

---

# *Architetture e Reti logiche*

## *— Esercitazioni VHDL —*

*a.a. 2003/04*

◇ *VHDL* ◇

Stefano Ferrari



Università degli Studi di Milano

Dipartimento di Tecnologie dell'Informazione

- Lo sviluppo di un modello VHDL a partire dalla specifica concettuale avviene attraverso un processo di raffinamento basato sulla ripetizione delle fasi di compilazione, analisi e simulazione.
  - **Compilazione** → **Analisi** → **Simulazione** → **Analisi** → **Compilazione** ...
- La specifica concettuale consiste in una descrizione:
  - dell' *interfaccia* del componente;
  - della *funzionalità* del componente.

- L'unità di base di un modello VHDL consiste nella Design Entity, che può rappresentare un intero sistema, un circuito stampato, un circuito integrato oppure una porta logica elementare.
- La *Entity Declaration* definisce l'interfaccia del modello.
- L'*Architecture Body* definisce la funzionalità del modello.
- All'interno di un modello VHDL, ad una stessa Entity Declaration possono corrispondere diverse Architecture Body.
- Un modello VHDL può essere creato a diversi livelli di astrazione (*behavioral, dataflow, structural*) secondo un processo di raffinamento del modello iniziale.

**Descrizione comportamentale** o behavioral: supporta descrizioni algoritmiche;

**Descrizione flusso dati** o dataflow: supporta descrizioni a livello di trasferimento del flusso dati tra registri.

**Descrizione strutturale** o structural: supporta descrizioni di strutture composte dall'interconnessione di componenti di livello gerarchico inferiore.

**Descrizione mista**

# Struttura di un modello VHDL

---

- Ad una singola interfaccia di un modello VHDL possono corrispondere diverse Architecture Body ciascuna descritta ad un diverso livello di astrazione (comportamentale, dataflow, strutturale o mista)
- Ogni diversa architettura rappresenta una diversa realizzazione della stessa funzionalità del modello per mettere in luce un diverso aspetto progettuale cioè fornisce una diversa implementazione di una stessa funzionalità.

- Supporta descrizione della funzionalità del modello a diversi livelli di astrazione.
- Concorrenza: l'hardware è intrinsecamente concorrente e composto dall'interconnessione di componenti elementari.
- Il concetto di concorrenza supportato sia dai modelli strutturali sia dai processi multipli (istruzioni sequenziali all'interno di un processo.)
- Gerarchia: data la complessità progettuale occorre organizzare il progetto su diversi livelli gerarchici, che possono essere descritti a diversi livelli di astrazione.
- Temporizzazioni: necessità di modellizzare l'andamento temporale dei segnali attraverso la descrizione di forme d'onda.

- La specifica ad alto livello consiste in generale in una descrizione della funzionalità desiderata.
- Esempio:
  - Creare una entity che riceve in ingresso due segnali digitali e genera in uscita un singolo segnale.
  - Se entrambi i segnali di ingresso sono bassi, il segnale di uscita deve essere alto.
  - Per ogni altra combinazione degli ingressi, l'uscita deve essere bassa.

Generazione della tabella della verità:

Ingressi		Uscita
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

- Definisce una entity e la sua interfaccia con l'ambiente esterno, non definisce la funzionalità del modello.
- Formato:

```
entity ENTITY_NAME is  
    port (PORT_LIST); -- lista dei segnali  
                                -- di interfaccia;  
end [ENTITY_NAME];
```

- Esempio: Definizione della entity `nor_gate`.

```
entity NOR_GATE is  
    port (A, B : in bit;  
          C : out bit);  
end NOR_GATE;
```

- Identifica l'insieme dei segnali di interfaccia della entity per comunicare con l'ambiente esterno.
- Formato:

```
port (NAME_LIST: mode type;  
      . . .  
      NAME_LIST: mode type);
```

- `mode` identifica la direzione del segnale cioè la direzione del flusso dati attraverso una porta (`in`, `out`, `inout`).
- `type` identifica il tipo del segnale cioè un insieme di valori che il segnale può assumere (esempio 0 e 1) e un insieme di operazioni che possono essere eseguite sul segnale.

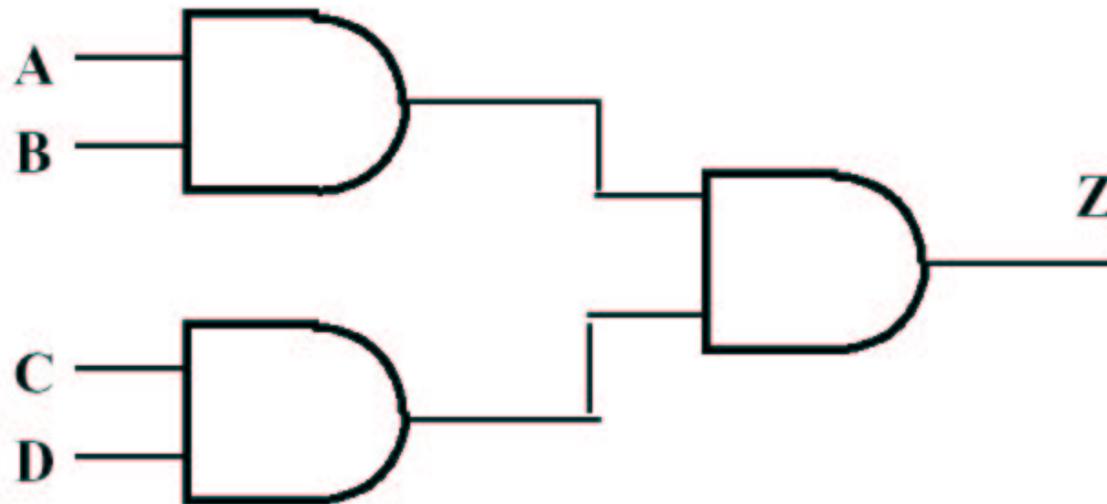
- Descrive la funzionalità del modello attraverso la definizione delle relazioni funzionali tra gli ingressi e le uscite di un modello.
- Formato:

```
architecture BODY_NAME of ENTITY_NAME is  
    -- istruzioni dichiarative  
begin  
    -- istruzioni per descrivere la  
    -- funzionalità del modello  
end [BODY_NAME];
```

- Esempio: Definizione della architecture `nor_gate`.

```
architecture DATA_FLOW of NOR_GATE is  
begin  
    C <= A nor B;  
end DATA_FLOW;
```

- Le strutture hardware sono intrinsecamente concorrenti e composte dall'interconnessione di componenti elementari.
- Le attività sono svolte in parallelo dai diversi componenti.
- Una descrizione strutturale descrive l'interconnessione tra componenti elementari.



```
entity AND4 is
    port (A, B, C, D: in bit;
          Z: out bit);
end AND4;

architecture DATA_FLOW of AND4 is
    signal T1, T2: bit;
begin
    T1 <= A and B;
    T2 <= C and D;
    Z <= T1 and T2;
end DATA_FLOW;
```

**Costanti** Oggetti assegnati ad un valore iniziale che non può essere modificato

**Variabili** Oggetti che ricevono un valore che può essere cambiato attraverso un istruzione di assegnamento tra variabili:

```
OLD_VALUE := NEW_VALUE ;
```

**Segnali** Oggetti che ricevono dei valori associati ad un fattore temporale e che possono essere cambiati attraverso un istruzione di assegnamento tra segnali:

```
OLD_VALUE <= NEW_VALUE after 1 ms ;
```

- I segnali sono caratteristici della descrizione hardware per modellare l'informazione che transita nel tempo o su una singola net oppure su un bus.
- L'istruzione di assegnamento tra segnali stabilisce un collegamento definitivo tra di essi. Esempio:

```
signal A, Z : bit;
```

```
Z <= A;
```

- In presenza di istruzioni di assegnamento multiple è necessaria una funzione di risoluzione :

```
signal A, B, Z: bit;
```

```
Z <= A;
```

```
Z <= B;
```

Nel package STD.STANDARD:

- BOOLEAN (false, true)
- BIT ('0', '1')
- INTEGER (intervallo che dipende dall'implementazione)
- tipi di dati definiti dall'utente
- Il VHDL è un linguaggio fortemente tipizzato.
- L'assegnamento tra segnali di diverso tipo richiede l'uso di appositi convertitori.

# Accesso a tipi comunemente usati

---

- La condivisione da parte di diverse entity di dichiarazioni di tipi e di operazioni tra tipi viene fatta attraverso dei costrutti noti come package.
- Tipi, procedure e funzioni comunemente usate sono poste in package standard come Std\_Logic\_1164 (IEEE standard).
- La dichiarazione library rende visibile ad un modello VHDL una libreria selezionata che contiene i package desiderati.
- La dichiarazione use rende un package visibile ad un modello.
- Esempio:

```
library IEEE;
```

```
use IEEE.Std_Logic_1164.all;
```

Nel package IEEE.STD\_LOGIC\_1164:

- STD\_ULOGIC ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-')
- STD\_LOGIC ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-')

STD\_LOGIC risolve i conflitti (U sta per unresolved!)

	'U'		'U' undefined
	'X'		'X' forcing unknown
'0'	'-'	'1'	'0' forcing 0
	'W'		'1' forcing 1
'L'		'H'	'Z' high impedance
	'Z'		'W' weak unknown
			'L' weak 0
			'H' weak 1
			'-' don't care

## vector

```
port (a: in std_logic_vector(0 to 3);  
      b: out bit_vector(3 downto 1));
```

## array

```
type T_DATA is array (0 to 3) of  
    std_logic_vector(7 downto 0);  
constant DATA : T_DATA := ("00000000",  
                             "00000001",  
                             "00000010",  
                             "00000100");
```

Nota: per costanti binarie si usano gli apici ('1'), mentre per i vettori di due o più bit si usano le doppie virgolette ("001")

- L'assegnamento:

```
signal Z_BUS: bit_vector(3 downto 0);  
signal A_BUS: bit_vector(1 to 4);  
-- ...  
Z_BUS <= A_BUS;
```

- equivale a:

```
Z_BUS(3) <= A_BUS(1); -- referenziazione per indici  
Z_BUS(2) <= A_BUS(2);  
Z_BUS(1) <= A_BUS(3);  
Z_BUS(0) <= A_BUS(4);
```

- equivale a:

```
Z_BUS(3 downto 1) <= A_BUS(1 to 3); -- bit-slicing  
Z_BUS(0) <= A_BUS(4);
```

I segnali si possono comporre per formare un bus tramite due operatori:

- Concatenazione (solo sul lato destro dell'assegnamento):

```
architecture EXAMPLE of CONCATENATION is  
signal BYTE : bit_vector (8 downto 0);  
signal A_BUS, B_BUS : bit_vector (3 downto 0);  
signal C : bit;  
begin  
    BYTE    <= A_BUS & C & B_BUS;  
end EXAMPLE;
```

- **Aggregazione:**

```
architecture EXAMPLE of AGGREGATES is
```

```
    signal BYTE : bit_vector (7 downto 0);
```

```
    signal Z_BUS : bit_vector (3 downto 0);
```

```
    signal A_BIT, B_BIT, C_BIT, D_BIT : bit;
```

```
begin
```

```
    Z_BUS <= ( A_BIT, B_BIT, C_BIT, D_BIT );
```

```
    ( A_BIT, B_BIT, C_BIT, D_BIT ) <= bit_vector("1011");
```

```
    ( A_BIT, B_BIT, C_BIT, D_BIT ) <= BYTE(3 downto 0);
```

```
    BYTE <= ( 7 => '1', 5 downto 1 => '1', 6 => B_BIT, others => '0' ) ;
```

```
end EXAMPLE;
```

Operatori definiti su  
BOOLEAN, BIT, BIT\_VECTOR, STD\_LOGIC,  
STD\_LOGIC\_VECTOR

logici            AND NAND OR NOR XOR NOT

relazionali    = /= < > <= >=

Gli operatori relazionali sui vettori sono definiti bit a bit, allineati a destra:

“111” > “1011” è vero

Disegnare lo schematico corrispondente

```
ENTITY example1 IS
```

```
    PORT ( x1, x2, x3 : IN BIT;  
           f : OUT BIT );
```

```
END example1;
```

```
ARCHITECTURE LogicFunc OF example1 IS
```

```
BEGIN
```

```
    f <= (x1 AND x2) OR (NOT x2 AND x3);
```

```
END LogicFunc;
```

Disegnare lo schematico e costruire la tabella di verità

```
ENTITY fulladd IS
```

```
    PORT (Cin, x, y : IN STD_LOGIC;  
          s, Cout : OUT STD_LOGIC );
```

```
END fulladd;
```

```
ARCHITECTURE LogicFunc OF fulladd IS
```

```
BEGIN
```

```
    s <= x XOR y XOR Cin;
```

```
    Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y);
```

```
END LogicFunc;
```

```
ENTITY latch IS
    port ( s, r: in bit;
           q, nq: out bit);
END latch;

ARCHITECTURE structure of latch IS
    component nor_gate
        port (a,b: in bit;
              c: out bit);
    end component;
BEGIN
    n1: nor_gate port map (r, nq, q);
    n2: nor_gate port map (b=>q, a=>s, c=>nq);
END structure;
```

```
ENTITY adder3 IS
  PORT (
    Cin : IN STD_LOGIC;
    x2, x1, x0 : IN STD_LOGIC;
    y2, y1, y0 : IN STD_LOGIC;
    s2, s1, s0 : OUT STD_LOGIC;
    Cout : OUT STD_LOGIC);
END adder3;

ARCHITECTURE Structure OF adder3 IS
  SIGNAL c1, c2 : STD_LOGIC;
BEGIN
  stage0: fulladd PORT MAP(Cin,x0,y0,s0,c1);
  stage1: fulladd PORT MAP(c1,x1,y1,s1,c2);
  stage2: fulladd PORT MAP (Cin=>c2, Cout=>Cout, x=>x2, y=>y2, s=>s2);
END Structure;
```

```
ENTITY adder3 IS
  PORT (Cin : IN STD_LOGIC;
        X, Y : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        S : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
        Cout : OUT STD_LOGIC ) ;
END adder3;

ARCHITECTURE Structure OF adder3 IS
  SIGNAL C : STD_LOGIC_VECTOR(1 TO 2) ;
BEGIN
  stage0: fulladd PORT MAP (Cin, X(0), Y(0), S(0), C(1));
  stage1: fulladd PORT MAP (C(1), X(1), Y(1), S(1), C(2));
  stage2: fulladd PORT MAP (C(2), X(2), Y(2), S(2), Cout);
END Structure;
```

- Il costrutto `WITH-SELECT-WHEN` è utilizzabile all'interno di una descrizione concorrente
- Sintassi:

```
WITH <expression> SELECT  
  <signal_name> <= <signal_or_value> WHEN <condition1>,  
    <signal_or_value> WHEN <condition2>,  
    ...  
    <signal_or_value> WHEN OTHERS;
```

- le condizioni devono essere mutuamente esclusive
- tutti i valori possibili dell'espressione devono essere coperti

- Esempio:

```
ENTITY and2 IS
    PORT (a : IN BIT_VECTOR (1 DOWNT0 0);
          y : OUT BIT);
END and2;

ARCHITECTURE arch_and2 OF and2 IS
BEGIN
    WITH a SELECT
        y <= '1' WHEN "11",
             '0' WHEN "00",
             '0' WHEN "01",
             '0' WHEN "10";
END arch_and2;
```

- L'uso di `OTHERS` rende la descrizione più sintetica e leggibile, come si può apprezzare all'aumentare delle possibili combinazioni:

```
ENTITY and12 IS
    PORT (a : IN BIT_VECTOR (11 DOWNT0 0);
          y : OUT BIT);
END and12;
```

```
ARCHITECTURE arch_and12 OF and12 IS
BEGIN
    WITH a SELECT
        y <= '1' WHEN "111111111111",
            '0' WHEN OTHERS;
END arch_and12;
```

# Assegnamento condizionato

- Il costrutto **WHEN-ELSE** è utilizzabile all'interno di una descrizione concorrente
- Sintassi:

```
<signal_name> <= <signal_or_value> WHEN <cond1> ELSE  
    <signal_or_value> WHEN <cond2> ELSE  
    -- ...  
    <signal_or_value> WHEN <condN> ELSE  
    <signal_or_value>;
```

- le condizioni vengono valutate in sequenza e ci si ferma alla prima verificata
- deve sempre esistere un assegnamento non condizionato

- Esempio:

```
ENTITY and2 IS
```

```
    PORT (a, b : IN BIT;  
          y : OUT BIT);
```

```
END and2;
```

```
ARCHITECTURE arch_and2 OF and2 IS
```

```
BEGIN
```

```
    y<='1' WHEN (A='1' AND B='1') ELSE  
        '0';
```

```
END arch_and2;
```

# Assegnamento condizionato (3)

---

- C'è sempre più di un modo per descrivere la stessa cosa:

```
ENTITY and2 IS
```

```
    PORT (a: IN BIT_VECTOR (1 DOWNTO 0));
```

```
        y : OUT BIT);
```

```
END and2;
```

```
ARCHITECTURE arch_and2_bis OF and2 IS
```

```
BEGIN
```

```
    y<='0' WHEN a(0)='0' ELSE
```

```
        '0' WHEN a(1)='0' ELSE
```

```
        '1';
```

```
END arch_and2_bis;
```

# Assegnamento condizionato (4)

---

- oppure:

```
ENTITY and2 IS
```

```
    PORT (a: IN BIT_VECTOR (1 DOWNTO 0);
```

```
           y : OUT BIT);
```

```
END and2;
```

```
ARCHITECTURE arch_and2_tris OF and2 IS
```

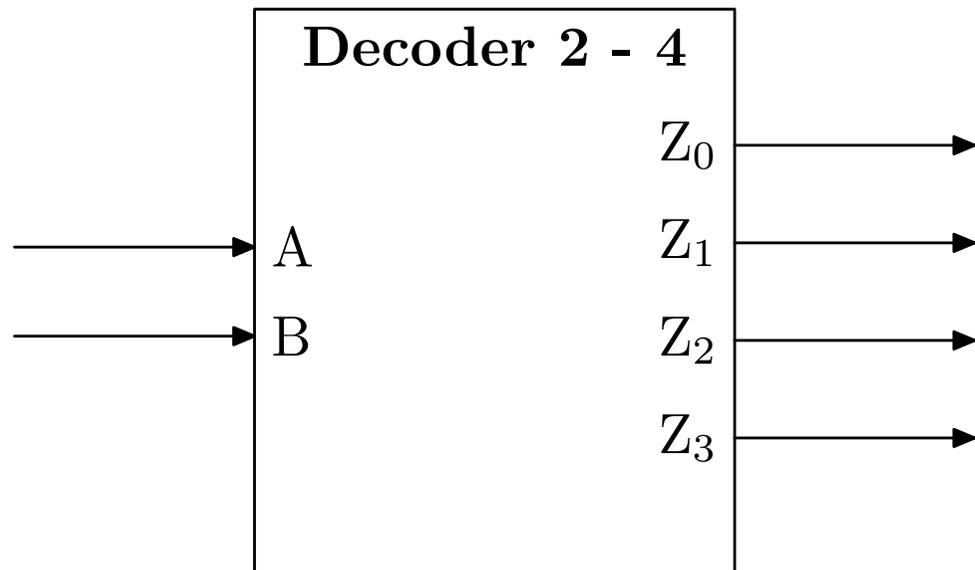
```
BEGIN
```

```
    y<='1' WHEN a='11' ELSE
```

```
        '0';
```

```
END arch_and2_tris;
```

# Esempio: Decoder a due ingressi



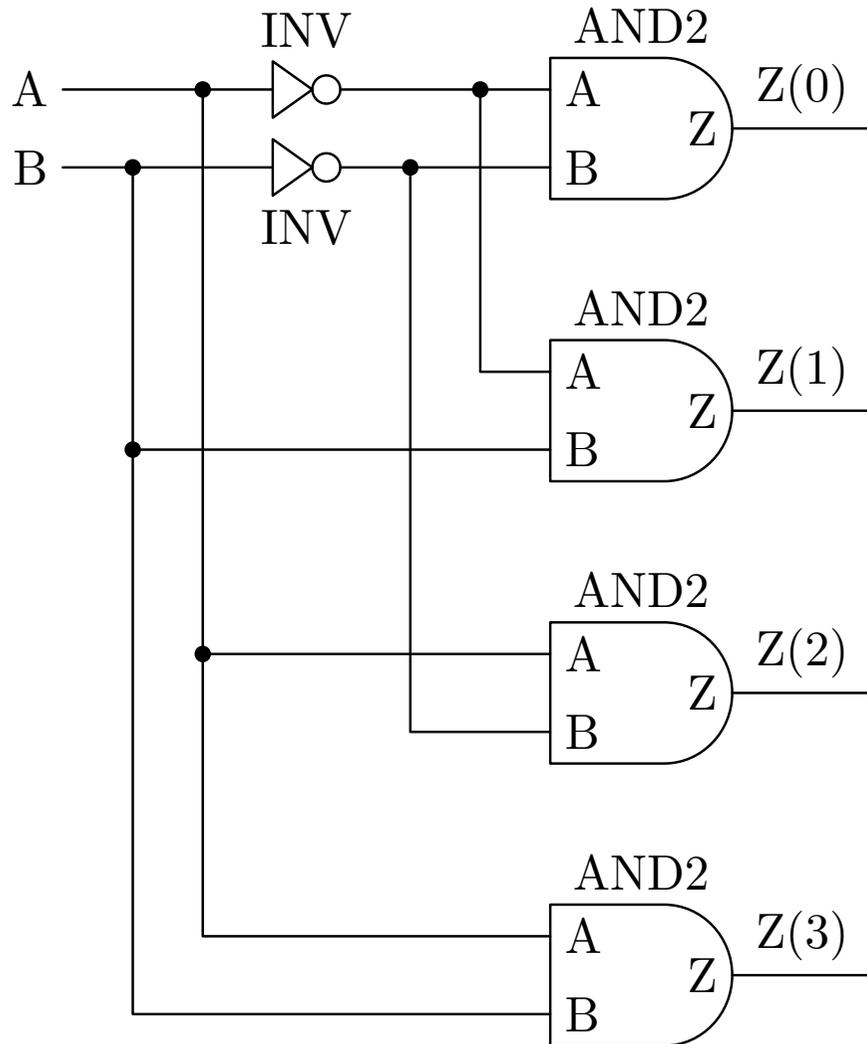
A	B	Z <sub>0</sub>	Z <sub>1</sub>	Z <sub>2</sub>	Z <sub>3</sub>
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

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

```
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;
```

```
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;
```

# Decoder: rappresentazione circuitale



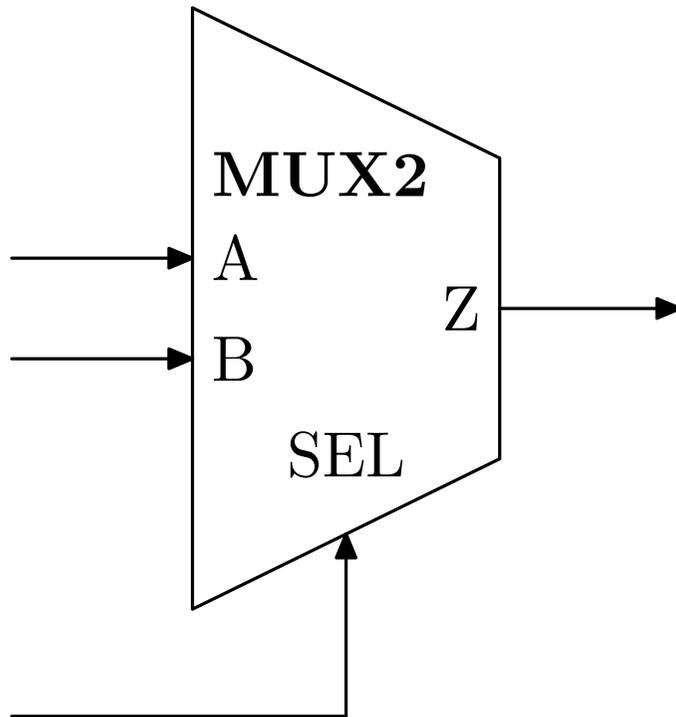
```
architecture DATA_FLOW3 of DEC24 is
begin
    Z(0) <= (not A and not B);
    Z(1) <= (not A and B);
    Z(2) <= (A and not B);
    Z(3) <= (A and B);
end DATA_FLOW3;
```

# Decoder: descrizione 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;
```

# Esempio: Multiplexer a due vie



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;
```

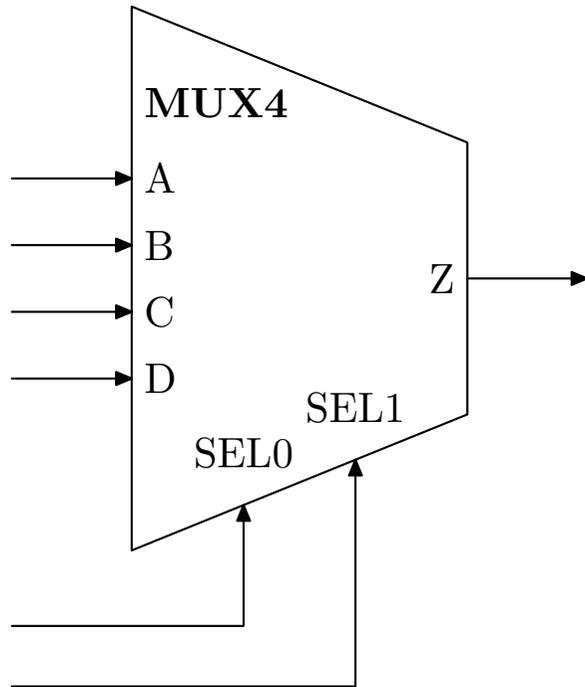
```
          Z: out bit);
```

```
end MUX2;
```

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

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

# Esempio: Multiplexer a 4 vie



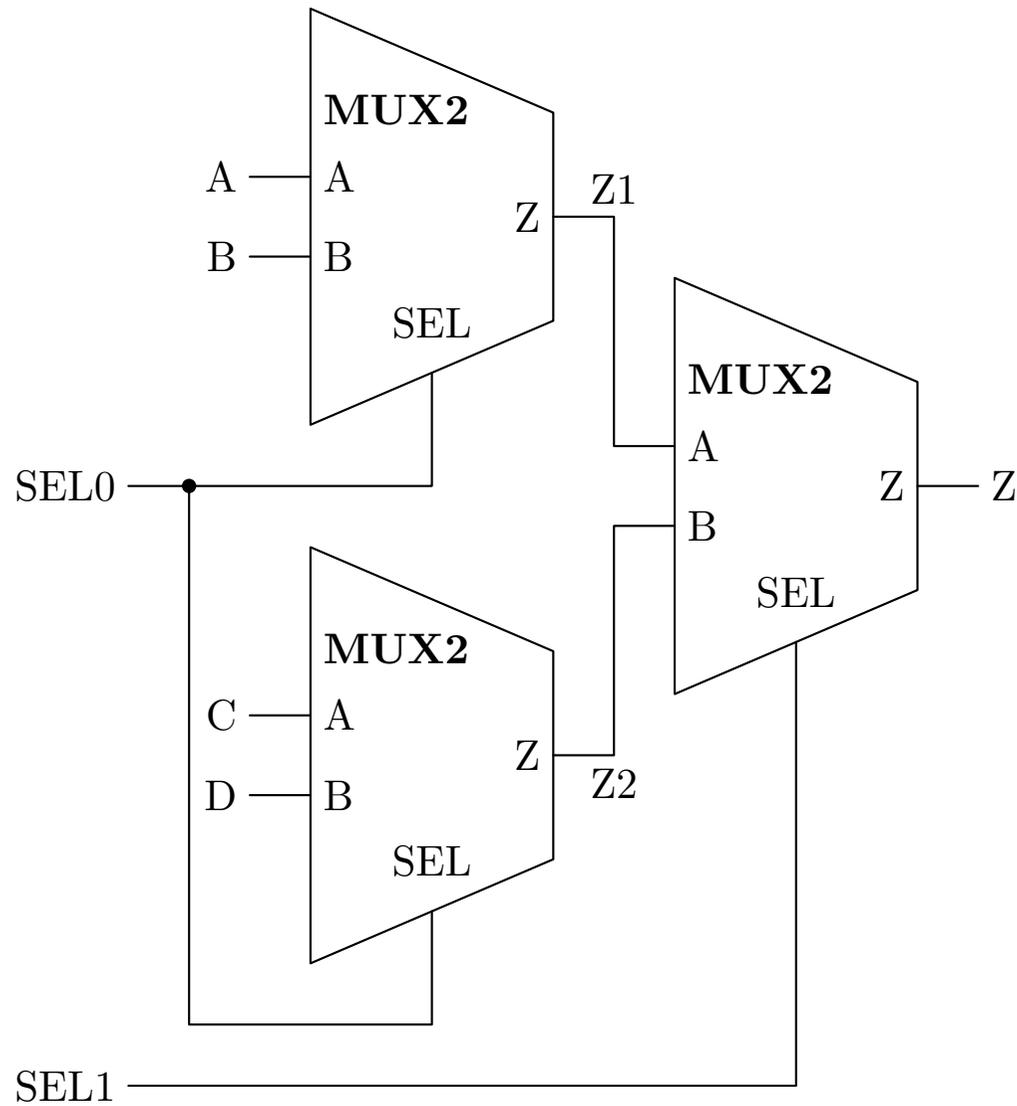
SEL0	SEL1	A	B	C	D	Z
0	0	0	—	—	—	0
0	0	1	—	—	—	1
0	1	—	0	—	—	0
0	1	—	1	—	—	1
1	0	—	—	0	—	0
1	0	—	—	1	—	1
1	1	—	—	—	0	0
1	1	—	—	—	1	1

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

```
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;
```

```
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;
```

# Descrizione 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;
```