10/9/2013

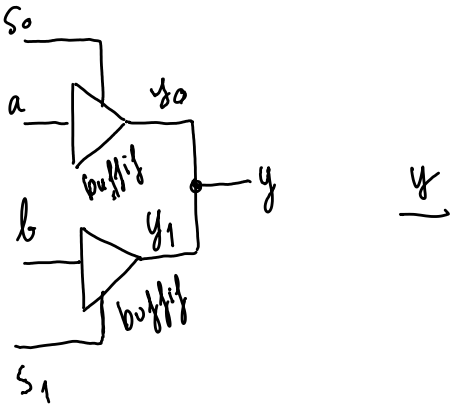
Tuesday, September 10, 2013 8:05 AM

CHƯƠNG 2: Kiểm tra chức năng IC số, hệ thống số (4 LT+1BT)
2.1. Khái niệm trong mô phỏng mạch số (Mô phỏng dựa trên sự kiện; logic 4 giá trị; thời gian trong mô phỏng) – 1 LT
2.1.4. Logic 4 trạng thái trong mô phỏng

Các tín hiệu trong mạch được mô phỏng có thể nhận 4 giá trị: 0, 1, x(unknown), z (high-impedance).

Ví dụ logic 4 trạng thái

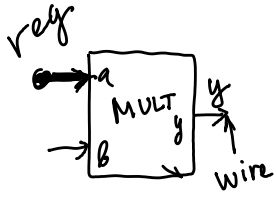
s0	0	0	1	1	1	1
a	0,1,x,z	0,1,x,z	0	1	x	z
s1	0	1	1	1	1	1
b	0,1,x,z	0,1,x,z	0	0	0	0
y0	z	z	0	1	x	x
y1	z	0,1,x,x	0	0	0	0
y	z	0,1,x,x	0	x	x	x



2.2. Testbench (Khái niệm; Chức năng; Các thành phần cơ bản của testbench; Phương pháp tạo kích thích đầu vào; Phương pháp kiểm tra đầu ra) – 1LT

2.2.1. Khái niệm và chức năng testbench

- Testbench là mô hình môi trường hoạt động của thiết kế vi mạch. Testbench được kết nối với thiết kế và được mô phỏng để kiểm tra thiết kế
- Testbench nó có 3 chức năng chính:
  - Tạo tổ hợp các giá trị đầu vào (input patterns, input stimulus) và đưa giá trị đầu vào tới thiết kế theo đúng định dạng và thời gian yêu cầu. Ví dụ: Khi cần kiểm tra thiết kế MIPS thực hiện đúng lệnh cộng, testbench cần tạo ra lệnh cộng dạng mã máy theo đúng quy định trong MIPS ISA và đưa lệnh đó vào đầu vào Instr của thiết kế tại thời điểm được yêu cầu.
  - Quan sát giá trị đầu ra, tự động tạo ra giá trị đầu ra đúng và kiểm tra giá trị đầu ra của thiết kế (so sánh với giá trị đầu ra đúng)
  - Giám sát quá trình mô phỏng và đo lường mức độ hoàn thành (chất lượng của quá trình mô phỏng)



2.2.2. Các thành phần cơ bản của testbench

a) Khai báo biến được sử dụng trong testbench
Các biến trong testbench được sử dụng để kết nối với cổng vào/ra của thiết kế hoặc sử dụng để kiểm soát quá trình đưa giá trị đầu vào tới thiết kế, kiểm soát quá trình mô phỏng

Biến trong Verilog được khai báo là **reg** hoặc **wire** theo cú pháp sau:

reg ten_bien;
wire ten_bien;

Các biến kiểu **reg** sẽ giữ nguyên giá trị cho đến khi nào nó được gán giá trị mới. Các biến kiểu reg sẽ được sử dụng trong các câu lệnh thủ tục nằm bên trong các khối lệnh **initial** và **always**.

Các biến kiểu **wire** dùng để mô tả dây dẫn trong thiết kế, nó sẽ thay đổi giá trị khi biến điều khiển nó thay đổi giá trị. Các biến kiểu wire được sử dụng trong các câu lệnh gán liên tục **assign** hoặc để kết nối các thành phần mạch.

Các biến trong Verilog có thể được khai báo là dạng bit vector hoặc dạng bộ nhớ theo cú pháp

reg [7:0] a,b;
wire [15:0] y;

reg [31:0] patterns [0:255]; // khai báo một mảng bộ nhớ 256 ô nhớ 32 bit

b) Tạo ra thiết kế như một thành phần bên trong testbench
(Module instantiation)

c) Tạo mẫu đầu vào (kích thích đầu vào)

```
module multiplier (
        input [7:0] a,b;
        output [15:0] y;
)

module multiplier_test;

        // khai báo các biến kết nối
        reg [7:0] a_sim;
        reg [7:0] b_sim;

        wire [15:0] y_sim;

        // khai báo bộ nhớ lưu trữ
        // 256 mẫu đầu vào ra và biến địa chỉ cho bộ nhớ
        reg [31:0] patterns [0:255];
        reg [7:0] addr;

        // tạo ra thiết kế nằm trong testbench
        // DUT
```

b) Tạo ra thiết kê như một thành phân bên trong testbench
(Module instantiation)

c) Tạo mẫu đầu vào (kích thích đầu vào)

Giá trị đầu vào được tạo ra và đưa tới thiết kế bằng các câu lệnh thủ tục (được thực hiện tuần tự) nằm trong khối lệnh **initial** hoặc **always**.

Khối lệnh **initial** có cú pháp như sau:

```
initial
    begin
    ... các lệnh thủ tục...
    end
```

Khối lệnh initial được thực hiện duy nhất một lần từ lúc bắt đầu mô phỏng. Các lệnh trong khối initial được thực hiện tuần tự.

Khối lệnh **always** có cú pháp như sau:

```
always @(danh_sách_tín_hiệu)
    begin
    ... các lệnh thủ tục
    end
```

Trong đó danh_sách_tín_hiệu là ***ten_bien_1 or ten_bien_2 or ...***
Khối lệnh always được thực hiện bất cứ khi nào có sự kiện xuất hiện ứng với các tín hiệu trong danh_sách_tín_hiệu. Các lệnh trong khối **always** sẽ được thực hiện tuần tự ngoại trừ lệnh gán **non-blocking.**

Mẫu tín hiệu đầu vào có thể được tạo ra theo các cách như sau:

1) Trực tiếp: một số tổ hợp giá trị đầu vào được gán trực tiếp cho các biến nối với đầu vào của thiết kế (direct test/simulation)
Thời gian đưa giá trị đầu vào tới thiết kế được điều khiển thông qua cú pháp tạo trễ lan truyền như sau:
 # giá_trị_trễ_theo_đơn_vị_thời_gian_mô_phỏng

Tạo đầu vào trực tiếp cho phép xây dựng testbench nhanh, ít tốn công sức, thời gian tuy nhiên dễ bỏ qua các kịch bản hoạt động khó trong mạch => không phát hiện hết các lỗi trong thiết kế.

2) Ngẫu nhiên: nhiều tổ hợp giá trị đầu vào ngẫu nhiên được tạo ra và đưa tới đầu vào của thiết kế.
Thời gian đưa giá trị đầu vào có thể được điều khiển thông qua cú pháp tạo trên lan truyền
Số tổ hợp đầu vào được tạo ra bằng lệnh lặp có điều kiện(Lệnh repeat, while, for)
Số ngẫu nhiên được tạo bởi hàm hệ thống $**random**()

Tạo đầu vào ngẫu nhiên cho phép xây dựng testbench nhanh, tạo được nhiều mẫu đầu vào, ít bị thiên lệch so với tạo đầu vào trực tiếp => xác suất xuất hiện lỗi cao hơn. Tuy nhiên: dễ tạo ra các đầu vào không hợp lệ, hoặc chỉ đi theo một hướng hoạt động, vẫn có khả năng để sót lỗi, bỏ qua các trường hợp hoạt động góc (corner cases)

3) Toàn bộ: tạo ra tất cả các tổ hợp đầu vào cho thiết kế.
Sử dụng vòng lặp for với biến lặp chính là các biến kết nối với đầu vào

```
// tạo ra thiết kế nằm trong testbench
// DUT

multiplier mult_dut(.a(a_sim), .b(b_sim), .y(y_sim));
```

```
// tạo kích thích đầu vào
// trực tiếp
intial
begin
    #5 a_sim = 0; b_sim=0;
    #5 a_sim = 10; b_sim = 10;
end
```

```
// tạo kích thích đầu vào
// ngẫu nhiên
initial
begin
    repeat (20)
    begin
        #5 a_sim = $random();
        b_sim=$random();
    end
end
```

```
// tạo ra tất cả các kích thích đầu vào
initial
begin
    for (a_sim=0;a_sim<255;a_sim=a_sim+1)
        for (b_sim=0;b_sim<255;b_sim=b_sim+1)
            #5;
```

Sử dụng vòng lặp for với biến lặp chính là các biến kết nối với đầu vào

Tạo đầu vào toàn bộ chỉ được sử dụng cho các mạch có kích thước nhỏ. Không khả thi cho mạch kích thước lớn. Tuy nhiên, đảm bảo chắc chắn phát hiện lỗi

4) Kết hợp ngẫu nhiên và trực tiếp theo kịch bản hoạt động
B1: Mô phỏng ngẫu nhiên mạch
B2: Đo chất lượng mô phỏng
B3: Phân tích các kịch bản hoạt động của mạch để tạo ra các trường hợp đầu vào trực tiếp tương ứng với các kịch bản hoạt động của mạch để đạt được chất lượng mô phỏng tốt hơn

Các kịch bản hoạt động của mạch cần được liệt kê trong bước Verification Plan (thực hiện đồng thời với bước Architecture design) bằng cách:

- Phân chia chức năng của thiết kế: ứng với mỗi chức năng sẽ là một trường hợp hoạt động
- Phân chia cấu hình thiết kế: ứng với mỗi cấu hình là một trường hợp hoạt động
- Phân chia sơ đồ khối của thiết kế: các trường hợp hoạt động cho phép kích hoạt tất cả các tín hiệu kết nối giữa các khối

d) Quan sát và kiểm tra đầu ra

- Quan sát và kiểm tra thủ công: Giá trị đầu ra của thiết kế được hiển thị lên màn hình dưới dạng biểu đồ dạng sóng hoặc dạng văn bản. Kỹ sư thiết kế sẽ so sánh thủ công giữa đầu ra của thiết kế và giá trị đúng (do anh ta tự tính toán) và quyết định sự đúng đắn của mạch.
Các giá trị tín hiệu được đưa ra bằng hàm hệ thống $**monitor(***chuỗi định dạng, danh sách các biến cần quan sát).* Hàm $**monitor** sẽ được thực hiện bất cứ khi nào có sự kiện xảy ra với các biến trong danh sách quan sát.
  - Không mất thời gian công sức để xây dựng testbench
  - Mất rất nhiều công sức để so sánh đối chiếu
  - Chỉ có thể áp dụng cho số lượng rất ít tổ hợp giá trị đầu vào
  - Chỉ áp dụng để gỡ lỗi
  - Cần được thay thế bằng phương pháp tự động và teschbench tự kiểm tra (self-test)
- Quan sát và kiểm tra tự động
Testbench cần có cơ chế tạo ra đầu ra đúng và so sánh đầu ra đúng với đầu ra của thiết kế, cảnh báo khi có sự sai khác
  - Đầu ra đúng có thể được tạo ra bằng mã Verilog ngay bên trong testbench. Testbench bao gồm triển khai thay thế (thứ 2) của thiết kế. Tuy nhiên bản triển khai thiết kế trong testbench không cần tối ưu, không cần tổng hợp và có thể ở mức độ trừu tượng cao hơn thiết kế thật. (Mô hình chuẩn trong testbench-golden model)
    Mô hình chuẩn và việc so sánh đầu ra được thực hiện trong khối lệnh always được kích hoạt khi các biến đầu vào thiết kế thay đổi giá trị.
  - Đầu ra đúng có thể được tạo ra bằng mô hình chuẩn ở mức phần mềm (mô hình C/C++, Matlab/Simulink).

```
    for (a_sim=0;a_sim<255;a_sim=a_sim+1)
        for (b_sim=0;b_sim<255;b_sim=b_sim+1)
            #5;
    // Chú ý: Do a_sim, b_sim là 2 số 8 bit,
    // vòng lặp chỉ test được đến 254 để tránh bị lặp
    // vô hạn lần
    // để test được đến 255 cần khai báo a_sim
    // b_sim là 2 số 9 bit
end
```

```
// Tạo đầu vào trực tiếp dựa trên kịch bản hoạt động

intial
begin
    #5 a_sim = 0; b_sim=0;
    #5 a_sim = 255; b_sim = 255;
    #5 a_sim = 0; b_sim = $random();
    #5 a_sim = 8'h55; b_sim = 8'h55;
end
```

```
// đưa ra giá trị mô phỏng
// để so sánh thủ công

initial
    $monitor("%t: %d*%d=%d",
            $time, a_sim, b_sim, y_sim);
```

```
always @(a_sim or b_sim)
begin
    if (a_sim*b_sim != y_sim)
        $display("%t: ERROR %d*%d!=%d",
            $time, a_sim, b_sim, y_sim);
end
```

đầu vào thiết kế thay đổi giá trị.

- Đầu ra đúng có thể được tạo ra bằng mô hình chuẩn ở mức phần mềm (mô hình C/C++, Matlab/Simulink). Mô hình chuẩn thường được tạo ra ở bước Architecture Design.
  Testbench giao tiếp với mô hình chuẩn để lấy đầu ra đúng thông qua:
  - Giao diện tập tin: Mô hình chuẩn sẽ ghi giá trị đầu vào và đầu ra vào tập tin và testbench sẽ đọc các giá trị này từ tập tin.
  - Giao diện lập trình: Testbench sẽ sử dụng giao diện lập trình PLI của Verilog để gọi các hàm C/C++ trong mô hình chuẩn nhằm yêu cầu tính giá trị đầu ra đúng
  - Giao diện đồng mô phỏng phần cứng (Hardware-In-the-Loop): Testbench và mô hình chuẩn Matlab/Simulink được đồng thời mô phỏng bằng kỹ thuật HIL trong môi trường Matlab/Simulink.

  Với giao diện tập tin, đầu vào và đầu ra sẽ được đọc vào mảng bộ nhớ bằng các lệnh $**readmemh**, $**readmemb**

Bài tập thuyết trình số 2: Cài đặt phần mềm mô phỏng Modelsim và mô phỏng kiểm tra chức năng của bộ nhân 8 bit song song.
Cần kiểm tra thiết kế của giáo viên đưa ra.

```
            $display(" %t: ERROR %d*%d!=%d ",
                $time, a_sim, b_sim, y_sim);
    end
```

```
    // đọc giá trị đầu vào, đầu ra từ một file text
    // định dạng file text được cho như sau
    // mỗi dòng là một số 32 bit ở hệ 16 trong đó
    // 8 bit cao nhất là giá trị của a,
    // 8 bit tiếp theo là giá trị của b,
    // 16 bit thấp nhất là giá trị của y
    // ví dụ:
    // 00000000
    // FFFFFE01
    // 00FF0000
    // 01FF00FF
    // 55551C39
    initial
    begin
       // đọc đầu vào, đầu ra từ file correct_outputs.txt
       $readmemh("correct-outputs.txt", patterns);
       // đưa kích thích tới thiết kế
       for (addr = 0;addr < 5; addr=addr+1)
         begin
              a_sim = patterns[addr][31:24];
              b_sim = patterns[addr][23:16];
              #5;
          end
    end

    // so sánh đầu ra của thiết kế và đầu ra từ file
    always @(a_sim or b_sim)
    begin
         if (patterns[addr][15:0] != y_sim)
              $display("%t: ERROR %d*%d!=%d",
                   $time, a_sim, b_sim, y_sim);
    end

endmodule
```

Chú ý: Các khối lệnh always, initial trong cùng một module có thể coi là hoạt động song song/đồng thời. Vì vậy, thời gian thực hiện các khối và mối quan hệ thời gian giữa chúng cần được xem xét cẩn thận. Ví dụ với module multiplier_test ở trên có thể xảy ra trường hợp a_sim được gán 2 giá trị khác nhau ở cùng một trường hợp mô phỏng (xung đột gán giá trị-contention). **Để tránh xung đột chỉ gán giá trị cho một biến trong một khối initial/always duy nhất.**