I linguaggi VHDL e Verilog

I linguaggi VHDL e Verilog sono i più comuni tra quelli utilizzati per descrivere le strutture hardware digitali.

VHDL

VHDL (**Very Hardware Description Language**) è standard IEEE dal 1987. Il file VHDL, scritto in modalità testo, si divide tipicamente in due parti. Dapprima si descrive il componente (ENTITY) e le sue interazioni con l'esterno; successivamente, se ne descrive il funzionamento interno (ARCHITECTURE). Di seguito è riportata, a titolo di esempio, la descrizione di un contatore binario a quattro bit.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY contatore IS
   PORT
   (
      up : IN STD_LOGIC;
      cont_bin : OUT STD_LOGIC_VECTOR(3 downto 0)
   );
END contatore;

ARCHITECTURE contatore_architecture OF contatore IS
   signal conteggio : STD_LOGIC_VECTOR(3 downto 0);
BEGIN
   process(up)
   begin
      if(rising_edge(up)) then
         conteggio <= conteggio + '1';
      end if;
   end process;
   cont_bin <= conteggio;
END contatore_architecture;
```

Dapprima si includono le librerie necessarie, poi si definiscono le relazioni esterne del dispositivo contatore (ENTITY), il clock di ingresso (up) e i quattro bit di uscita (cont_bin). Si passa poi alla sua architettura (ARCHITECTURE), definendo una variabile interna di conteggio a quattro bit (conteggio) e le due procedure concorrenti, il processo (process) sensibile alle variazioni del segnale up, in particolare ai suoi fronti di salita (rising_edge), e l'assegnazione (<=) di conteggio ai pin di uscita.

Verilog

Verilog non è un linguaggio di programmazione standard ufficiale, ma, di fatto, è come se lo fosse. Ha una sintassi più simile al linguaggio C ed è meno formale rispetto a VHDL. Di seguito è riportata, per confronto, la descrizione del medesimo contatore a quattro bit, realizzata questa volta in linguaggio Verilog, con le righe numerate al solo scopo di facilitarne il commento.

```
1  module contatore (up, cont_bin);
   //contatore binario a quattro bit
2     input up;
3     output [3:0] cont_bin;

4     reg[3:0] cont_bin;

5     always @(posedge up)
6      begin
7         cont_bin <= cont_bin + 4'b0001;
8      end
9  endmodule
```

La **riga 1** definisce il nome del modulo (contatore) e di tutti i segnali che interagiscono con l'esterno, distinti, rispettivamente in output (cont_bin, da quattro bit) e input (up), mentre la **riga 9** chiude definitivamente il modulo. La **riga 4** definisce il segnale 'cont_bin' di tipo registro, in modo che, in assenza di variazioni, questo mantenga il valore precedente. Il blocco procedurale always (**5**) indica il fronte positivo del segnale up quale unico evento abilitato ad innescare l'elaborazione descritta al suo interno, aperta da begin (**6**) e chiusa con end (**8**), e costruita mediante l'uso delle frecce di assegnazione (<=).
I commenti, come nel linguaggio C, sono contrassegnati dalla doppia barra (//) se stanno su una sola riga, oppure aperti (/*) e chiusi (*/) se occupano più righe.

ESERCIZIO A

Individuare il componente descritto dal listato che segue (le righe sono numerate al solo scopo di facilitarne il commento).

```
1  module MUX_2_1(uscita, in_a, in_b, sel);
2     output uscita;
3     input in_a, in_b, sel;
4     reg uscita; // Commento
5     always @(sel or in_a or in_b)
6      begin
7        if( sel == 0 )
8         uscita <= in_a;
9        else
10        uscita <= in_b;
11     end
12    /* Altro commento */
13 endmodule
```

SOLUZIONE

Si tratta di un file di testo in linguaggio Verilog.
La **riga 1** definisce il nome del modulo (MUX_2_1) e di tutti i segnali che interagiscono con l'esterno, distinti, rispettivamente, in *output* (**2**) e *input* (**3**), mentre la riga 13 chiude definitivamente il modulo.
La **riga 4** definisce il segnale 'uscita' di tipo *registro*, in modo che, in assenza di variazioni, il segnale mantenga il valore precedente. Il blocco procedurale *always* (**5**) specifica dapprima quali sono i segnali la cui variazione innesca l'elaborazione descritta al suo interno, aperta da *begin* (**6**) e chiusa con *end* (**11**), e costruita mediante l'impiego dello statement decisionale *if-else* e delle frecce di assegnazione (<=).
I commenti, come nel linguaggio C, sono contrassegnati dalla doppia barra (//) se stanno su una sola riga, oppure aperti (/*) e chiusi (*/) se occupano più righe.
Si tratta, in definitiva di un multiplexer a due ingressi e una uscita, con selezione mediante l'ingresso sel: se sel = 0 in uscita è riportato l'ingresso in_a, mentre invece, se sel = 1, in uscita va in_b.

ESERCIZIO B

Individuare le prestazioni del decoder 7 segmenti descritto dal listato VHDL che segue. Individuare le posizioni dei singoli segmenti e comprendere se è predisposto per display ad anodo comune.

```
ENTITY decodifica7seg IS
   PORT
   (
     conta : IN STD_LOGIC_VECTOR(3 downto 0);
     ss : OUT STD_LOGIC_VECTOR(7 downto 1)
   );
END decodifica7seg;

ARCHITECTURE decodifica7seg_architecture OF decodifica7seg IS
   BEGIN
     process(conta)begin
          case conta is
              when "0000" =>  ss <= "0111111"; -- scrive 0
              when "0001" =>  ss <= "0000110"; -- scrive 1
              when "0010" =>  ss <= "1011011"; -- scrive 2
              when "0011" =>  ss <= "1001111";
              when "0100" =>  ss <= "1100100";
              when "0101" =>  ss <= "1101101";
              when "0110" =>  ss <= "1111100";
              when "0111" =>  ss <= "0000111";
              when "1000" =>  ss <= "1111111";
              when "1001" =>  ss <= "1100111"; -- scrive 9
              when others =>  ss <= "0000000";
          end case;
     end process;

END decodifica7seg_architecture;
```

SOLUZIONE

Si tratta di un decoder BCD/7 segmenti, perché riconosce solo i valori binari a quattro bit compresi tra zero e nove.
Le uscite sono attive alte, perciò è adatto per display a catodo comune. Seguendo l'attivazione dei diversi numeri, si comprende che la posizione dei segmenti è g, f, e, d, c, b, a.

ESERCIZIO C

Sapendo che l'ingresso up del componente descritto dal listato Verilog che segue è sollecitato con una quadra di frequenza 20 MHz, determinare la forma d'onda in uscita.

```
module cont_2000 (up, usc);
   input up;
   output usc;
   reg[24:0] cont;
   reg usc;

   always @(posedge up)
      begin
       cont = cont + 25'b1;
       if (cont == 20000000) cont =0;
       if (cont < 10000000) usc=0;
       else usc=1;
      end
endmodule
```

SOLUZIONE

Il dispositivo è un contatore binario di modulo 20.000.000 e l'uscita è bassa per i primi 10.000.000 valori e alta per i successivi. Si tratta, quindi, della generazione di un segnale da 1 Hz, simmetrico, alto per 0,5 s e basso per 0,5 s.

ESERCIZIO 1

Individuare il componente descritto dal listato VHDL che segue.

```
ENTITY comp IS
   PORT (a, b: in bit_vector (3 downto 0);
      eq : out bit;
   );
END comp;
ARCHITECTURE funz OF comp IS
   signal pippo : bit;
   BEGIN
      pippo <= '1' when (a = b) else '0';
eq <= pippo;
END comp;
```

[Ris.: comparatore a 4 bit, con uscita eq attiva alta]

ESERCIZIO 2

Individuare il componente descritto dal listato Verilog che segue.

```
module cont_dec (up, cont_dec);
   input up;
   output [3:0] cont_dec;
   reg[3:0] cont;

   always @(negedge up)
      begin
       cont = cont + 4'b0001;
       if (cont > 9) cont = 0;
      end

      assign cont_dec = cont;

endmodule
```

[Ris.: contatore binario modulo dieci]

ESERCIZIO 3

Il componente descritto dal listato Verilog che segue è un decoder BCD/7 segmenti. Individuare le posizioni dei singoli segmenti, comprendere se è predisposto per display catodo comune e aggiungere quanto serve a trasformarlo in un decoder BCD/esadecimale.

```
module decodifica7seg (conta, ss );

   input [3:0]conta;
   output [7:1] ss;

   reg [7:1]ss;

   always @(conta)
      begin
       case (conta)
          4'b0000: ss <= ~7'b0111111;
          4'b0001: ss <= ~7'b0000110;
          4'b0010: ss <= ~7'b1011011;
          4'b0011: ss <= ~7'b1001111;
          4'b0100: ss <= ~7'b1100110;
          4'b0101: ss <= ~7'b1101101;
          4'b0110: ss <= ~7'b1111100;
          4'b0111: ss <= ~7'b0000111;
          4'b1000: ss <= ~7'b1111111;
          4'b1001: ss <= ~7'b1101111;
          default: ss <= ~7'b0000000;
       endcase
      end
endmodule
```

[Ris.: g, f, e, d, c, b, a; il decoder è predisposto per display anodo comune, perché il simbolo cediglia (~), come nel C, inverte tutti i bit; per visualizzare anche le lettere A, b, C, d, E, F, bisogna aggiungere sei righe all'interno del costrutto case]

```
          4'b1010: ss <= ~7'b1110111; // A
          4'b1011: ss <= ~7'b1111100; // b
          4'b1100: ss <= ~7'b0111001; // C
          4'b1101: ss <= ~7'b1011110; // d
          4'b1110: ss <= ~7'b1011001; // E
          4'b1111: ss <= ~7'b1110001; // F]
```

ESERCIZIO 4

Individuare il componente descritto dal blocco procedurale *always* che segue.

```
always @(enable, d)
   begin
    if (enable) q <= d;
   end
```

[Ris.: si tratta di un D latch, trasparente sul livello alto dell'ingresso enable]