Verrilog HDL

Viết Testbench mô phỏng

Testbench là gì?

- Tesbench là chương trình để kiểm tra hoạt động thiết kế.
- Viết Testbech cũng phức tạp tương tự RTL code của chương trình cần kiểm tra.
- Hiện nay các ASIC ngày càng phức tạp do đó việc kiểm tra hoạt động của nó cũng phức tạp.
- Trong thiết kế ASIC, thông thường 70% thời gian cần dùng cho công việc kiểm tra ở các công đoạn (verification/validation/Testing)
- Để viết Testbench, điều quan trọng cần có thông số thiết kế và hiểu rõ các thông số đó. Sau đó đưa ra Test plan và Test case cụ thể.

Ví dụ: Bộ đếm lên 4 bit

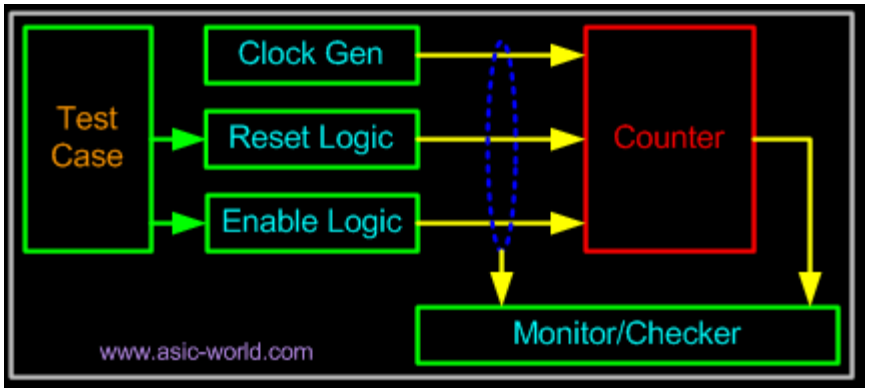
- Xét bộ đếm lên 4 bit. Bộ đếm sẽ tăng khi enable=1 và reset về 0 khi tín hiệu reset=1. tín hiệu reset được đồng bộ với xung clock.
- Code như sau

```
module counter (clk, reset, enable, count);
      input clk, reset, enable;
      output [3:0] count;
      reg [3:0] count;
always @ (posedge clk)
      if (reset == 1'b1)
       begin
              count <= 0;
       end
      else if ( enable == 1'b1)
       begin
              count <= count + 1;
       end
endmodule
```

Test Plan/ Test Case

- **Test Plan**
  - Môi trường testbench như hình bên.
  - DUT (Design-under-Test) được thực hiện trong testbench.
  - Testbech gồm khối tạo xung clock, khối tạo xung reset, khối tạo tín hiệu logic enable và mạch logic so sánh. Mạch so sánh này sẽ tính giá trị mong muốn của bộ đếm và so sánh với giá trị output của bộ đếm.

- **Test Cases**
  - Reset Test : đặt reset = 1 trong vài nhịp clock sau đó đưa reset= 0 xem bộ đếm có vê 0 không
  - Enable Test: đặt enable=1/0 sau khi đặt reset.
  - Thay đổi ngẫu nhiên enable và reset..



Writing a TestBench

- Bước 1: xây dựng một tập mẫu giả định gồm các input (kiểu reg) và các output (kiểu wire) của DUT. Sau đó thực hiện DUT theo code sau. Chú ý testbench không có port list.
- Bước 2: thêm bộ tạo xung clock. Trước khi đưa bộ tạo xung clock cần đặt cho tất cả các input các giá trị xác định như code sau.
  - Initial: chỉ được thực hiện 1 lần duy nhất, simulator sẽ thiết lập các tín hiệu clk, reset, enable về 0 (disable).
  - Có nhiều các tạo xung clock với tần số khác nhau
  - Kiểm tra xem testbench có tạo xung clock chính xác không.

```
module counter_tb;
 reg clk, reset, enable;
 wire [3:0] count;
 counter U0(.clk (clk),
              .reset (reset),
             .enable (enable),
              .count (count)
               );
// tạo xung clock
    initial
          begin
            clk = 0;
            reset = 0;
            enable = 0;
          end
    always
    #5 clk = !clk;
// phần tiếp theo của testbench (cnt.)

endmodule
```

Test Bench continues..

- **$dumpfile:** chỉ tên file sẽ lưu dạng sóng mô phỏng
- **$dumpvars:** hướng trình biên dịch Verilog bắt đầu trút toàn bộ tín hiệu vào file counter.vcd.
- $display: in ra màn hình text hoặc các giá trị.
- \t: chèn tab.
- .$monitor: theo dõi sự thay đổi của các biến (clk, reset, enable, count). Khi có bất kỳ sự thay đổi nào của các biến nó sẽ in các giá trị đó.
- $finish : dùng để dừng mô phỏng sau 100 đơn vị time. (note: initial và always block bắt đầu thực hti tại time 0).

```
// phần tiếp theo của testbench
initial
begin
    $dumpfile("counter.vcd");
    $dumpvars;
end
initial
 begin
$display("\t\ttime,\tclk,\treset,
    \tenable,\tcount");
$monitor("%d,\t%b,\t%b,\t%b,\t%d",
    $time, clk,reset,enable,count);
 end
initial #100
$finish;
//phần tiếp theo của testbench
```

Tín hiệu reset

- Phần trên đã minh họa cơ bản cách thực hiện testbench. Tiếp theo có thể thêm tín hiệu reset.
- Nhìn vào testcase, ta cần thêm một rằng buộc để có thể kích hoạt tín hiệu reset tại bất kỳ thời điểm mô phỏng nào.
- Trong Verilog có các "sự kiện" (event), các event này có thể được kích hoạt và được giám sát xem có xảy ra không.
- Viết code theo event, tín hiệu reset sẽ đợi khi sự kiện được kích hoạt "reset_trigger": khi sự kiện xảy ra, đặt reset =1 tại sườn âm của xung clock và sau đó lại cho reset =0 ở sườn âm clock tiếp theo. sau khi đặt reset=0, kích hoạt tín hiệu reset tại một sự kiện khác"reset_done_trigger". Sự kiện này sẽ được dùng ở chỗ khác trong testbench để đồng bộ.

```
event reset_trigger;
   event reset_done_trigger;
 initial begin
   forever begin
         @(reset_trigger);
         @ (negedge clk);
                reset = 1;
         @ (negedge clk);
               reset = 0;
 -> reset_done_trigger;
   end
end
```

Thêm các Test case

- Thêm một số tổ hợp test, xét 3 trường hợp
  - Reset Test : ban đầu đặt reset=0, sau đó đặt reset =1 trong vài chung kỳ clock tiếp. Quan sát xem bộ đếm có reset về 0 không.
  - Enable Test: đặt enable=1/0 sau khi đưa reset.
  - Thay đổi ngẫu nhiên Enable và fresset.
- Chú ý: có rất nhiều cách để viết test case tùy thuộc vào sự sáng tạo của người thiết kế. Trong ví dụ này xét một số trường hợp đơn giản.

Test Case

- **Test Case 1 – thay đổi tín hiệu reset**

```
//--------------------
initial
begin: TEST_CASE
 #10 -> reset_trigger;
 end
//--------------------
```

- Trong test case 1 chỉ kích hoạt sự kiện reset_trigger sau 10 đơn vị time mô phỏng.
- Trong test case 2: kích hoạt tín hiệu reset, đợi đến khi hoàn tất reset →đặt tín hiệu enable =1.

- **Test Case 2: thay đổi enable sau khi đặt reset.**

```
//-----------------------
initial
    begin: TEST_CASE
    #10 -> reset_trigger;
    @ (reset_done_trigger);
    @ (negedge clk);
          enable = 1;
    repeat(10)begin
    @ (negedge clk);
    end
enable = 0;

end
//-----------------------
```

Test Case

- **Test Case 3 – thay đổi reset/enable ngẫu nhiên.**
- Chú ý: 3 test case này không được cùng tồn tại trong 1 file vì sẽ dẫn đến hiện tượng chạy đua do 3 khối initial cùng tạo tín hiệu reset và enable.
- Thông thường , khi code các testbench, các test case được code riêng biệt và được đưa vào testbench với chỉ dẫn `include.

```
initial
   begin : TEST_CASE #10
   -> reset_trigger;
   @ (reset_done_trigger);
    fork
         repeat (10) begin
          @ (negedge clk);
                  enable = $random;
         end
         repeat (10) begin
           @ (negedge clk);
                  reset = $random;
           end
   join
end
```