀI GIẢNG **VERILOG**

CHƯƠNG I - TỔNG QUAN

I. Giới thiệu

Verilog HDL là một trong hai ngôn ngữ mô phỏng phần cứng thông dụng nhất, được dùng trong thiết kế IC, ngôn ngữ kia là VHDL. HDL cho phép mô phỏng các thiết kế dễ dàng, sửa chữa lỗi, hoặc thực nghiệm bằng những cấu trúc khác nhau. Các thiết kế được mô tả trong HDL là những kỹ thuật độc lập, dễ thiết kế, dễ tháo gỡ, và thường dễ đọc hơn ở dạng biểu đồ, đặc biệt là ở các mạch điện lớn.

Verilog thường được dùng để mô tả thiết kế ở bốn dạng:

Thuật toán (một số lệnh giống ngôn ngữ C như: if, case, for,while…).
Chuyển đổi thanh ghi (kết nối bằng các biểu thức Boolean).
Các cổng kết nối(cổng: OR, AND, NOT…).
Chuyển mạch (BJT, MOSFET)
Ngôn ngữ này cũng chỉ rõ cách thức kết nối, điều khiển vào/ra trong mô phỏng.

Cấu trúc chương trình dùng ngôn ngữ Verilog

// Khai báo module
Module tên chương trình (tên biến I/O); // tên chương trình trùng tên file.v.
Input [msb:lsb] biến;
Output [msb:lsb] biến;

Reg [msb:lsb] biến reg;
Wire [msb: lsb] biến wire;
// Khai báo khối always, hoặc khối initial.
… các lệnh …

Endmodule

II. Ý nghĩa các thuật ngữ trong VERILOG

Các tập tin văn bản nguồn Verilog bao gồm những biểu hiện thuộc tính từ vựng sau đây:

1. Khoảng trắng

Khoảng trắng ngăn những từ và có thể chứa khoảng cách, khoảng dài, dòng mới và dạng đường dẫn. Do đó, một lệnh có thể đưa ra nhiều dòng phức tạp hơn mà không có những đặc tính đặc biệt.

2. Chú giải

Những chú giải có thể chỉ định bằng hai cách: (giống trong C/C++).

Chú giải được viết sau hai dấu gạch xiên (//). Được viết trên cùng một dòng.

Được viết giữa /* */, khi viết nhiều dòng chú giải.

3. Chữ số

Lưu trữ số được định nghĩa như là một con số của các bit, giá trị có thể là: số nhị phân, bát phân, thập phân, hoặc thập lục phân.

Ví dụ: 3’b001, 5’d30 = 5’b11110,
16’h5ED4 = 16’d24276 = 16’b0101111011010100

4. Từ định danh

Từ định danh do người dùng quy định cho biến số, tên hàm, tên môđun, tên khối và tên trường hợp. Từ định danh bắt đầu bằng một mẫu tự hoặc đường gạch dưới ’_’ (không bắt đầu bằng một con số hoặc $) và kể cả mọi chữ số của mẫu tự, những con số và đường gạch dưới, từ định danh trong Verilog phân biệt dạng chữ.

5. Cú pháp

Kí hiệu cho phép:
ABDCE…abcdef…1234567890_$
Không cho phép: các kí hiệu khác -, &, #, @

6. Toán tử

Toán tử là một, hai, hoặc ba kí tự dùng để thực hiện các toán hạng trên biến. Các toán tử bao gồm >, +, &, !=.

7. Từ khóaVerilog

Có nhiều từ mã có ý nghĩa đặc biệt trong Verilog. Ví dụ: **assign, case, while, wire, reg, and, or, nand,** và **module**. Chúng không được dùng như từ định danh. Từ khóa Verilog cũng bao gồm cả chỉ dẫn chương trình biên dịch và System Task (hệ thống soạn thảo) và các hàm.

Chương II - CÁC DẠNG DỮ LIỆU

I. Đặt giá trị

Verilog bao gồm 4 giá trị cơ bản. Hầu hết các dạng dữ liệu Verilog chứa các giá trị sau:

0: mức logic 0, hoặc điều kiện sai.
1: mức logic 1, hoặc điều kiện đúng.
X: mức logic tuỳ định
Z: trạng thái tổng trở cao.
X và Z dùng có giới hạn trong tổng hợp (synthesis)

II. Wire

Mô tả vật liệu đường dây dẫn trong một mạch điện và được dùng để kết nối các cổng hay các module. Giá trị của Wire có thể đọc, nhưng không được gán trong hàm (function) hoặc khối (block). Wire không lưu trữ giá trị của nó nhưng va?n phải được thực thi bởi 1 lệnh gán kế tiếp hay bởi sự kết nối Wire với đầu ra của 1 cổng hoặc 1 module. Những dạng đặc biệt khác của Wire:

Wand(wired_and): giá trị phụ thuộc vào mức logic And toàn bộ bộ điều khiển kết nối đến Wire.

Wor (wired_or): giá trị phụ thuộc vào mức logic Or toàn bộ bộ điều khiển kết nối đến Wire.

Tri(three_state): tất cả bộ điều khiển kết nối đến 1 tri phải ở trạng thái tổng trở cao.

1. Cú pháp

Wire [msb:lsb] tên biến wire.
Wand [msb:lsb] tên biến wand.
Wor [msb:lsb] tên biến wor.
Tri [msb:lsb] tên biến tri.

2. Ví dụ

```
Wire c;
Wand d;
Assign d= a;
Assign d= b;// giá trị d là mức logic của phép And a và b.
Wire [9:0] A; // vectơ A có 10 wire.
```

III. Reg

Reg (register) là đối tượng dữ liệu mà nó chứa có giá trị từ một thủ tục gán kế tiếp. Reg chỉ được dùng trong hàm và khối thủ tục. Reg là một loại biến Verilog và không nhất thiết là thanh ghi tự nhiên. Trong thanh ghi nhiều bit, data được lưu trữ bằng các chữ số không dấu và không có kí hiệu đuôi mở rộng, được thực hiện mà người sử dụng có chủ yếu là số bù hai.

1. Cú pháp:

Reg [msb:lsb] tên biến reg.

2. Ví dụ:

```
Reg a; // biến thanh ghi đơn giản 1 bit.
Reg [7:0] A; // một vectơ 8 bit; một bank của 8 thanh ghi.
Reg [5:0]b, c; // hai biến thanh ghi 6 bit.
```

IV. Input, Output, Inout

Những từ khoá này biểu thì đầu vào, đầu ra, và port hai chiều của một module hoặc task. Một port đầu ra có thể được cấu h?nh từ các dạng: wire, reg, wand, wor, hoặc tri. Mặc định là wire.

1. Cú pháp:

Input [msb:lsb] port đầu vào.
Output [msb:lsb] port đầu ra.
Inout [msb:lsb] port đầu vào,ra hai chiều.

2. Ví dụ:

```
Module sample (b, e, c, a);
Input a; // một đầu vào mặc định là kiểu wire.
Output b, e; // hai đầu ra mặc định là kiểu wire.
Output [1:0] c; /* đầu ra hai bit, phải được khai báotrong một lệnh riêng*/
Reg [1:0] c; // đầu c được khai báo như một reg.
```

V. Integer (Số nguyên)

Integer là một biến đa năng. Trong tổng hợp chúng được dùng chủ yếu cho vòng lặp, tham số, và hằng số. Chúng hoàn toàn là reg. Tuy nhiêu chúng chứa dữ liệu bằng những số có dấu, trong khi đó khai báo dạng reg chứa chung bằng số không dấu. Nếu chúng chứa những số mà không định nghĩa thời gian biên dịch thì kích thước mặc định là 32 bit. Nếu chúng chứa hằng, sự tổng hợp điều chỉnh các số có kích thước nhỏ nhất cần thiết cho sự biên dịch.

1. Cú pháp:

Integer tên biến nguyên;
…tên hằng nguyên…;

2. Ví dụ:

```
Integer a;              // số nguyên đơn giản 32bit.
Assign b= 63; // mặc định là một biến 7 bit.
```

VI. Supply 0, Supply1

Xác định chổ đường dẫn lên mức logic 0 (đất), logic 1(nguồn) theo thứ tự định sẵn.

VII. Time

Time là một lượng 64 bit mà được sử dụng cùng với $time, hệ thống thao tác chứa lượng thời gian mô phỏng. Time không được hỗ trợ tổng hợp và và thế chỉ được dùng trong mục đích mô phỏng.

1. Cú pháp:

Time biến time;

2. Ví dụ:

Time c;
c = **$time**; // c = thời gian mô phỏng dòng điện.

VIII. Parameter (Tham số)

Một Parameter xác định 1 hằng số mà được đặt khi bạn cho ví dụ cụ thể là một module. Các này cho phép ta có thể sửa chữa.

1. Cú pháp:

Parameter par_1= giá trị, par_2= giá trị, …;
Parameter [giới hạn] par_3 = giá tr?;

2. Ví dụ:

Parameter add = 2b'00, sub = 3b'111;
Parameter n = 4;
Parameter [3:0] par_2 = 4b'1010;
…
reg [n-1:0] harry; /* một thanh ghi 4 bít mà độ rộng được đặt bởi tham số n ở trên */.

```
always @(x)
        y = {{(add - sub) {x}}}
if (x) begin
        state = par_2[1];
else
        state =par_2[2];
end.
```

Chương III - CÁC CỔNG CƠ BẢN TRONG VERILOG

Các cổng logic cơ sở là một bộ phận của ngôn ngữ Verilog. Có hai đặc tính được chỉ rõ là: drive_strenght và delay.

Drive_strenght chỉ sức bền của cổng. Độ bền đầu ra là sự kết nối một chiều đến nguồn, kế đó tạo nên sự kết nối trong suốt trans dẫn, kết thúc là tổng trở kéo lên hoặc xuống. Drive_strenght thường không được chỉ rõ, trong trường hợp này độ bền mặc định là strong1 và strong0 .

Delay: nếu delay không được chỉ rõ, thì khi đó cổng không có trì hoãn truyền tải; nếu có hai delay được chỉ định, thì trước tiên là miêu tả trì hoãn lên, thứ hai là trì hoãn xuống. Nếu chỉ có một delay được chỉ định, thì khi đó trì hoãn lên xuống là như nhau. Delay được bỏ qua trong tổng hợp. Phương pháp của sự trì hoãn chỉ định này là một trường hợp đặc biệt của "Parameterized Modules". Các tham số cho các cổng cơ sở phải được định nghĩa trước như delay.

I. Các cổng cơ bản

Các cổng cơ bản có một đầu ra, và có một hoặc nhiều đầu vào. Trong các cổng, cú pháp cụ thể biểu diễn bên dưới, các từ khoá của các cổng: and, or, nand, nor.

1. Cú pháp

GATE (drive_strength)#(delays)
Tên từ khóa cổng _tên (output, input_1, input_2, …, input_N);

Delay: #(lên, xuống) hoặc #lên_và_xuống hoặc #(lên_và_xuống)

2. Ví dụ

And c1 (o, a, b, c. d); // có 4 đầu vào cổng And gọi là c1
c2 (p, f, g); // và 2 đầu vào cổng and gọi là c2
Or #(4,3) ig (o, b, c); // cổng Or được gọi là ig, rise time = 4, fall time = 3
Xor #(5) xor1 (a, b, c); // sau 5 đơn và thời gian thì a = b xor c

II. Cổng buf, not

Các cổng này thực thi đệm và đảo theo theo thứ tự định sẵn. Chúng có một đầu vào, hai hay nhiều đầu ra. Cú pháp cụ thể biểu diễn xem ở bên dưới; từ khoá buf, not.

1. Cú pháp

Tên từ khóa cổng _tên (output_1, output_2, …, output_N, input);

2. Ví dụ

Not #(5) not_1(a,c); // sau 5 đơn và thời gian thì a = đảo c
Buf c1 (o, p, q, r, in); // bộ đệm 5 đầu ra và 2 đầu ra
c2 (p, f, g);

ChươngV- TOÁN TỬ

I. Toán tử số học

Những toán tử này thực hiện các phép tính số học. Dấu '+' và '-' có thể được sử dụng một trong hai toán tử đơn (-z) hoặc kép (x - y).

1. Toán tử:

+, -, *, /, %.

2. Ví dụ:

```
parameter n = 4;
Reg[3:0] a, c, f, g, count;
f= a +c;
g= c –n;
count = (count +1) % 16; // có thể đếm từ 0 đến 15.
```

II. Toán tử quan hệ

Toán tử quan hệ so sánh hai toán hạng và trả về một đơn bit là 0 hoặc 1. Những toán tử này tổng hợp vào dụng cụ so sánh. Biến Wire và Reg là những biến dương. Vì thế, (-3b001) = (3b111) và (-3b001) > (3b110) nhưng nếu là số nguyên thì -1< 6.

1. Các toán tử quan hệ:

<, <=, >, >=, = =, !=.

2. Ví dụ:

```
If (x= =y) e =1;
Else e= 0;
        // so sánh hai vector a, b
reg [3:0] a, b;
if (a[3] = =b [3]) a[2:0] >b[2:0];
else b[3];
```

III. Toán tử bit_wire

So sánh từng bit hai toán toán hạng.

1. Các toán tử:

~ (bitwire NOT), & (bitwire AND), | (bitwire OR), ^ (bitwire XOR), ~^ hoặc, ^~ (bitwire XNOR).

2. Ví dụ:

```
Module and2(a, b, c);
Input [1:0] a, b;
Output [1:0] c;
Assign c = a & b;
Endmodule
```

IV. Toán tử logic

Toán tử logic trả về 1 bit đơn 0 hoặc 1. chúng giống như toán tử bitwire chỉ là những toán hạng đơn bit. Chúng có thể làm việc trên biểu thức, số nguyên hoạc nhóm bit, và coi nhu tất cả các giá trị không bằng 0 là '1'. Toán tử logic được dùng nhiều trong lệnh điều kiện (if… else), khi chúng làm việc trên biểu thức.

1. Toán tử:

!(NOT), && (AND), || (OR)

2. Ví dụ:

```
Wire [7:0] x, y, z;
Reg a;
…
if ((x= = y)&&(z)) a=1;
else a=! x;
```

V. Toán tử biến đổi

Có tác dụng trên tất cả các bit của một vectơ toán hạng và trả về giá trị đơn bit. Những toán tử này là h?nh thức tự đổi số của các toán tử bitwire ở trên.

1. Các toán tử:

~ (biến đổi NOT), & (biến đổi AND), ~&(biến đổi NAND), | (biến đổi OR), ~| (biến đổi NOR), ^ (biến đổi XOR), ~^ hoặc ^~ (biến đổi XNOR).

2. Ví dụ:

```
Module chk_zero (a,z);
Input [2:0] a;
Output z;
Assign z = ~| a;
Endmodule
```

VI. Toán tử ghép

Dịch toán tử đầu bằng chữ số của các bit được định nghĩa bởi toán tử thứ hai. Vị trí còn trống sẽ được điền vào với những số 0 cho cả hai trường hợp dịch trái hoặc phải.

1. Toán tử

<< (dịch trái), >> (dịch phải).

2. Ví dụ:

assign c = a<<2; c = a dịch trái 2 bit các chỗ trống được điền với những số 0.

VII. Toán tử dịch

Ghép hai hoặc nhiều toán hạng thành một vectơ lớn.

1. Toán tử:

{} (concatenation)

2. Ví dụ:

Wire [1:0] a, b;
Wire [2:0] x;
Wire [3:0] y, Z;
Assign x = {1'b0, a}; // x[2] = 0, x[1] = a[1], x[0] = a[0].
Assign y = {a, b}; // y[3]= a[1], y[2] = a[0], y[1] = b[1], y[0] = b[0].

VIII. Toán tử thứ bản

Tạo ra nhiều bản sao của một mục chọn.

1. Toán tử:

{n{ mục chọn }} n nhóm thứ bản trong một mục chọn.

2. Ví dụ:

Wire [1:0] a, b;
Wire [3:0] x;
Assign x = {2{1'b0},a}; // x= {0, 0, a}.

IX. Toán tử điều kiện

Giống như C/C++. Chúng định giá một trong hai biểu thức cơ bản trong một điều kiện. Nó sẽ tổng hợp thành bộ đa cộng (MUX).

1. Toán tử :

(điều kiện)? kết quả khi điều kiện đúng : kết quả khi điều kiện sai.

2. Ví dụ:

assign a = (g) ? x : y;
Assign a = (inc = =2) ? a+1: a-1;

X. Thứ tự toán tử

Những toán tử trong mức giống nhau định giá từ trái sang phải

Toán tử	Tên
[ ]	Chọn bit, chọn phần
( )	Phần trong ngoặc đơn
!, ~	Mức logic và bit_wire NOT
&, |, ~&, ~|, ^, ~^	Biến đổi: AND, OR, NAND, NOT, XOR, XNOR.
+, -	Dấu chỉ số âm số dương.
{ }	Ghép nối { 3'b101,3'b110} = 6'b101110
{{ }}	Thứ bản {3{3'b101 }}=9'b101101101
*, /, %	Nhân, chia, phần trăm.
+, -	Cộng trừ nhị phân.
<<, >>	Dịch trái, phải.
<, <=, >, >=	Dấu so sánh. Biến Reg và wire được lấy bằng những số dương.
= =, !=	Bằng và không bằng trong toán tử logic.
&	Bit_wire AND, and tất cả các bit với nhau.
^, ~^	Bit_wire XOR, Bit_wire XNOR.
|	Bit_wire OR.
&&, ||	Toán tử logic AND, OR.
?:	x = (điều kiện) T:F

Chương VI- TOÁN HẠNG

I. Literals (dạng kí tự)

Là toán hạng có giá trị không đổi mà được dùng trong biểu thức Verilog. Có hai dạng kí tự là:

Chuỗi: là một mảng có nhiều kí tự được đặt trong dấu “”.

Chữ số: là những số không đổi, nhị phân, bát phân, thập phân, hoặc số hex.

1. Cú pháp các chữ số:

n’F dddd…

Trong đó:

n : số nguyên miêu tả số bit.

F: một trong bốn định dạng sau: b(số nhị phân), o(số bát phân), d(số thập phân), h(số hex).

2. Ví dụ:

```
“time is”      // chuỗi kí tự.
267            // mặc định 32 bit số thập phân.
2’b01          // 2 bit nhị phân.
20’h B36E      // 20 bit số hex.
’o62           // 32 bit bát phân.
```

II. Chọn 1 phần tử bit và chọn 1 phần các bit

Đây là sự lựa chọn một bít đơn hoặc một nhóm bit theo thứ tự, từ một wire, reg hoặc từ tham số đặt trong ngoạc [ ]. Chọn 1 phần tử bit và chọn 1 phần các bit có thể được dùng như là các toán hạng trong biểu thức bằng nhiều cách thức giống nhau mà các đối tượng dữ liệu gốc được dùng.

1. Cú pháp:

Tên biến [thứ tự bit].

Tên biến [msb: lsb].

2. Ví dụ:

```
Reg [7:0] a, b;
Reg [3:0] ls;
c = a[7] & b[7];
ls = a[7:4] + b[3:0];
```

III. Gọi hàm chức năng

Giá trị trả về của một hàm có thể được dùng trực tiếp trong biểu thức mà không cần gán trước cho biến reg hoặc wire. Gọi hàm chức năng như là một trong những toán hạng. Chiều rộng bít của giá trị trả về chắc chắn được biết trước.

1. Cú pháp:

Tên hàm(danh sách biến).

2. Ví dụ:

```
Assign a = b & c & chk_bc(b, c);
Function chk_bc;
Input c, b;
Chk_bc = b^ c;
Endfunction
```

IV. Wire, reg, và tham số

Wire, reg, và tham số có thể được dùng như là các toán hạng trong biểu thức Verilog.

Chương VII - MODULES

I. Khai báo modules:

Một module là bản thiết kế chủ yếu tồn tại trong Verilog. Dòng đầu tiên của khai báo module chỉ rõ danh sách tên và port (các đối số). Các dòng kế tiếp chỉ rõ dạng I/O (input/output, hoặc inout) và chiều rộng của mỗi port. Mặc định độ rộng port là 1 bit. Sau đó, các biến port phải được khai báo wire, wand, …, reg (mặc định là wire). Các đầu vào là dạng wire khi dữ liệu được chốt bên ngoài module. Các đầu ra là dạng reg nếu các t/hiệu của chúng được chứa trong khối always hoặc initial.

1. Cú pháp:

Module tên module (danh sách port);
Input [msb:lsb] danh sách port đầu vào;
Output [msb:lsb] danh sách port đầu ra;
Inout [msb:lsb] danh sách port vào_ ra;
… các lệnh…
endmodule

2. Ví dụ:

Module add_sub(add, in1, in2, out);
Wire, reg, và tham số:
Input[7:0] in1, in2;
Wire in1, in2;
Output [7:0] out;
Reg out;
… các lệnh khác…
Endmodule

II. Chỉ định liên tiếp:

Các chỉ định liên tiếp được dùng để gán một giá trị lên trên một wire trong một module; bên ngoài khối always hoặc khối initial. Các chỉ định liên tiếp được thực hiện với một lệnh gán (assign) rõ ràng hoặc bằng sự chỉ định một giá trị đến một wire trong lúc khai báo. Lưu ý, các lệnh chỉ định liên tiếp thì tồn tại và được chạy liên tục trong suốt quá trình mô phỏng. Thứ tự các lệnh gán không quan trọng. Mọi thay đổi bên phải của bất cứ đầu vào sẽ lập tức thay đổi bên trái của các đầu ra.

1. Cú pháp:

Wire biến wire = giá tr?;
Assign biến wire = biểu thức;

2. Ví dụ:

Wire [1:0] a = 2'b 01;
Assign b = c &d;
Assign d = x | y;

III. Module instantiations:

Các khai báo module phải theo mẫu từ các đối tượng thực tế (instantiation). Các module đơn bên trong các module khác, và mỗi dẫn chứng tạo một đối tượng độc nhất từ khuôn mẫu. Ngoại trừ đó là module mức trên là những dẫn chứng từ chính chúng.

Các port của module ví dụ phải thỏa những định nghĩa trong khuôn mẫu. Đây là mặt lý thuyết: bằng tên, sử dụng dấu chấm(.) ".tên port khuôn mẫu (tên của wire kết nối đến port)". Bằng và trí, đặt những port ở những và trí giống nhau trong danh sách port của cả khuôn mẫu lẫn instance.

1. Cú pháp:

Tên instance1 (danh sách kết nối port);
Tên instance2(danh sách kết nối port);
…

2. Ví dụ:

```
// định nghĩa module
module and4(a,b,c);
input [3:0]a,b;
output [3:0]c;
assign c = a&b;
endmodule
// module instantiations
wire [3:0] in1, in2;
wire [3:0] o1, o2;
        // đặt và trí
and4 C1(in1, in2,o1);
        // tên
and4 C2(.c(o2), .a(in1), .b(in2));
```

Chương VIII - KHUÔN MẪU HÀNH VI (BEHAVIORAL)

Verilog có 4 mức khuôn mẫu:

- Chuyển mạch. (Không xét ở giáo trình này).
- Cổng.
- Mức tràn dữ liệu.
- Hành vi hoặc thủ tục được đề cập ở bên dưới.

Các lệnh thủ tục Verilog được dùng tạo một mẫu thiết kế ở mức cao hơn. Chúng chỉ ra những cách thức mạnh của vệc làm ra những thiết kế phức tạp. Tuy nhiên, những thay đổi nhỏ n phương pháp mã hóa có thể gây ra biến đổi lớn trong phần cứng. Các lệnh thủ tục chỉ có thể được dùng trong những thủ tục.

I. Những chỉ định theo thủ tục

Là những chỉ định dùng trong phạm vi thủ tục Verilog (khối always và initial). Chỉ biến reg và integers (và chọn đơn bit/ nhóm bit của chúng, và kết nối thông tin) có thể được đặt bên trái dấu '=' trong thủ tục. Bên phải của chỉ định là một biểu thức mà có thể dùng bất cứ dạng toán tử nào.

II. Delay trong chỉ định

Trong chỉ định trễ Δt là khoảng thời gian trải qua trước khi một lệnh được thực thi và bên trái lệnh gán được tạo ra. Với nhiều chỉ định trễ (intra-assignment delay), bên phải được định giá trị trực tiếp nhưng có một delay của Δt trước khi kết quả được đặt bên trái lệnh gán. Nếu thêm một quá trình thay đổi nửa cạnh bên phải tín hiệu trong khoảng thơi gian Δt, thì không cho kết quả ở đầu ra. Delay không được hỗ trợ bởi các công cụ.

1. Cú pháp chỉ định thủ tục:

Biến = biểu thức;

Chỉ định trễ:

#Δt biến = biểu thức;

intra_assignment delay:

biến = #Δt biểu thức.

2. Ví dụ:

```
Reg [6:0] sum; reg h, zilch;
Sum[7] = b[7]^c[7];   // thực thi tức thời;
Ziltch = #15 ckz & h; // ckz & h định giá trị tức thời;
                      //ziltch thay đổi sau 15 đơn vị thời gian.
#10 hat = b & c;      /* 10 đơn và thời gian sau khi ziltch thay đổi,
b & c được định giá và hat thay đổi*/
```

III. Chỉ định khối

Chỉ định khối (=) thực hiện liên tục trong thứ tự lệnh đã được viết. Chỉ định thứ hai không được thực thi nếu như chỉ định đầu cho hoàn thành.

1. Cú pháp:

Biến = biểu thức;

Biến = #Δt biểu thức;

#Δt biến = biểu thức;

2. Ví dụ:

```
Initial
Begin
a = 1; b = 2; c = 3;
#5 a = b + c;   // sau 5 đơn và thời gian thực hiện a = b + c = 5.
d = a;          // d = a = 5.
Always @(posedge clk)
Begin
        Z = Y; Y = X;       // thanh ghi dịch.
        y = x; z = y;       // flip flop song song.
```

IV. Begin …end

Lệnh khối begin … end được dùng để nhóm một vài lệnh mà một lệnh cú pháp được cho phép. Bao gồm function, khối always và khối initial. Những khối này có thể được tùy ý gọi tên. Và bao gồm khai báo reg, integer, tham số.

1. Cú pháp:

```
Begin: tên khối
Reg[msb:lsb] danh sách biến reg;
Integer [msb:lsb] danh sách integer;
Parameter [msb:lsb] danh sách tham số;
…các lệnh…
End
```

2. Ví dụ:

```
function trivial_one;// tên khối là: trivial_one
input a;
begin: adder_blk
integer i;
… lệnh…
end
```

V. Vòng lặp for

Giống như c/c++ được dùng để thực hiện nhiều lần một lệnh hoặc khối lệnh. Nếu trong vòng lặp chỉ chứa một lệnh thì khối begin … end có thể bỏ qua.

1. Cú pháp:

For (biến đếm = giá trị 1; biến đếm </ <=/ >/ >= giá trị 2;
biến đếm = biến đếm +/- giá tr?)
begin
… lệnh …
end

2. Ví dụ:

For (j = 0; j<=7; j = j+1)
Begin
c[j] = a[j] & b[j];
d[j] = a[j] | b[j];
end

VI. Vòng lặp while

Vòng lặp while thực hiện nhiều lần một lệnh hoặc khối lệnh cho đến khi biểu thức trong lệnh while định giá là sai.

1. Cú pháp:

While (biểu thức)
Begin
… các lệnh…
end

2. Ví dụ:

While (!overflow)
@(posedge clk);
a = a +1;
end

VII. Khối lệnh if… else if… else

Thực hiện một lệnh hoặc một khối lệnh phụ thuộc vào kết quả của biểu thức theo sau mệnh đề if.

Cú pháp

If (biểu thức)
Begin
… các lệnh…
end
else if (biểu thức)
Begin
… các lệnh…
end
else
Begin
… các lệnh…
end

VIII. Case

Lệnh case cho phép lựa chọn trường hợp. Các lệng trong khối default thực thi khi không có trường hợp lựa chọn so sánh giống nhau. Nếu không có sự so sánh, bao gồm cả default, là đúng, sự tổng hợp sẽ tạo ra chốt không mong muốn.

1. Cú pháp:

Case (biểu thức)
Case 1:
Begin
… các lệnh…
end
Case 2:
Begin
… các lệnh…
end
Case 3:
Begin
… các lệnh…
end
…
default:
begin
… các lệnh…
end
endcase

2. Ví dụ:

Case (alu_clk)
2’b00: aluout = a + b;
2’b01: aluout = a - b;
2’b10: aluout = a & b;
default:
aluout = 1’bx;
endcase

Chương IX KHỐI ALWAYS VÀ KHỐI INITIAL

I. Khối always:

Là cấu trúc chính trong khuôn mẫu RTL (Register Transfer Level). Khối always có thể được dùng trong chốt, flip flop hay các kết nối logic. Tất cả các khối always trong một module thực thi một cách liên tục. Nếu các lệnh của khối always nằm trong phạm vi khối begin… end thì được thực thi liên tục, nếu nằm trong khối fork… join, chúng được thực thi đồng thời (chỉ trong mô phỏng). Khối always thực hiện bằng mức, cạnh lên/xuống của một or nhiều tín hiệu (các tín hiệu cách nhau bởi từ khóa OR).

Cú pháp:

Always @(sự kiện 1 or sự kiện 2 or…)
Begin
… các lệnh…
end
Always @(sự kiện 1 or sự kiện 2 or…)
Begin: tên khối
… các lệnh…
end

II. Khối initial

Tương tự khối always nhưng khối initial chỉ thực thi một lần từ lúc bắt đầu của quá trình mô phỏng. Khối này là tiêu biểu để biến khởi chạy và chỉ định dạng sóng tín hiệu trong lúc mô phỏng.

1. Cú pháp:

Initial
Begin
… các lệnh…
end

2. Ví dụ:

```
Initial
Begin
Clr = 0;
Clk = 1;
End
Initial
Begin
a = 2'b00;
#50 a = 2'b01;
#50 a = 2'b10;   end
```

Chương X- HÀM

Hàm được khai báo trong phạm vi một module, và có thể được gọi từ các lệnh liên tục, khối always, hoặc các hàm khác. Trong lệnh chỉ định liên tục, cũng được chỉ định liên tục khi bất kỳ các hàm khai báo đầu vào thay đổi. Trong chương trinh chúng được chỉ định tới khi cần gọi.

Các hàm mô tả sự kết nối logic, và không tạo ra chốt. Do đó một lệnh *if mà không else* sẽ mô phỏng , mặc dù nó có chốt dữ liệu nhưng mô phỏng thì không có. Đây là trường hợp dở của tổng hợp không có mô phỏng theo sau. Đây là khái niệm tốt để mã hóa hàm, và vậy chúng sẽ không tạo ra chốt nếu mã hàm được dùng trong một chương trình.

I. Khai báo hàm:

Khai báo hàm là chỉ ra tên hàm, chiều rộng của hàm giá trị trả về, đối số hàm dữ liệu vào, các biến (reg) dùng trong hàm, và tham số cục bộ của hàm, số nguyên của hàm.

1. Cú pháp:

Function [msb:lsb] tên hàm;
Input [msb:lsb]biến vào;
Reg [msb:lsb]biến reg;
Parameter [msb:lsb] tham số;
Integer [msb:lsb] số nguyên;
… các lệnh…
endfunction

2. Ví dụ

```
Function [7:0] my_func; // hàm trả về giá trị 8 bit
Input [7:0] i;
Reg [4:0] temp;
Integer n;
temp = i[7:4]| (i[3:0]);
my_func =    temp,i[1:0]    ;
endfunction
```

II. Ví dụ:

Một hàm chỉ có chứa một dữ liệu ra. Nếu có nhiều hơn một giá trị trả về được yêu cầu, đầu ra sẽ phải kết nối tạo thành một vector trước khi đặt giá trị cho hàm để gọi tên hàm. Gọi tên chương trình module có thể trích ra sau đó, riêng đối với đầu ra từ các biểu mẫu nối vào nhau. Ví dụ dưới đây minh họa tổng quát cách dùng và cú pháp hàm trong verilog.

1. Cú pháp:

Tên hàm = biểu thức.

2. Ví dụ:

```
Module simple_processor (instruction, outp);
Input [31:0] instruction;
Output [7:0] outp;
Reg [7:0] outp;// có thể được gán trong khối always.
Reg func;
Reg [7:0] opr1, opr2;
Function[16:0] decode add(instr)
Input [31:0] instr;
Reg add_func;
Reg [7:0] opcode, opr1, opr2;
Begin
        Opcode = instr[31:24];
        Opr1 = instr[7:0];
Case (opcode)
        8'b 10001000:
begin
        add_func = 1;
        opr2 = instr[15:8];
end
        8'b 10001001:
begin
        add_func = 0;
        opr2 = instr[15:8];
end
        8'b 10001010: begin
        add_func = 1;
        opr2 = 8'b 00000001;
end
default: begin
        add_func = 0;
        opr2 = 8'b00000001;
end
endcase
        decode_add = □   add_func, opr2, opr1□   ;
end
endfunction
always @(intruction) begin
        {func, opr2, opr1}= decode_add (intruction);
if (func= =1)
        outp = opr1+ opr2;
else
        outp = opr1 – opr2;
end
endmodule
```

Chương XI
CHỨC NĂNG LINH KIỆN

Chốt dữ liệu (latches): được suy nếu một biến, một trong các bit không được gán trong các nhánh của một lệnh if. Chốt dữ liệu cũng được suy ra từ lệnh case nếu một biến được gán chỉ trong một vài nhánh.

Cú pháp:

If… else if… else và case.

I. Thanh ghi Edge_triggered, flip_flop, bộ đếm:

Một thanh ghi (flip_flop) được suy luận bằng việc dùng xung kích cạnh lên hoặc xuống trong danh sách sự kiện của lệnh khối always.

Cú pháp:

```
Always @(posedge clk or posedge reset1 or nesedge reset2)
Begin
If (reset1) begin
Các chỉ định reset
end
else if (reset2) begin
Các chỉ định reset
End
Else begin
Các chỉ định reset
End
```

II. Bộ đa cộng:

Được suy ra bởi việc gán một biến mà giá trị mỗi biến khác nhau trong mỗi nhánh của lệnh if hoặc case. Có thể tránh các chỉ định và mọi nhánh có thể tồn tại bằng việc sử dụng ngoài những nhánh mặc định. Chú ý rằng chốt sẽ được tạo ra nếu một biến không được gán cho các điều kiện nhánh có thể tồn tại.

Để hoàn thiện mã có thể đọc được, dùng lệnh case để tạo mẫu đa cộng lớn.

III. Bộ cộng, trừ:

Toán tử cộng trừ trong bộ cộng trừ mà có chiều rộng phụ thuộc vào chiều rộng của toán tử lớn hơn.

IV. Bộ đệm 3 trạng thái:

Bộ đệm ba trạng thái được suy ra nếu biến được gán theo điều kiện giá trị tổng trở cao Z dùng một trong các toán tử: if, case,…

V. Các linh kiện khác:

Hầu hết các cổng logic được suy ra từ việc dùng những toán hạng tương ứng của chúng. Như một sự lựa chọn một cổng hoặc một thành phần có thể được giải thích rõ ràng bằng ví dụ cụ thể và sử dụng các cổng cơ sở (and, or, nor, inv…) miễn là bằng ngôn ngữ Verilog.

Chương XII - MỘT SỐ VÍ DỤ

I. Cấu trúc một chương trình dùng ngôn ngữ Verilog:

// Khai báo module
Module tên chương trình (tên biến I/O); // tên chương trình trùng tên file.v.
Input [msb:lsb] biến;
Output [msb:lsb] biến;
Reg [msb:lsb] biến reg;
Wire [msb: lsb] biến wire;
// Khai báo khối always, hoặc khối initial.
… các lệnh …
Endmodule

II. Một số ví dụ:

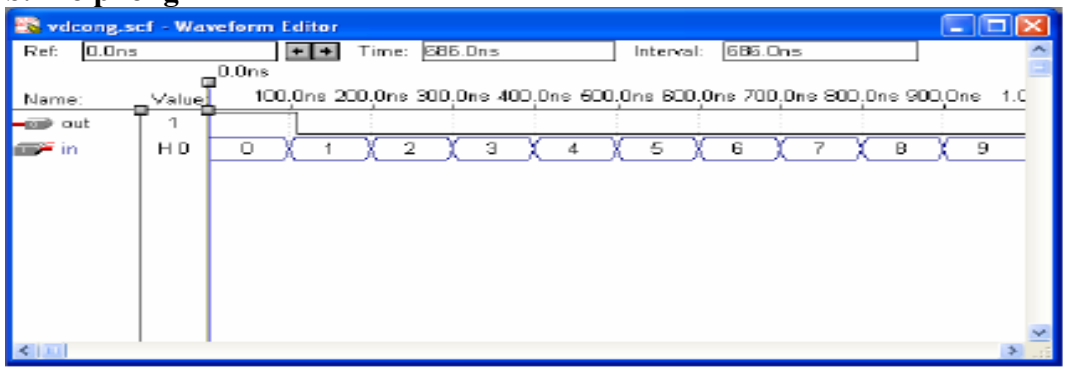
Phần mềm hỗ trợ: **MAX+plusII 10.0 BASELINE**

1. Ví dụ 1:

a. Chương trình tính NOR các bit của biến vào

```
module vdcong(in,out);
input[3:0] in;
output out;

    assign out= ~|in;

endmodule
```

b. Mô phỏng



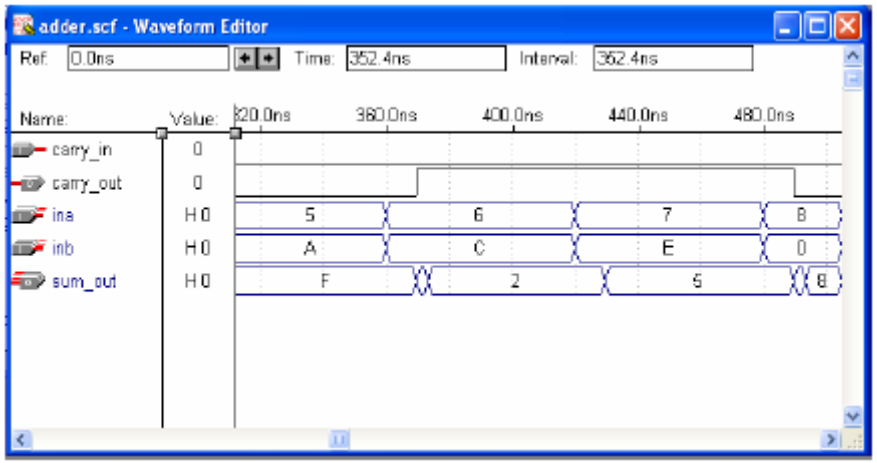
2. Ví dụ 2:

a. Chương trình cộng hai biến bốn bit

```
module adder (sum_out, carry_out, carry_in, ina, inb);
```

```
output [3:0]sum_out;
input [3:0]ina, inb;
output carry_out;
input carry_in;

wire carry_out, carry_in;
wire[3:0] sum_out, ina, inb;

assign
        { carry_out, sum_out } = ina + inb + carry_in;

Endmodule
```

b. Mô phỏng

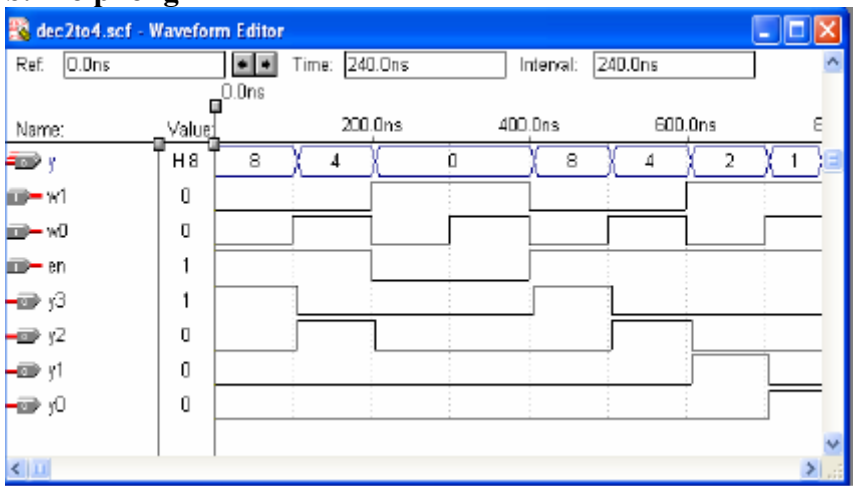


3. Ví dụ 3:

a. Chương trình giải mã 2 sang 4

```
module dec2to4 (w, en, y);
input [1:0] w;
input en;
output[3:0] y;

wire[1:0]w;
reg[3:0]y;
wire en;

always @(w or en)

begin
if(en==1'b1)
begin

case(w)
        2'b00: y<=4'b1000;
        2'b01: y<=4'b0100;
        2'b10: y<=4'b0010;
default:y<=4'b0001;
endcase
end

else
        y<= 4'b0000;
end

endmodule
```

b. Mô phỏng

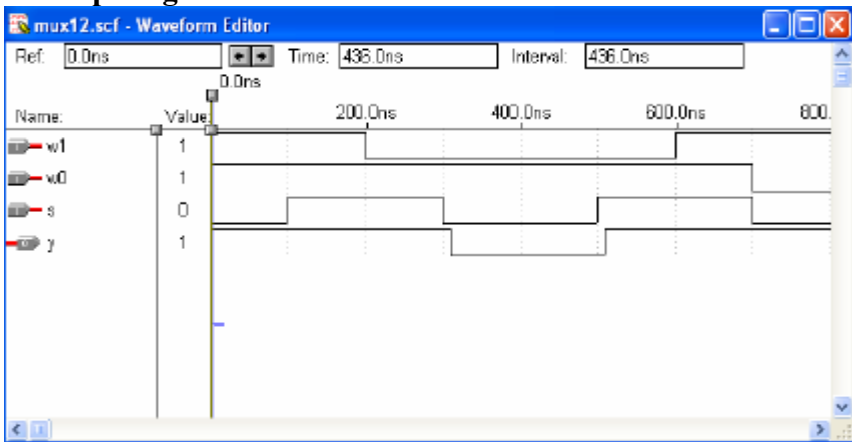


4. Ví dụ 4:

a. Bộ dồn kênh 2 sang 1

```
module mux12(w0, w1, s, y);
input w0, w1;
input s;
output y;

wire w0, w1, s;
reg y;
always @(w0 or w1 or s)

begin
if(s==1)
        y = w0;
else
        y = w1;
end

endmodule
```

b. Mô phỏng

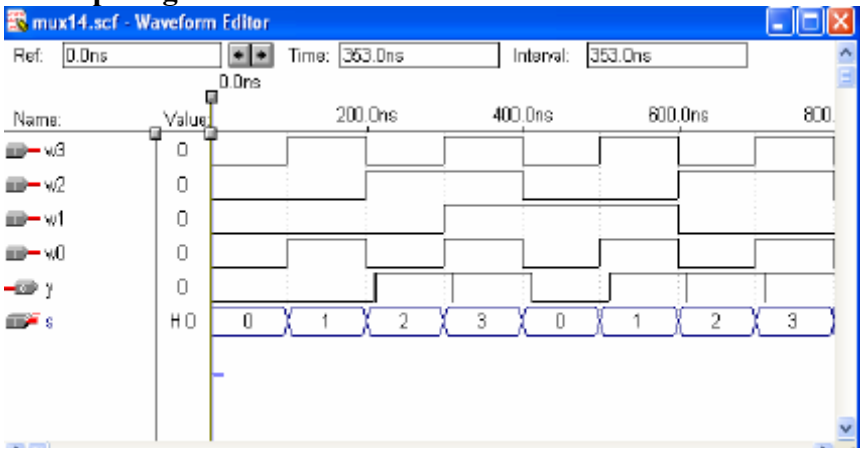


5. Ví dụ 5:

a. Chương trình dồn kênh 4 sang 1

```
module mux14(w0, w1, w2, w3, s, y);
input w0, w1, w2, w3;
input[1:0] s;
output y;

wire w0, w1,w2,w3;
reg y;
always @(w0 or w1 or s)
begin
case (s)
        2'b00: y=w0;
        2'b01: y=w1;
        2'b10: y=w2;
default: y = w3;

endcase

end

endmodule
```

b. Mô phỏng



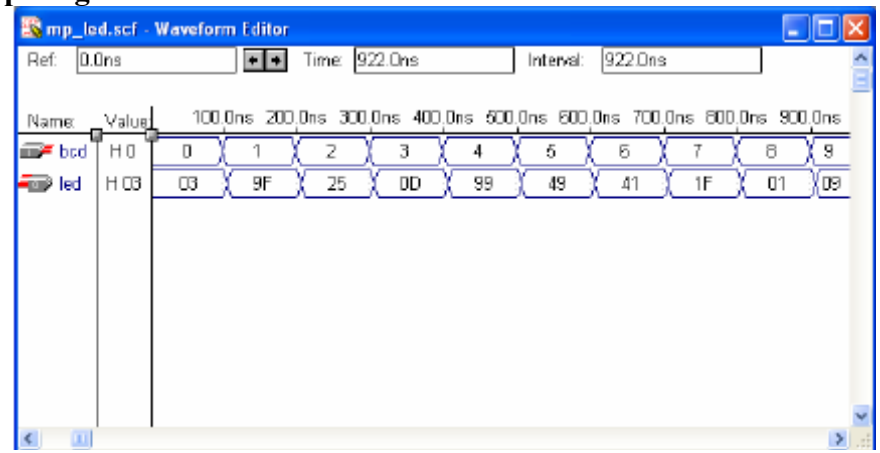
6. Ví dụ 6:

a. Chương trình đổi BCD sang bảy đoạn

```
Module mp_led(bcd,led);
input [3:0] bcd;
output [7:0] led;
```

```
wire [3:0] bcd;
reg [7:0] led;

always @(bcd)

begin
case(bcd)
        4'b0000: led = 8'b00000011;
        4'b0001: led = 8'b10011111;
        4'b0010: led = 8'b00100101;
        4'b0011: led = 8'b00001101;
        4'b0100: led = 8'b10011001;
        4'b0101: led = 8'b01001001;
        4'b0110: led = 8'b01000001;
        4'b0111: led = 8'b00011111;
        4'b1000: led = 8'b00000001;
        4'b1001: led = 8'b00001001;
        default: led = 8'b00000000;
endcase

end

endmodule
```

b. Mô phỏng

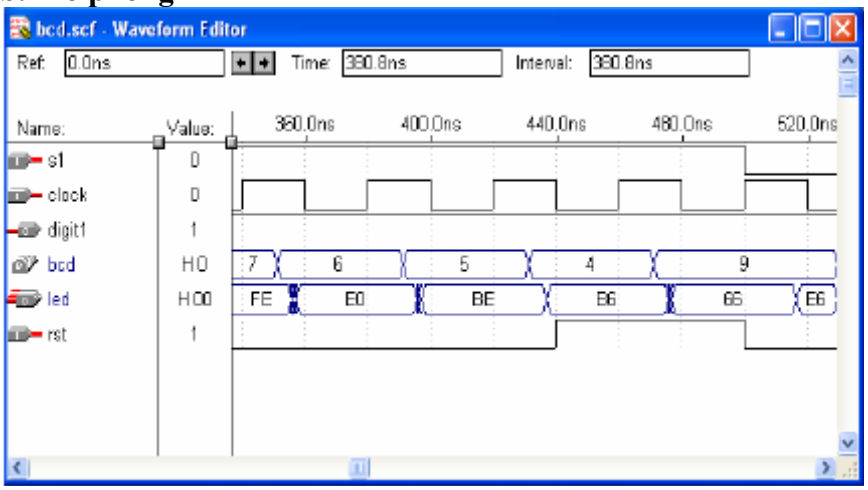


7. Ví dụ 7:

a. Chương trình giảm từ 9 xuống 0, hiển thì ra led 7 đoạn

```
module bcd (clock, rst, s1, led, digit1);
input clock, s1, rst;
output [7:0] led;
output digit1;

reg [7:0] led;
reg [3:0] bcd;
wire digit1;
assign digit1 = 1'b1;

always @(posedge clock )
begin
        if (rst == 1'b1) bcd <= 4'b1001;
        else if (s1 == 1'b1) bcd <= bcd - 1'b1;
        if (bcd == 4'b0) bcd <= 4'b1001;
end
always @(posedge clock)
begin
case(bcd)
        4'b0000: led = 8'b11111100;
        4'b0001: led = 8'b01100000;
        4'b0010: led = 8'b11011010;
        4'b0011: led = 8'b11110010;
        4'b0100: led = 8'b01100110;
        4'b0101: led = 8'b10110110;
        4'b0110: led = 8'b10111110;
        4'b0111: led = 8'b11100000;
        4'b1000: led = 8'b11111110;
        4'b1001: led = 8'b11100110;
        default: led = 8'b11111111;
endcase

end

endmodule
```

b. Mô phỏng

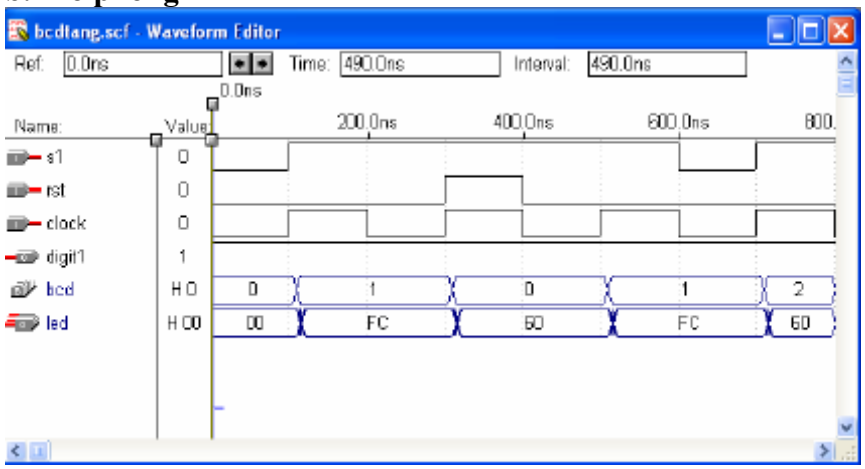


8. Ví dụ 8:

a. Chương trình tăng từ 0 đến 9, hiển thì ra led 7 đoạn

```
        module bcdtang (clock, rst, s1, led, digit1);
        input clock, s1, rst;
        output [7:0] led;
        output digit1;

        reg [7:0] led;
        reg [3:0] bcd;
        wire digit1;
        assign digit1 = 1'b1;

        always @(posedge clock )
        begin
                if (rst == 1'b1) bcd <= 4'b0;
                else if (s1 == 1'b1) bcd <= bcd + 1'b1;
                if (bcd == 4'b1001) bcd <= 4'b0000;
        end

        always @(posedge clock)
        begin
        case(bcd)
                4'b0000: led = 8'b11111100;
                4'b0001: led = 8'b01100000;
                4'b0010: led = 8'b11011010;
                4'b0011: led = 8'b11110010;
                4'b0100: led = 8'b01100110;
                4'b0101: led = 8'b10110110;
                4'b0110: led = 8'b10111110;
                4'b0111: led = 8'b11100000;
                4'b1000: led = 8'b11111110;
                4'b1001: led = 8'b11100110;
                default: led = 8'b11111111;
        endcase

        end

        endmodule
```

b. Mô phỏng



TÀI LIỆU THAM KHẢO

1. ***"Verilog Digital System Design"***
2. ***"Introduction of Verilog"*** **Peter M. Nyasulu**
3. ***"Cadence Verilog – XL Reference Manual"***
4. ***"Synopsys HDL Compiler for Verilog Reference Manual"***