Testbench in Verilog

▼ Creatore originale: @Gianbattista Busonera

Una volta terminata l'implementazione di un modulo, può essere utile realizzare un ulteriore modulo, denominato "testbench" (banco di prova). Questo modulo non realizza un'ulteriore circuito, ma serve esclusivamente per verificare la corretta funzionalità di un modulo creato in precedenza.

Il componente che viene viene sottoposto al test può essere chiamato:

- MUT, ovvero module under test;
- DUT, ovvero design under test.

Come scrivere un testbench

Primo passo - Istanziare modulo testbench e DUT

Il primo passo della scrittura di un testbench consiste nella creazione di un modulo, il quale solitamente non ha né input, né output.

```
module <module_name>_tb ();
   // istanziazione DUT
   // testbench body
endmodule
```

Come convenzione, il nome del modulo di testbench è definito con il nome del modulo da testare, seguito dalla dicitura `_tb`, salvando i file nello stesso modo.

Adesso possiamo istanziare il modulo da testare, che utilizzerà la seguente sintassi:

```
<module_name> #(.<param_name1>(<param_value1>), .<param_name2>(<param_value2>))
   <instance_name> (.<parametro_formale1>(<parametro_attuale1>), ...);

// in maniera più compatta
<module_name> #(<elenco_parametri>) <instance_name> (<elenco_porte>);
```

Il tutto risulterà, quindi, come segue:

```
module <module_name>_tb ();
   // istanziazione parametri e porte
   // ISTANZIAZIONE DUT
   <module_name> #(<elenco_parametri>) <nome_istanza> (<elenco_porte>);
   // testbench body
endmodule
```

Secondo passo - Inizializzazione del sistema in esame

L'inizializzazione del sistema in esame riguarda anche, ma non solo, la gestione del segnale di clock nel caso in cui ci troviamo a testare un circuito sequenziale.

Modellazione del tempo

Per modellare il tempo:

- si può fare in modo di aspettare un certo quantitativo di tempo tra un'istruzione e l'altra;

  ```
  <istruzione 1>;
  #5 // l'assenza del ";" è voluta!
  <istruzione 2>;
  ```

  In questo caso, viene eseguita l'istruzione 1, si attendono 5 unità di tempo definite dalla variabile `` `timescale `` e, successivamente, viene eseguita l'istruzione 2.

- si può fare in modo di assegnare un risultato ad una variabile con un certo ritardo, come già noto.

  ```
  ... // definizione variabili
  #10 a = 1'b1; // Assegna ad a il valore 1 dopo 10 unità di tempo
  ```

Blocco initial

Solitamente, l'inizializzazione del sistema in esame avviene attraverso i cosiddetti blocchi `initial`, tali per cui il codice al loro interno viene eseguito una sola volta ad inizio simulazione. Come per il blocco `always`, si tratta di un blocco procedurale, ma, a differenza di quest'ultimo, non è un blocco sintetizzabile.

```
initial begin
   ... // istruzioni varie
end
```

Un esempio di utilizzo può essere il seguente:

```
initial begin
   gate_in = 2b'00; // assegna 00 a gate in
   #10              // attende 10 unità di tempo
   gate_in = 2b'01; // assegna 01 a gate in
   #10              // attende 10 unità di tempo
   gate_in = 2b'10; // assegna 10 a gate in
   #10              // attende 10 unità di tempo
   gate_in = 2b'11; // assegna 11 a gate in
end // si noti che gate_in mantiene il valore 11 fino alla fine della simulazione
// Tale blocco initial termina dopo 30 unità di tempo in totale.
```

Ciclo forever e generazione di un segnale di clock

La parola chiave `forever` può modellare un ciclo infinito, tale per cui le istruzioni al suo interno sono eseguite per una durata indefinita durante la simulazione.

```
forever begin
    ... // istruzioni varie
end
```

Si può utilizzare per generare un segnale di clock periodico.

```
initial begin
    clk = 1b'0; // inizializzo il clock a 0... perchè posso
    forever begin
        #1 clk = ~clk; // con un ritardo di una unità di tempo il clock nega il suo valore
    end
end
```

Nel frammento di codice appena visto, si può notare come si istanzi un clock inizializzato a 0 che, per ogni unità di tempo (la quale può essere scelta in base alle necessità), commuta il proprio valore all'infinito, secondo la sequenza 0→1→0→1→0→...→0→1→...

Unità di tempo	0	1	2	3	4	5	6
clk	0	1	0	1	0	1	0

Terzo passo - Definire il comportamento dell'unita di benchmark

In questa sezione, si definiscono i blocchi `always` necessari a descrivere il comportamento dell'unità di benchmark, il quale è, generalmente, un modulo che genera dei test e, opzionalmente, verifica le risposte del DUT.

Si deve notare come una parte dell'inizializzazione o della generazione degli stimoli possa apparire in blocchi di questo tipo, come i segnali di clock.

Valutazione del comportamento

Per valutare il comportamento di un circuito si possono utilizzare le cosiddette "system tasks" (o "system functions"), il cui nome è generalmente preceduto dal simbolo "$", e risultano simili a delle syscall.

Tra le "funzioni" più utilizzate ricordiamo:

- `$display("...")`, che è simile a una `printf` in C, permettendo di stampare a schermo una volta;

  Si riporta un esempio di utilizzo di tale funzione, accostato dai modificatori utilizzabili:

  ```
  $display("x (binario) = %b");
  $display("x (decimale) = %d");
  $display("x (esadecimale) = %h");
  ```

Modificatore	Descrizione
%h, %H	Rappresentazione in esadecimale
%d, %D	Rappresentazione in decimale
%b, %B	Rappresentazione in binario
%o, %O	Rappresentazione in ottale
%m, %M	Visualizza nome gerarchico del modulo corrente
%s, %S	Rappresentazione come stringa di testo
%t, %T	Rappresentazione come tempo
%f, %F	Rappresentazione in virgola mobile
%e, %E	Rappresentazione in formato esponenziale

Tabella dei modificatori utilizzabili

- `$monitor("...")` permette di stampare a schermo ogni volta che un parametro tra i suoi argomenti cambia valore;

  Si riporta un esempio di utilizzo di tale funzione:

```
$monitor("ingresso_1 = %b, ingresso_2 = %d, ingresso_3 = %h", bin, dec, hex);
// STAMPERA' OGNI QUAL VOLTA UNO TRA I 3 INGRESSI CAMBIA VALORE!
```

- `$time()` permette di stampare il tempo raggiunto dall'inizio di una simulazione in un certo formato;

  Il system task `$time` ritorna il tempo come vettore su 64 bit, anche se esistono alcune varianti:

  - `$stime` ritorna il tempo come intero su 32 bit;
  - `$realtime` ritorna il tempo come numero reale.

  Si può modificare il formato della rappresentazione attraverso la seguente sintassi:

  ```
  $timeformat(esponente, precision, "suffix", min_width);
  // unità di misura del tempo, numero di cifre dopo la virgola, suffisso,
  ```

  Un esempio di applicazione è il seguente:

  ```
  // Multipli di 1ns, 2 posizioni decimali e almeno 10 cifre complete del dato
  $timeformat(-9, 2, "ns", 10); // 10E-9s, 2 decimali, suffisso "ns", campo da 10
  $display("ingresso = %b al tempo time = %t", binario, $time);
  ```

  ⚠️ A cosa serve `min_width`?

  Si tratta della larghezza minima, in termini di caratteri, del campo di stampa:

  - se il tempo stampato occupa meno di `min_width` caratteri, verranno aggiunti spazi a sinistra (padding) per raggiungere quella larghezza minima.
  - se il tempo stampato occupa più spazio, viene stampato comunque interamente e non viene quindi troncato.

  Vediamo un esempio pratico:

  ```
  $timeformat(-9, 2, " ns", 12);
  $display("Time: %t", $realtime);
  ```

  Se il tempo corrente è `123.45 ns`, l'output sarà:

  ```
  Time:    123.45 ns
  ```

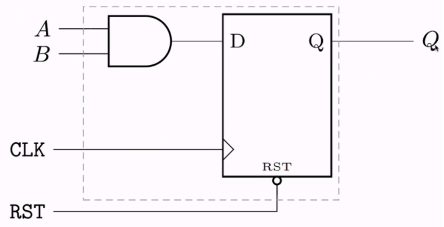
  (larghezza totale del campo = 12 caratteri, spazi inclusi)

- `$finish` conclude la simulazione, altrimenti la simulazione andrebbe avanti per sempre.

Esempio - Test di un modulo già realizzato

Si supponga di voler testare il circuito (DUT) in figura, già realizzato, in un file chiamato “example.v” in cui vi sia un modulo chiamato `example`.

Il nostro intento è quello di creare un ulteriore file “example_tb.v” e un modulo `example_tb` per eseguire dei test su di esso.



Modulo DUT

Istanziazione dei componenti

Creiamo il modulo `example_tb` e istanziamo il modulo `example`, di cui si riporta una possibile realizzazione:

```
`timescale 1ns / 1ps
module example(output reg Q, input clk, rst, a, b);
   // simuliamo che l'uscita Q possa cambiare o sul fronte di salita del clock
   // o quando il reset è attivo basso
   always @(posedge clk or !rst) begin
      if(!rst)
         Q = 0;
      else
         Q = a&b;
   end
endmodule
```

Procediamo con la creazione del testbench:

```
`timescale 1ns / 1ps
module example_tb();
   reg clock, reset;
   wire a, b;
   wire Q; // il testbench di solito non ha ingressi e uscite!
   // ISTANZIAZIONE module "example"
   example dut(
      .Q(Q),
      .clk(clock),
      .rst(reset),
      .a(a),
      .b(b)
   );

   // CONTINUAZIONE
   ...
endmodule
```

Inizializzazione del sistema in esame

Si generano i segnali di clock e reset, i quali saranno utili nella definizione del comportamento del sistema.

```
`timescale 1ns / 1ps
module example_tb();
   reg clock;
   wire reset, a, b;
```

```
    wire Q;  // il testbench di solito non ha ingressi e uscite!
    // ISTANZIAZIONE module "example"
    example dut(
        .Q(Q),
        .clk(clock),
        .rst(reset),
        .a(a),
        .b(b)
    );

    /// NUOVO CODICE QUI
    // istanziazione clock tramite ciclo infinito
    initial begin
        clock = 1'b0;
        forever begin
            #1 clock = ~clock;
        end
    end

    // facciamo variare il reset...
    initial begin
        reset = 1'b0; // inizializziamo il sistema in esame
        #10 // attendiamo 10 unità di tempo
        reset = 1'b1; // ora il sistema può funzionare
    end
    /// FINE NUOVO CODICE
    // CONTINUAZIONE
    ...
endmodule
```

▼ Comportamento del sistema

Definiamo gli stimoli da applicare: in questo caso, essi saranno dentro un blocco `initial`, poiché non ha senso testarlo all'infinito, ma basta testarlo con le 4 combinazioni possibili di `a` e `b`.

```
`timescale 1ns / 1ps
module example_tb();
    reg clock;
    wire reset, a, b;
    wire Q; // il testbench di solito non ha ingressi e uscite!
    // ISTANZIAZIONE module "example"
    example dut(
        .Q(Q),
        .clk(clock),
        .rst(reset),
        .a(a),
        .b(b)
    );

    // istanziazione clock tramite ciclo infinito
    initial begin
        clock = 1'b0;
        forever begin
            #1 clock = ~clock;
```

```
    end
  end

  // facciamo variare il reset...
  initial begin
    reset = 1'b0; // inizializziamo il sistema in esame
    #10 // attendiamo 10 unità di tempo
    reset = 1'b1; // ora il sistema può funzionare
  end

  /// NUOVO CODICE QUI
  initial begin
    $monitor("time=%3d, in_a=%b, in_b=%b, q=%b\n", $time, a, b, q);
    // STAMPIAMO A SCHERMO AD OGNI VARIAZIONE DI UNO TRA a, b, time e q
    a = 1'b0;
    b = 1'b0;
    // configurazione a = 0, b = 0
    #20  // propagazione dei valori pari a 20 unità di tempo = 20ns
    a = 1'b1;
    // configurazione a = 1, b = 0
    #20  // propagazione dei valori pari a 20 unità di tempo = 20ns
    a = 1'b0;
    b = 1'b1;
    // configurazione a = 0, b = 1
    #20  // propagazione dei valori pari a 20 unità di tempo = 20ns
    a = 1'b1;
    // configurazione a = 1, b = 1
  end
  /// FINE NUOVO CODICE

endmodule
```

Esempio - Test di un up-counter a 4 bit

Consideriamo un up-counter a 4 bit come elemento da modellare e testare, con:

- un segnale di clock che regola la temporizzazione del conteggio;
- un segnale di reset, attivo con il valore basso, per gestire il reset del dispositivo;
- un'uscita su 4 bit per rappresentare il conteggio;

Questo componente, al raggiungimento del valore massimo del conteggio, riparte dal valore 0.

```
module uc4b (out, clk, reset);
  output reg[3:0] out;
  input clk, reset;

  always @(posedge clk)
    if (!reset)
      out <= 0; // 4'b0000
    else
      out <= out + 1;
```

```
endmodule
```

Nel seguente frammento di codice, si può vedere una modalità alternativa per fare il ciclo `forever` con `always #5 clk = ~clk`, per cui è necessario definire un primo valore del clock. In caso non si facesse, si avrebbe valore `x`, che ricordiamo essere il valore sconosciuto. Per questo motivo si vede l'istruzione `clk <= 0` nel blocco `initial`.

```
module uc4b_tb ();
   reg clk, reset;
   wire [3:0] out;

   uc4b c(.clk(clk), .reset(reset), .out(out));

   always #5 clk = ~clk;

   initial begin
      $monitor("[%0tns] clk=%b rstn=%b out=0x%0h", $time, clk, reset, out);
      clk <= 0;
      reset <= 0;
      #20 reset <= 1;
      #80 reset <= 0;
      #50 reset <= 1;
      #20 $finish;
   end
endmodule
```