Descrizione VHDL di componenti combinatori

5 giugno 2003

1 Decoder

Il decoder è un componente dotato di N ingressi e 2^N uscite. Le uscite sono poste tutte a "0" tranne quella corrispondente al numero binario in ingresso, la quale viene posta ad "1".

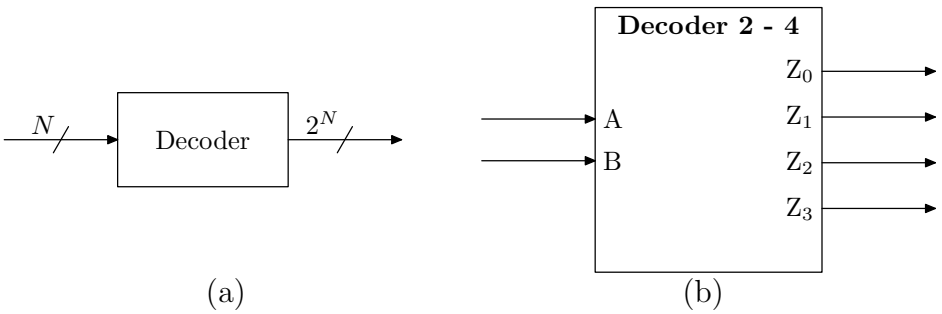

Figura 1: Rappresentazione schematica di un decoder: (a) a N ingressi; (b) a due ingressi e quattro uscite.

Ad esempio, per $N = 2$, il funzionamento del decoder è rappresentato dalla seguente tabella di verità:

A	**B**	$\mathbf{Z_0}$	$\mathbf{Z_1}$	$\mathbf{Z_2}$	$\mathbf{Z_3}$
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

La descrizione VHDL di tale componente è la seguente:

```
entity DEC24 is
  port (A, B: in bit;
        Z: out bit_vector (0 to 3));
end DEC24;
```

La descrizione del funzionamento del decoder può essere espressa dal seguente codice:

```
-- modello comportamentale --
architecture ARC1 of DEC24 is
begin
  P1: process (A, B)
  begin
    if (A='0' and B='0') then
      Z <= "1000";
    elsif (A='0' and B='1') then
      Z <= "0100";
    elsif (A='1' and B='0') then
      Z <= "0010";
    elsif (A='1' and B='1') then
      Z <= "0001"; end if;
  end process P1;
end ARC1;
```

Alternativamente, per evidenziare la mutua esclusività delle configurazioni dei segnali di ingresso, è possibile utilizzare il seguente codice:

```
-- modello comportamentale --
architecture ARC2 of DEC24 is
begin
  P2: process (A, B)
  begin
    case (A & B) is
      when "00" => Z <= "1000";
      when "01" => Z <= "0100";
      when "10" => Z <= "0010";
      when "11" => Z <= "0001";
    end case;
  end process P2;
end ARC2;
```

Il corrispondente modello "dataflow" è il seguente:

```
-- modello data-flow --
architecture DATA_FLOW1 of DEC24
is
begin
  Z <= "1000" when (A='0' and B='0')
       else "0100" when (A='0' and B='1')
       else "0010" when (A='1' and B='0')
       else "0001";
end DATA_FLOW1;
```

Oppure:

```
-- modello data-flow --
architecture DATA_FLOW2 of DEC24 is
```

```
begin
  with A & B select
    Z <= "1000" when "00",  -- 0
         "0100" when "01",  -- 1
         "0010" when "10",  -- 2
         "0001" when "11";  -- 3
end DATA_FLOW2;
```

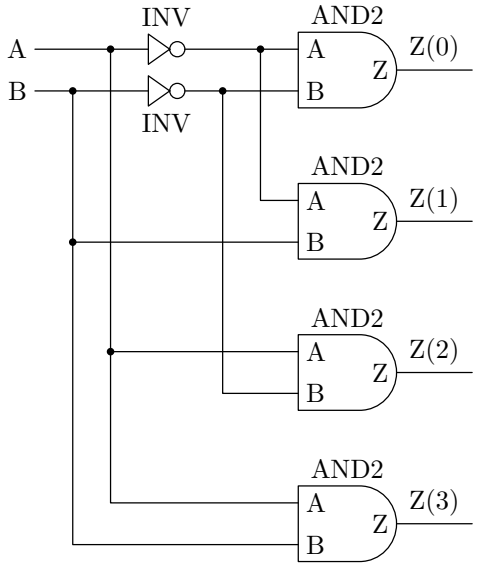

Figura 2: Rappresentazione circuitale del decoder a due vie

In alternativa, considerando lo schema in figura 2 (derivato tramite le note tecniche di sintesi dei circuiti combinatori) il decoder può essere descritto tramite la seguente architettura dataflow:

```
-- modello data-flow --
architecture DATA_FLOW3 of DEC24 is
begin
  Z(0) <= (not A and not B);
  Z(1) <= (not A and B);
  Z(2) <= (A and not B);
  Z(3) <= (A and B);
end DATA_FLOW3;
```

Inoltre, considerando le porte logiche come componenti, è possibile dare la seguente descrizione strutturale del decoder in esame:

```
-- modello strutturale --
architecture STRUCT of DEC24 is
  component INV
    port (A: in bit;
```

```
        Z: out bit);
  end component;
  component AND2
    port (A, B: in bit;
          Z: out bit);
  end component;
  signal NOT_A, NOT_B: bit;
begin
  u1: INV port map (A, NOT_A);
  u2: INV port map (B, NOT_B);
  u3: AND2 port map (NOT_A, NOT_B, Z(0));
  u4: AND2 port map (NOT_A, B, Z(1));
  u5: AND2 port map (A, NOT_B, Z(2));
  u6: AND2 port map (A, B, Z(3));
end STRUCT;
```

Va sottolineato il fatto che il precedente modello è da intendersi come un *esempio* di utilizzo dei costrutti sintattici per la descrizione strutturale dei componenti. Le porte logiche sono infatti descrivibili (più efficentemente e con maggiore leggibilità) mediante gli operatori logici, come mostrato nell'**architecture** DATA_FLOW2 precedentemente riportata.

1.1 Decoder BCD 7 segmenti

Il concetto di decoder può essere generalizzato, ricordando che la codifica è sempre una scelta arbitraria. La codifica BCD (*Binary Coded Decimal*) viene utilizzata per codificare i simboli "0", "1", ..., "9", cioè le cifre utilizzate nella notazione posizionale decimale per la rappresentazione dei numeri. Dato che vi sono dieci simboli da codificare, la codifica BCD consta di dieci diverse configurazioni di una stringa di 4 bit:

numero	*codifica BCD*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

I display a sette segmenti sono comunemente utilizzati per la visualizzazione di valori numerici (vedi fig. 3).

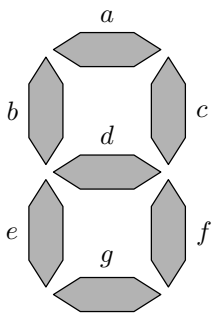

Figura 3: Display a sette segmenti

Un decoder BCD - 7 segmenti è un componente che riceve in ingresso un segnale codificato in BCD e invia in uscita un segnale in grado di attivare la corrispondente configurazione di led. Indicando ognuno dei sette led con un segnale differente, i led possono essere accesi e spenti portando sulla linea corrispondente i valori "1" e "0", rispettivamente. Per esempio, al segnale in ingresso "0010" ("2" codificato in BCD) viene fatto corrispondere il segnale in uscita "1011101" (accesi i led $\{a, c, d, e, g\}$, spenti gli altri).

La tabella di verità del decoder BCD - 7 segmenti è quindi la seguente:

codice BCD				led accesi						
$\mathbf{X}_0$	$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{X}_3$	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{c}$	$\boldsymbol{d}$	$\boldsymbol{e}$	$\boldsymbol{f}$	$\boldsymbol{g}$
0	0	0	0	1	1	1	0	1	1	1
0	0	0	1	0	0	1	0	0	1	0
0	0	1	0	1	0	1	1	0	0	1
0	0	1	1	1	0	1	1	0	1	1
0	1	0	0	0	1	1	1	0	1	0
0	1	0	1	1	1	0	1	0	1	1
0	1	1	0	1	1	0	1	1	1	1
0	1	1	1	1	0	1	0	0	1	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

La descrizione VHDL di tale componente (dove i segnali a, ..., g sono stati rinominati Z(0), ..., Z(6)) è la seguente:

```
entity DEC_BCD_7 is
  port (X: in bit_vector(0 to 3);
        Z: out bit_vector (0 to 6));
end DEC_BCD_7;
```

La descrizione del funzionamento del decoder può essere espressa dal seguente codice:

```
architecture DATAFLOW of DEC_BCD_7 is
```

```
begin
  with X select
    --    abcdefg        BCD
    Z <= "1110111" when "0000",  -- 0
         "0010010" when "0001",  -- 1
         "1011001" when "0010",  -- 2
         "1011011" when "0011",  -- 3
         "0111010" when "0100",  -- 4
         "1101011" when "0101",  -- 5
         "1101111" when "0110",  -- 6
         "1010010" when "0111",  -- 7
         "1111111" when "1000",  -- 8
         "1111011" when "1001",  -- 9
         "1101101" when others;  -- Errore
end DATAFLOW;
```

Da notare che è stata inserita la clausola **when others** per visualizzare un messaggio di errore in caso si presenti in ingresso una configurazione che non corrisponde ad alcun elemento della codifica BCD.

2 Multiplexer

Un multiplexer è un componente dotato di due tipi di linee di ingresso (dati e selezione) e da una linea di uscita. Esso pone sulla linea di uscita il valore di una delle sue linee dati in ingresso. Quest'ultima viene scelta tramite una opportuna combinazione dei valori delle linee di selezione.

Se il multiplexer ha N linee dati, necessita quindi di almeno $\log_2 N$ linee di selezione.

Il multiplexer più semplice è il multiplexer a due vie, dotato di due linee dati e una linea di selezione (fig 4).

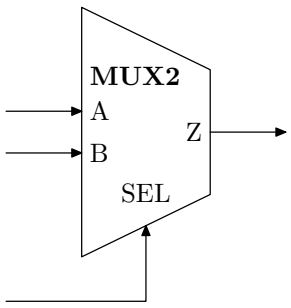

Figura 4: Multiplexer a due vie

La sua tabella di verità è la seguente:

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

La descrizione dell'entità VHDL corrispondente è:

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

La descrizione del comportamento di questo componente può essere espressa tramite il seguente frammento di codice:

```
-- modello comportamentale --
architecture ARC1 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 ARC1;
```

Il corrispondente modello dataflow è il seguente:

```
-- modello data-flow --
architecture DATA_FLOW1 of MUX2 is
begin
  Z <= A when (SEL = '0') else
       B ;
end DATA_FLOW1;
```

Tramite le note tecniche di sintesi di circuiti combinatori, si può verificare che il multiplexer a due vie è equivalente allo schema rappresentato in fig. 5.

```
-- modello data-flow --
architecture DATA_FLOW2 of MUX2 is
begin
  Z <= (A and not SEL) or (B and SEL);
end DATA_FLOW2;
```

Con le stesse considerazioni esposte nel caso del decoder, lo schema circuitale in fig. 5 può essere descritto dalla seguente architettura VHDL:

```
-- modello strutturale --
architecture STRUCT of MUX2 is
```

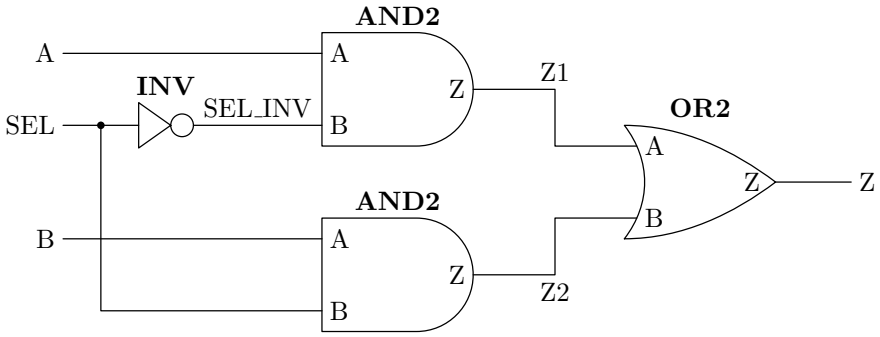

Figura 5: Schema circuitale del multiplexer a due vie.

```
  component INV
    port (A: in bit;
          Z: out bit);
  end component;
  component AND2
    port (A, B: in bit;
              Z: out bit);
  end component;
  component OR2
    port (A, B: in bit;
              Z: out bit);
  end component;

  signal NOT_SEL, Z1, Z2: bit;

begin
  u1: INV port map (SEL, NOT_SEL);
  u2: AND2 port map (A, NOT_SEL, Z1);
  u3: AND2 port map (B, SEL, Z2);
  u4: OR2 port map (Z1, Z2, Z);
end STRUCT;
```

2.1 Multiplexer a 4 vie

Il funzionamento del multiplexer a due vie può essere facilmente esteso ad un numero maggiore di linee dati. In fig. 6 è riportato un multiplexer a quattro vie. Il funzionamento di tale componente è riportato nella seguente tabella:

SEL0	SEL1	A	B	C	D	Z
0	0	0	—	—	—	0
0	0	1	—	—	—	1
0	1	—	0	—	—	0
0	1	—	1	—	—	1
1	0	—	—	0	—	0
1	0	—	—	1	—	1
1	1	—	—	—	0	0
1	1	—	—	—	1	1

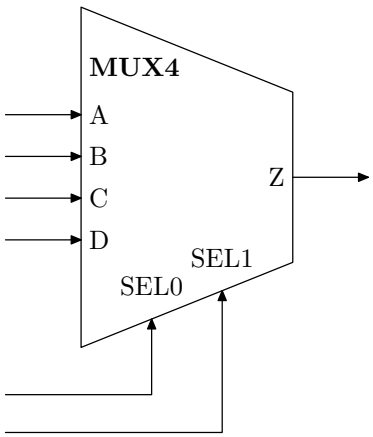

Figura 6: Multiplexer a quattro vie

La descrizione dell'entità VHDL corrispondente è:

```
entity MUX4 is
  port (A, B, C, D, SEL0, SEL1: in bit;
                             Z: out bit);
end MUX4;
```

La descrizione del comportamento di questo componente può essere espressa tramite il seguente frammento di codice VHDL:

```
  -- modello comportamentale --
  architecture ARC1 of MUX4 is
  begin
    P1: process (A, B, C, D, SEL0, SEL1)
    begin
    if (SEL0 = '0' and SEL1 = '0') then
      Z <= A;
    elsif (SEL0 = '0' and SEL1 = '1') then
      Z <= B;
    elsif (SEL0 = '1' and SEL1 = '0') then
      Z <= C;
    elsif (SEL0 = '1' and SEL1 = '1') then
```

```
      Z <= D;
    end if;
  end process P1;
end ARC1;
```

Lo stesso comportamento può essere descritto in maniera più appropriata usando il costrutto case-when e l'operatore di aggregazione &:

```
-- modello comportamentale --
architecture ARC2 of MUX4 is
begin
  P2: process (A, B, C, D, SEL0, SEL1)
  begin
    case (SEL0 & SEL1) is
      when "00" =>
        Z <= A;
      when "01" =>
        Z <= B;
      when "10" =>
        Z <= C;
      when "11" =>
        Z <= D;
    end case;
  end process P2;
end ARC2;
```

Il seguente codice descrive il modello dataflow di un multiplexer a quattro vie:

```
-- modello data-flow --
architecture DATA_FLOW1 of MUX4 is
begin
  Z <= A when (SEL0 = '0' and SEL1 = '0') else
       B when (SEL0 = '0' and SEL1 = '1') else
       C when (SEL0 = '1' and SEL1 = '0') else
       D;
end DATA_FLOW1;
```

Sfruttando le tecniche di sintesi di circuiti combinatori, si può ottenere il circuito in fig. 7, descritto dal seguente modello dataflow:

```
-- modello data-flow --
architecture DATA_FLOW2 of MUX4 is
begin
  Z <= (A and not SEL0 and not SEL1)or (B and not SEL0 and SEL1)
       or (C and SEL0 and not SEL1) or (D and SEL0 and SEL1) ;
end DATA_FLOW2;
```

Per puro esercizio, lo schema in fig. 7 può essere descritto utilizzando il un modello VHDL strutturale:

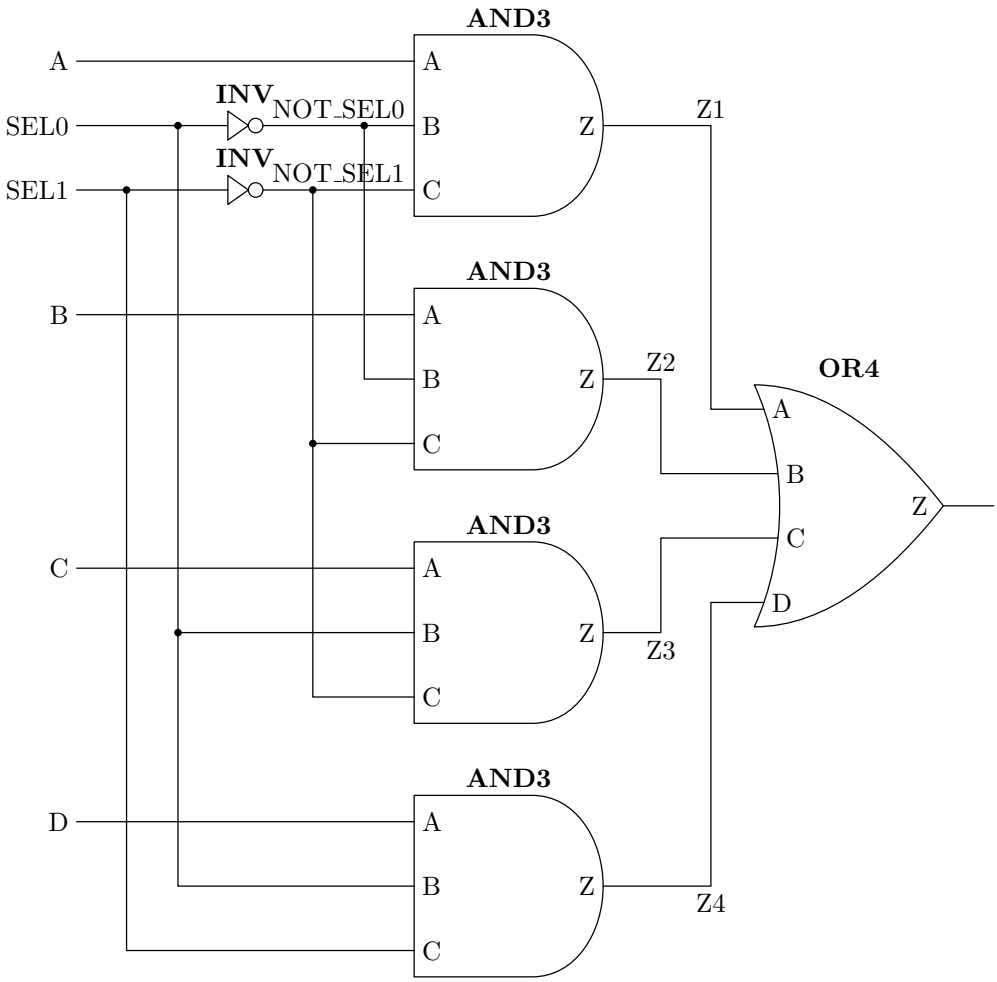

Figura 7: Schema circuitale del multiplexer a quattro vie.

```
-- modello strutturale --
architecture STRUCT1 of MUX4 is
  component INV
    port (A: in bit;
          Z: out bit);
  end component;
  component AND3
    port (A, B, C: in bit;
          Z: out bit);
  end component;
  component OR4
    port (A, B, C, D: in bit;
          Z: out bit);
  end component;
  signal NOT_SEL0, NOT_SEL1, Z1, Z2, Z3, Z4: bit;
begin
  u1: INV port map (SEL0, NOT_SEL0);
  u2: INV port map (SEL1, NOT_SEL1);
  u3: AND3 port map (A, NOT_SEL0, NOT_SEL1, Z1);
  u4: AND3 port map (B, NOT_SEL0, SEL1, Z2);
  u5: AND3 port map (C, SEL0, NOT_SEL1, Z3);
  u6: AND3 port map (D, SEL0, SEL1, Z4);
  u7: OR4 port map (Z1, Z2, Z3, Z4, Z);
end STRUCT1;
```

La fig. 8 illustra come si possano comporre tre multiplexer a due vie per ottenere un multiplexer a quattro vie. Due multiplexer, entrambi pilotati dalla prima linea di selezione, lasciano passare solo due dei segnali dati, i quali vanno in ingresso al terzo multiplexer. Quest'ultimo è pilotato dalla seconda linea di selezione, la quale stabilisce quale dei due segnali dovrà essere portato in uscita.

Il codice VHDL che descrive questo circuito può giovarsi della descrizione strutturale:

```
-- modello strutturale --
architecture STRUCT2 of MUX4 is
  component MUX2
    port (A, B, SEL: in bit;
          Z: out bit);
  end component;
  signal Z1, Z2: bit ;
begin
  u1: MUX2 port map (A, B, SEL1, Z1);
  u2: MUX2 port map (C, D, SEL1, Z2);
  u3: MUX2 port map (Z1, Z2, SEL0, Z);
end STRUCT2;
```

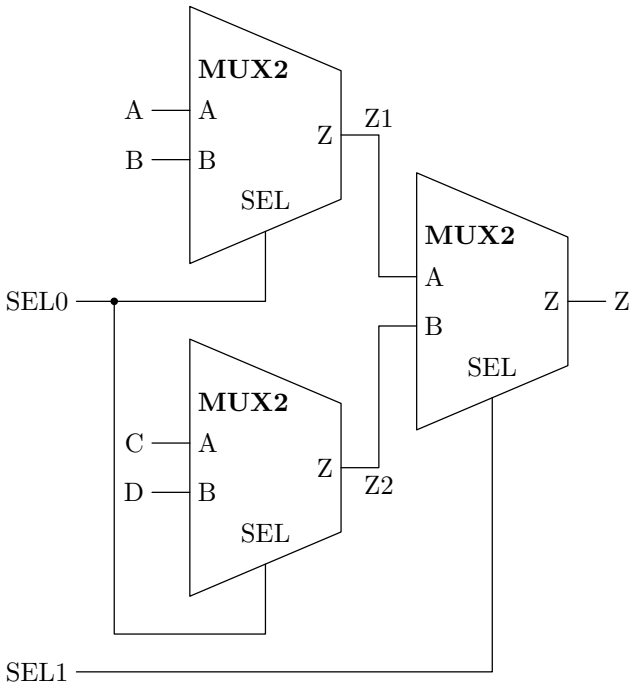

Figura 8: MUX4 costruito assemblando tre MUX2