Esercizi VHDL nelle prove d'esame di Architettura degli elaboratori (a.a. 2002/03)

18 settembre 2003

Prova del 9 giugno 2003

Esercizio

Descrizione VHDL (a scelta, dataflow o comportamentale) di un decoder per il codice di Gray a 3 bit:

X0	X1	X2	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0
1	1	1	0	0	0	0	0	1	0	0
1	0	1	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	0	1

Soluzione

Il decoder descritto nel testo dell'esercizio ha un segnale di ingresso costituito da un bus a tre bit ed un segnale di uscita costituito da un bus a 8 bit. Pertanto, può essere descritto in VHDL tramite la seguente entità:

```
entity DEC_GRAY3 is
  port (X: in bit_vector (0 to 2);
        Y: out bit_vector (0 to 7));
end DEC_GRAY3;
```

Il funzionamento del decoder può essere espressa in stile comportamentale dal seguente codice:

```
-- modello comportamentale --
architecture BEHAV of DEC_GRAY3 is
begin
  P1: process (X)
  begin
    case (X) is
      when "000" => Y <= "10000000";
      when "001" => Y <= "01000000";
      when "011" => Y <= "00100000";
      when "010" => Y <= "00010000";
      when "110" => Y <= "00001000";
      when "111" => Y <= "00000100";
      when "101" => Y <= "00000010";
      when "100" => Y <= "00000001";
    end case;
  end process P1;
end BEHAV;
```

Alternativamente, il funzionamento del decoder può essere specificato in stile dataflow mediante la seguente **architecture**:

```
architecture DATAFLOW of DEC_GRAY3 is
begin
  Y <= "10000000" when X="000"
       else "01000000" when X="001"
       else "00100000" when X="011"
       else "00010000" when X="010"
       else "00001000" when X="110"
       else "00000100" when X="111"
       else "00000010" when X="101"
       else "00000001";
end DATAFLOW;
```

Prova del 26 giugno 2003

Esercizio

Descrizione in VHDL comportamentale di un flip-flop set-reset con set sincrono e reset asincrono. L'operazione di reset è da considerare prioritaria rispetto a quella di set.

```
entity flip_flop_set_reset is
    port (set, reset, clock: in bit;
                          q: out bit);
end flip_flop_set_reset;
```

Soluzione

```
architecture behav of flip_flop_set_reset is
begin
  process (clock, reset)
  begin
    if (reset='1') then -- il reset e' prioritario
      q <= '0';
    elseif (clock'event and clock='1') then
      if (set = '1') then
        q <= '1';
      end if;
    end if;
  end process;
end behav;
```

Prova del 24 luglio 2003

Esercizio

Descrizione VHDL comportamentale o dataflow (a scelta) del componente descritto dalla seguente tabella di verità (multiplexer a 2 vie):

SEL	A	B	Z
0	0	—	0
0	1	—	1
1	—	0	0
1	—	1	1

Soluzione

La descrizione dell'entità VHDL corrispondente è:

```
entity MUX2 is
  port (A, B, SEL: in bit;
                Z: out bit);
end MUX2;

-- modello comportamentale --
architecture BEHAV of MUX2 is
begin
  P1: process (A, B, SEL)
  begin
    if (SEL = '0') then
      Z <= A;
    else
      Z <= B;
```

```
    end if;
  end process P1;
end BEHAV;

-- modello data-flow --
architecture DATA_FLOW of MUX2 is
begin
  Z <= (A and not SEL) or (B and SEL);
end DATA_FLOW;
```

Il modello comportamentale poteva essere descritto anche tramite il costrutto **case-when**, mentre per il modello dataflow poteva essere utilizzato l'assegnamento condizionato **when-else**.

Prova del 9 settembre 2003

Esercizio

Modificare la seguente descrizione VHDL di un contatore a 8 bit con reset asincrono includendo i segnali `load` e `data`. Rimodellare il comportamento dell'entità in modo che quando `load` assume il valore '1' l'uscita `outa` assuma il valore di `data` (in modo asincrono). Porre `reset` prioritario rispetto a `load`.

```
entity cont8 is
port(clk, reset: in std_logic;
       outa: out std_logic_vector(0 to 7));
end cont8;

architecture rtl of cont8 is
  signal t: std_logic_vector(0 to 7);
begin
  process(clk, reset)
  begin
    if (reset = '1' ) then
      t <= "00000000";
    elseif (clk'event and clk = '1' ) then
        t <= t + "00000001";
    end if;
  end process;

  outa <= t;
end rtl;
```

Soluzione

Il segnale `load` deve solo abilitare o inibire il caricamento del valore di `data` in `outa`. Un bit è quindi sufficiente per `load`. Il segnale `data` deve portare i valori da assegnare all'uscita del contatore: dovrà quindi essere un vettore di 8 bit. Entrambi i segnali saranno di ingresso. La dichiarazione di **`entity`** sarà quindi modificata come segue:

```
entity cont8 is
port(clk, reset: in std_logic;
           load: in std_logic;                -- nuovo segnale
           data: in std_logic_vector(0 to 7); -- nuovo segnale
           outa: out std_logic_vector(0 to 7));
end cont8;
```

Il caricamento abilitato da `load` deve essere (per specifica) asincrono. Da ciò derivano due condizioni: `load` deve essere posto nella sensitivity list e il codice relativo al caricamento deve precedere la valutazione delle condizioni su `clk`. Inoltre, per specifica, `load` deve avere una priorità più bassa di `reset`. Pertanto, la valutazione delle condizioni su `reset` dovranno precedere quelle su `load`. Il codice modificato è quindi il seguente:

```
architecture rtl of cont8 is
  signal t: std_logic_vector(0 to 7);
begin
  process(clk, reset)
  begin
    if (reset = '1' ) then
      t <= "00000000";
    elseif (load = '1') then
      t <= data;
    elseif (clk'event and clk = '1' ) then
        t <= t + "00000001";
    end if;
  end process;

  outa <= t;
end rtl;
```

Prova del 18 settembre 2003

Esercizio

Modificare la seguente descrizione VHDL di un contatore a 8 bit con reset asincrono includendo il segnale `null_val`[1]. Rimodellare il comportamento dell'entità in modo che quando `reset` assume il valore '1' l'uscita `outa` assuma il valore di `null_val` (in modo asincrono).

```
entity cont8 is
port(clk, reset: in std_logic;
       outa: out std_logic_vector(0 to 7));
end cont8;

architecture rtl of cont8 is
  signal t: std_logic_vector(0 to 7);
begin
  process(clk, reset)
  begin
    if (reset = '1') then
      t <= "00000000";
    elseif (clk'event and clk = '1' ) then
        t <= t + "00000001";
    end if;
  end process;

  outa <= t;
end rtl;
```

Soluzione

Si tratta di estendere il comportamento di base del segnale di reset per rendere dinamico il valore di reset.

A tal proposito, bisogna aggiungere nella dichiarazione di **`entity`** il nuovo segnale `null_val`. Poiché il valore di `null_val` dovrà essere assegnato al segnale `outa`, `null_val` dovrà avere modalità **`in`** ed essere dello stesso tipo di `outa` (`std_logic_vector (0 to 7)`).

Sostituendo, nella descrizione dell'architettura, il valore `"00000000"` con `null_val`, si realizzerà l'assegnamento richiesto dall'esercizio.

Il seguente codice riporta le modifiche sopra descritte:

```
entity cont8 is
```

[1]Nel testo originale della prova di esame, il segnale da aggiungere aveva, erroneamente, nome **`null`**. Essendo **`null`** una parola chiave del VHDL, si è ritenuto più conveniente utilizzare `null_val` nella soluzione qui riportata.

```
port(clk, reset: in std_logic;
           null_val: in std_logic_vector(0 to 7);  -- modifica
           outa: out std_logic_vector(0 to 7));
end cont8;

architecture rtl of cont8 is
  signal t: std_logic_vector(0 to 7);
begin
  process(clk, reset)
  begin
    if (reset = '1') then
      t <= null_val;    -- modifica
    elseif (clk'event and clk = '1' ) then
        t <= t + "00000001";
    end if;
  end process;

  outa <= t;
end rtl;
```

Le specifiche dell'esercizio non sono molto dettagliate, e non precisano se il valore di `outa` debba essere aggiornato o meno nel caso in cui il valore di `null_val` venga modificato (sempre all'interno di una finestra temporale in cui `reset` vale `'1'`) . Il codice modificato sopra riportato, non includendo `null_val` nella sensitivity list non è in grado di reagire ai cambiamenti del valore di `null_val`: l'assegnamento di reset viene quindi effettuato solo quando `reset` assume il valore `'1'` o in presenza di un fronte (sia di salita che di discesa) del segnale `clock` (se `reset` vale `'1'`).

L'aggiunta, di `null_val` nella sensitivity list, invece, rende il collegamento tra `null_val` e `outa` completamente trasparente e abilitato da `reset`:

```
  ...
  process(clk, reset, null_val)
  ...
```

Entrambe le soluzioni, tuttavia, sono da ritenersi accettabili.