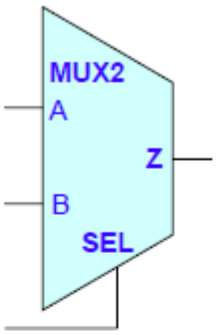
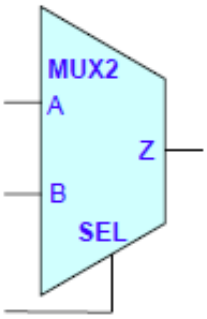
**1.** Scrivere il modello VHDL *(comportamentale e dataflow)* del componente **MUX2** descritto dalla sequente specifica:

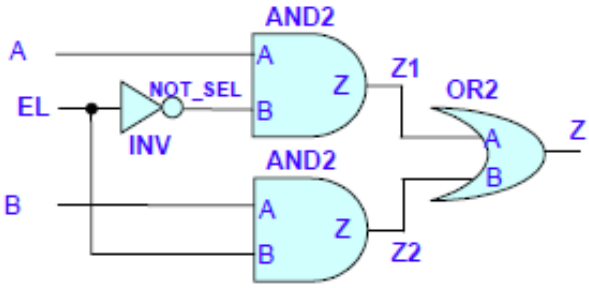



| Ingressi | | | Uscita |
|---|---|---|---|
| SEL | A | B | Z |
| 0 | 0 | - | 0 |
| 0 | 1 | - | 1 |
| 1 | - | 0 | 0 |
| 1 | - | 1 | 1 |

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

-- 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;

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

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

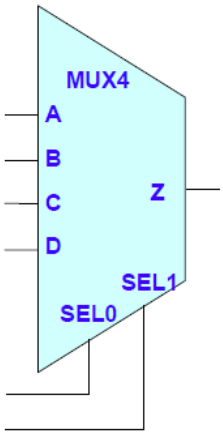
Scrivere il modello VHDL *(strutturale)* del componente **MUX2** descritto dalla sequente specifica:



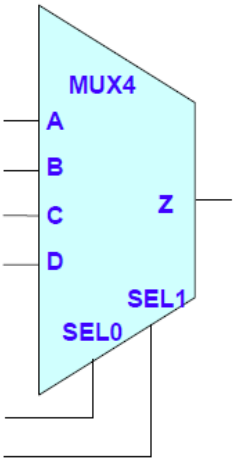

```
entity MUX2 is
port ( A, B, SEL: in bit;
       Z: out bit);
end MUX2;

-- modello strutturale --
architecture STRUCT of MUX2 is

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;
```

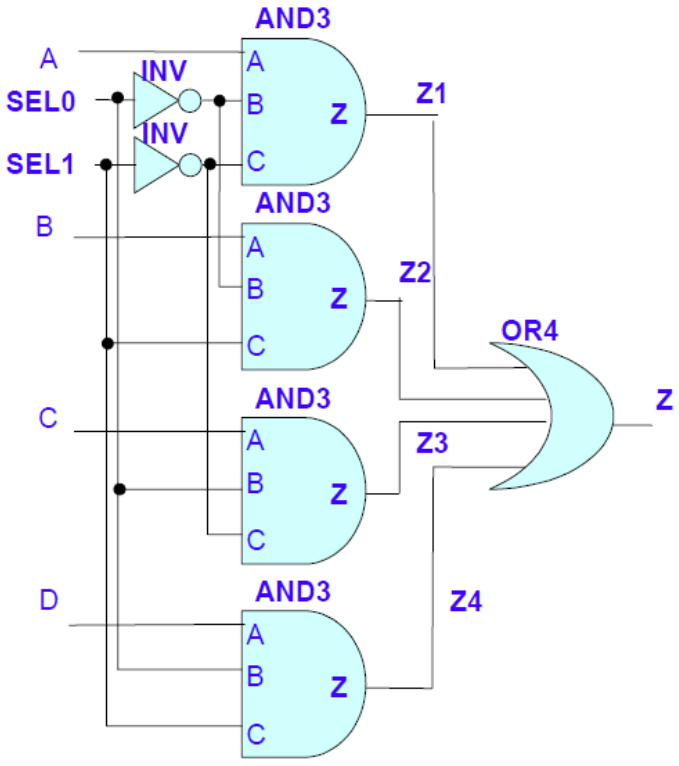
**3.** Scrivere il modello VHDL *(comportamentale e dataflow)* del componente **MUX4** descritto dalla sequente specifica:



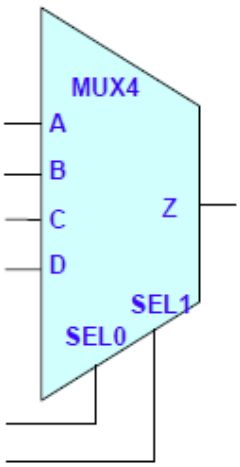

| Ingressi | | | | | | Uscita |
|---|---|---|---|---|---|---|
| SEL0 | SEL1 | A | B | C | D | Z |
| 0 | 0 | 0 | - | - | - | 0 |
| 0 | 0 | 1 | - | - | - | 1 |
| 0 | 1 | - | 0 | - | - | 0 |
| 0 | 1 | - | 1 | - | - | 1 |
| 1 | 0 | - | - | 0 | - | 0 |
| 1 | 0 | - | - | 1 | - | 1 |
| 1 | 1 | - | - | - | 0 | 0 |
| 1 | 1 | - | - | - | 1 | 1 |

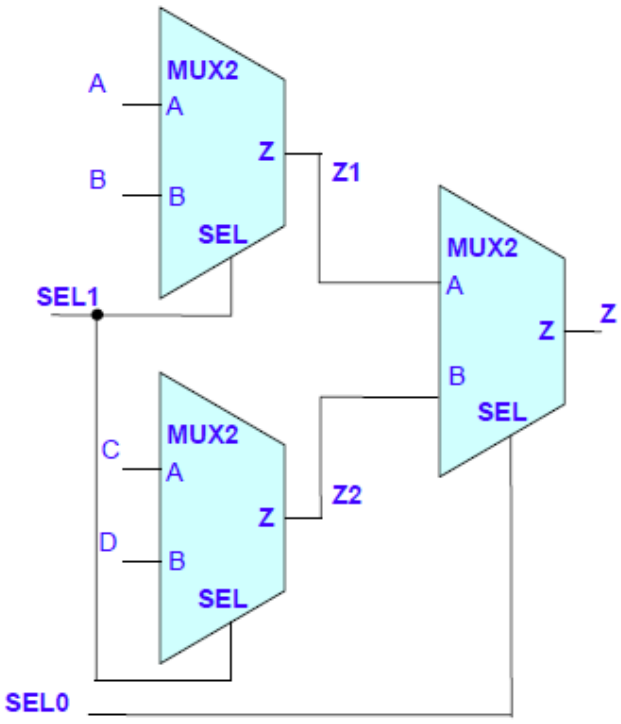
Scrivere il modello VHDL *(strutturale)* del componente **MUX4** descritto dalla sequente specifica:



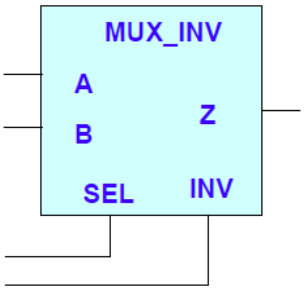



Scrivere il modello VHDL *(strutturale)* del componente **MUX4** descritto dalla sequente specifica:



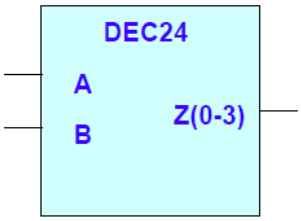




**6.** Scrivere il modello VHDL *(comportamentale e dataflow)* del componente **MUX_INV** descritto dalla seguente specifica:



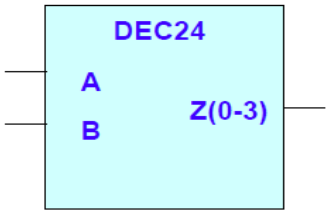

| Ingressi | | | | Uscita |
|---|---|---|---|---|
| SEL | INV | A | B | Z |
| 0 | 0 | 0 | - | 0 |
| 0 | 0 | 1 | - | 1 |
| 0 | 1 | 0 | - | 1 |
| 0 | 1 | 1 | - | 0 |
| 1 | 0 | - | 0 | 0 |
| 1 | 0 | - | 1 | 1 |
| 1 | 1 | - | 0 | 1 |
| 1 | 1 | - | 1 | 0 |

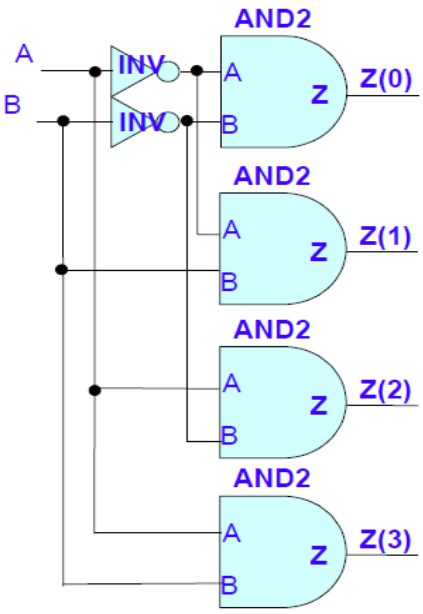
**7.** Scrivere il modello VHDL *(comportamentale e dataflow)* del componente **DEC24** descritto dalla seguente specifica:



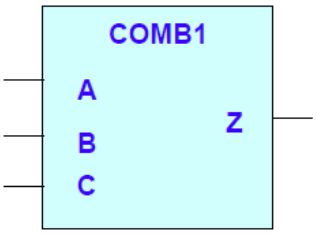

| Ingressi | | Uscita | | | |
|---|---|---|---|---|---|
| A | B | Z(0) | Z(1) | Z(2) | Z(3) |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

Scrivere il modello VHDL *(strutturale)* del componente **DEC24** descritto dalla seguente specifica:



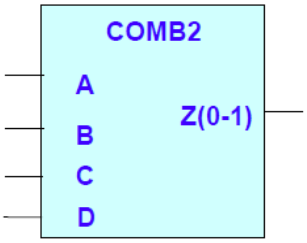



**9.** Scrivere il modello VHDL *(comportamentale)* del componente **COMB1** descritto dalla sequente specifica:



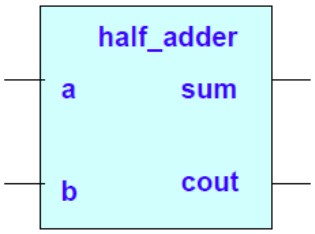

| Ingressi | | | Uscita |
|---|---|---|---|
| A | B | C | Z |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

**10.** Scrivere il modello VHDL *(comportamentale)* del componente **COMB2** descritto dalla sequente specifica:



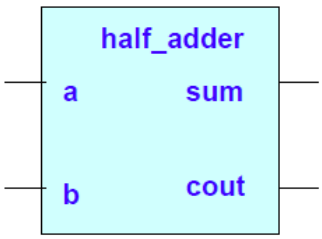

| Ingressi | | | | Uscite | |
|---|---|---|---|---|---|
| A | B | C | D | Z(0) | Z(1) |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | - | - | - | 0 | 1 |

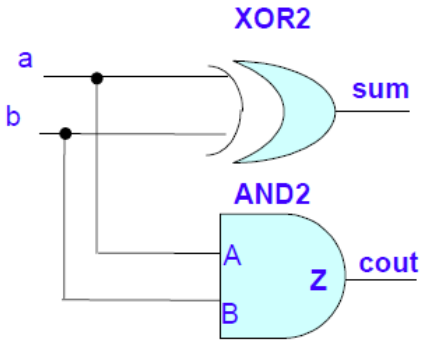
**11.** Scrivere il modello VHDL *(comportamentale e dataflow)* del componente **HALF_ADDER** descritto dalla sequente specifica:



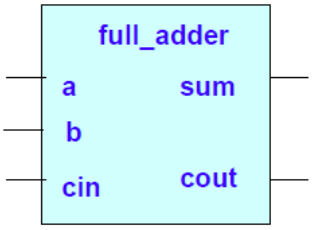

| Ingressi | | Uscite | |
|---|---|---|---|
| a | b | sum | cout |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Scrivere il modello VHDL *(strutturale)* del componente **HALF_ADDER** descritto dalla sequente specifica:



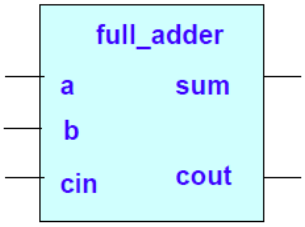



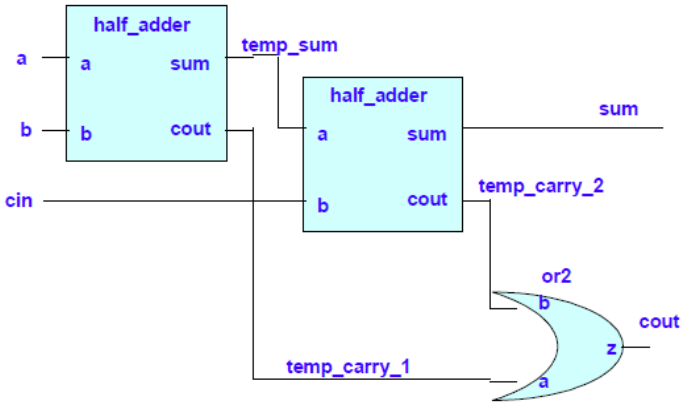
**13.** Scrivere il modello VHDL *(comportamentale e dataflow)* del componente **FULL_ADDER** descritto dalla sequente specifica:




| Ingressi | | | Uscite | |
|---|---|---|---|---|
| a | b | cin | sum | cout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Scrivere il modello VHDL *(strutturale)* del componente **FULL_ADDER** descritto dalla sequente specifica:



## 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.

## 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;
```

## 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;
```

### Esercizio

Si realizzi in VHDL un modello comportamentale di un componente combinatorio con 3 parole in ingresso $a, b, c$ ciascuna delle quali contiene 4 bit che rappresentano interi senza segno. La rete ha una parola $y$ in uscita anch'essa di 4 bit che assume un valore uguale al massimo fra $a$, $b$ e $c$. I dati in ingresso e in uscita devono essere codificati con il tipo di dato `std_logic_vector`.

# 1. 4-bit Unsigned Up Counter with Asynchronous Clear

The following table shows pin definitions for a 4-bit unsigned up counter with asynchronous clear.

| IO Pins | Description |
|---|---|
| C | Positive-Edge Clock |
| CLR | Asynchronous Clear (active High) |
| Q[3:0] | Data Output |

# 2. 4-bit Unsigned Up/Down counter with Asynchronous Clear

The following table shows pin definitions for a 4-bit unsigned up/down counter with asynchronous clear.

| IO Pins | Description |
|---|---|
| C | Positive-Edge Clock |
| CLR | Asynchronous Clear (active High) |
| UP_DOWN | up/down count mode selector |
| Q[3:0] | Data Output |

## 3. 4-bit Unsigned Up Counter with Asynchronous Load from Primary Input

The following table shows pin definitions for a 4-bit unsigned up counter with asynchronous load from primary input.

| IO Pins | Description |
|---|---|
| C | Positive-Edge Clock |
| ALOAD | Asynchronous Load (active High) |
| D[3:0] | Data Input |
| Q[3:0] | Data Output |

## 4. 8-bit Shift-Left Register with Positive-Edge Clock, Asynchronous Clear, Serial In, and Serial Out

The following table shows pin definitions for an 8-bit shift-left register with a positive-edge clock, asynchronous clear, serial in, and serial out.

| IO Pins | Description |
|---|---|
| C | Positive-Edge Clock |
| SI | Serial In |
| CLR | Asynchronous Clear (active High) |
| SO | Serial Output |

## 5. 8-bit Shift-Left/Shift-Right Register with Positive-Edge Clock, Serial In, and Parallel Out

The following table shows pin definitions for an 8-bit shift-left/shift-right register with a positive-edge clock, serial in, and serial out.

| IO Pins | Description |
|---|---|
| C | Positive-Edge Clock |
| SI | Serial In |
| LEFT_RIGHT | Left/right shift mode selector |
| PO[7:0] | Parallel Output |

## 6. Logical shifter

The following table shows pin descriptions for a logical shifter.

| IO pins | Description |
|---|---|
| D[7:0] | Data Input |
| SEL | shift distance selector |
| SO[7:0] | Data Output |