DEI
DIPARTIMENTO DI
INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Sistemi Digitali

Esercizi sui circuiti combinatori

Marta Bagatin, marta.bagatin@unipd.it

Corso di Laurea in Ingegneria dell'Informazione

Anno accademico 2022-2023

Esempio 3-17: Implementazione di una funzione a 4 variabili con multiplexer

- Implementare la seguente funzione booleana tramite un multiplexer con un segnale di selezione a 3 bit

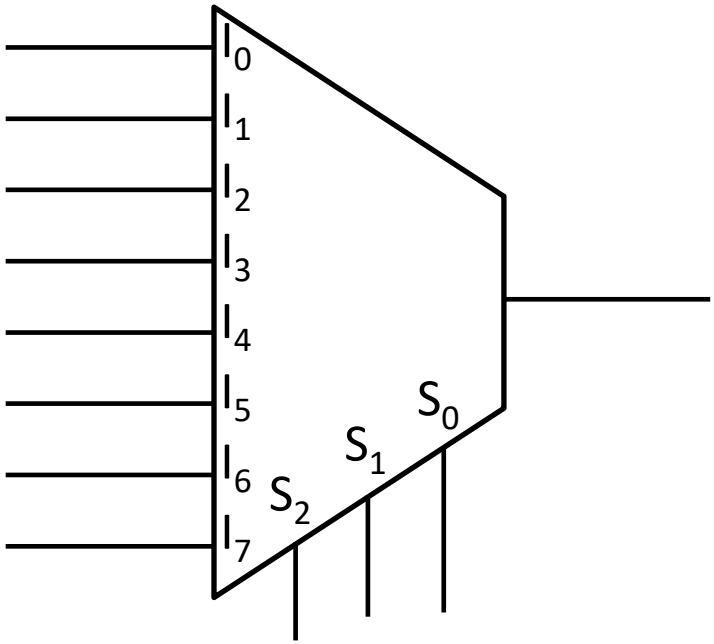
$$\mathrm{F}\,(\mathrm{A},\mathrm{B},\mathrm{C},\mathrm{D}) = \sum \mathrm{m}\,(1,3,4,11,12,13,14,15)$$

Esempio 3-17: Implementazione di una funzione a 4 variabili con multiplexer

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

a) Troviamo la tabella di verità della funzione

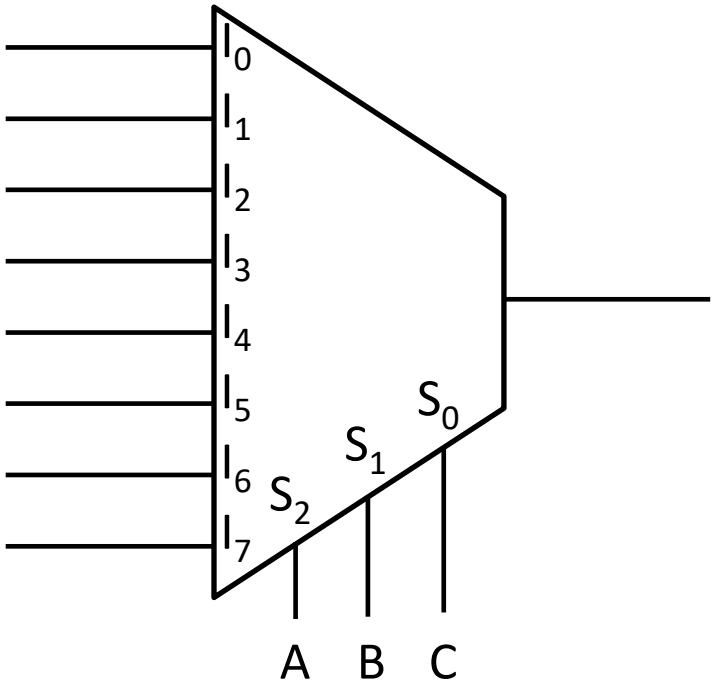
b) Usiamo un mux con 3 input di selezione (line mux 8-to-1)



Esempio 3-17: Implementazione di una funzione a 4 variabili con multiplexer

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

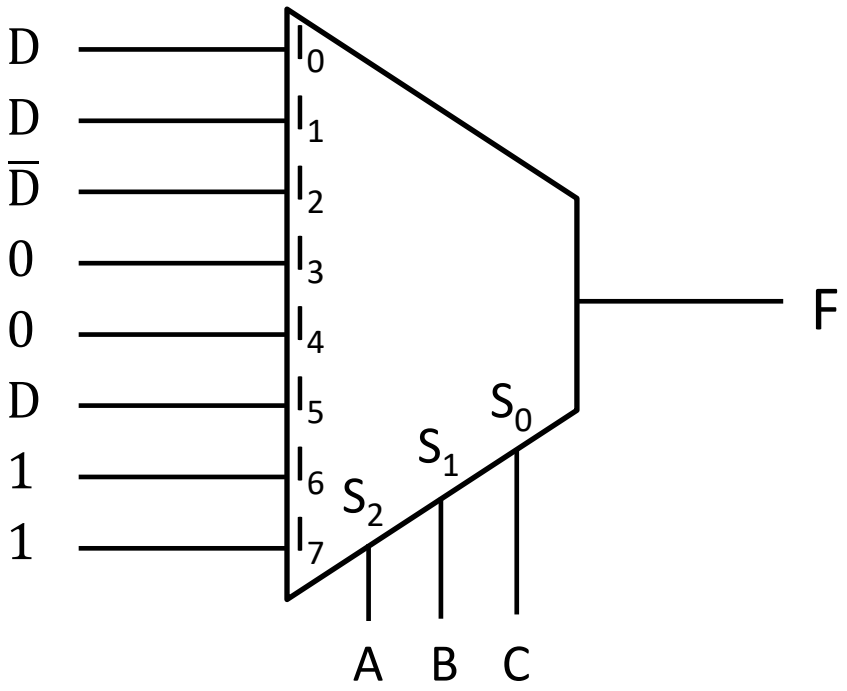
a) Troviamo la tabella di verità della funzione
b) Usiamo un mux con 3 input di selezione (line mux 8-to-1)
c) Colleghiamo 3 dei 4 ingressi della funzione F agli ingressi di selezione del mux (attenzione all'ordine!)



Esempio 3-17: Implementazione di una funzione a 4 variabili con multiplexer

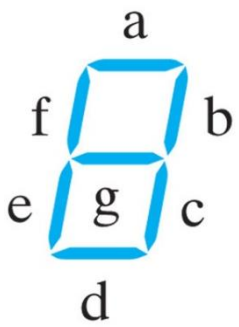
A	B	C	D	F	
0	0	0	0	0	F = D
0	0	0	1	1	
0	0	1	0	0	F = D
0	0	1	1	1	
0	1	0	0	1	$F = \overline{D}$
0	1	0	1	0	
0	1	1	0	0	F = 0
0	1	1	1	0	
1	0	0	0	0	F = 0
1	0	0	1	0	
1	0	1	0	0	F = D
1	0	1	1	1	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	

a) Troviamo la tabella di verità della funzione
b) Usiamo un mux con 3 input di selezione (line mux 8-to-1)
c) Colleghiamo 3 dei 4 ingressi della funzione F agli ingressi di selezione del mux (attenzione all'ordine!)
d) Fissiamo opportunamente il valore degli ingressi di dati del mux (value fixing) in base alla tabella di verità



Es. 3-18: BCD-sette segmenti

- **Specifiche**: realizzare un circuito combinatorio che, data una cifra BCD in ingresso, produce in uscita la configurazione a 7 segmenti per visualizzare la cifra in un display a LED. Ogni cifra del display è rappresentata da un'opportuna combinazione di 7 segmenti, costituiti da LED, la cui illuminazione deve essere controllata dal circuito



Nomi dei 7 segmenti

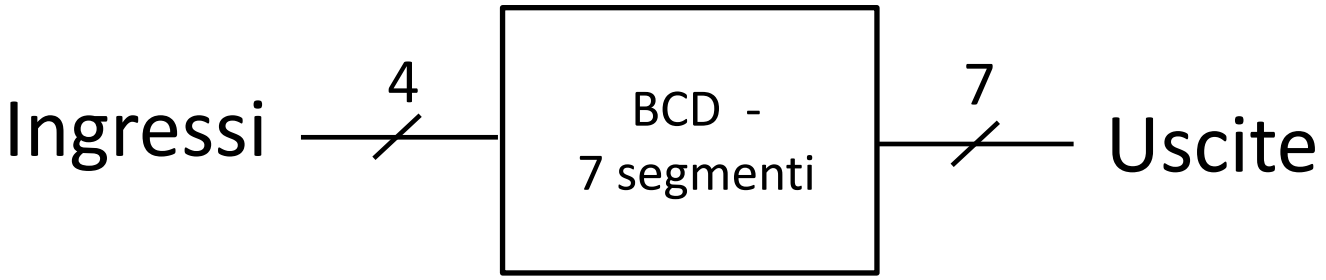
Rappresentazione delle 9 cifre BCD sul display

Es. 3-18: BCD-sette segmenti

- **Specifiche**: realizzare un circuito combinatorio che, data una cifra BCD in ingresso, produce in uscita la configurazione a 7 segmenti per visualizzare la cifra in un display a LED. Ogni cifra del display è rappresentata da un'opportuna combinazione di 7 segmenti, costituiti da LED, la cui illuminazione deve essere controllata dal circuito

  Ingressi e uscite:
  - 4 ingressi: A B C D (rappresentano le 10 cifre BCD)
  - 7 uscite: a b c d e f g (controllano i 7 segmenti)

=> le uscite del circuito selezionano il segmento corrispondente nel display, per visualizzare la cifra (uscita = '0': segmento spento, uscita = '1': segmento acceso)



Es. 3-18: BCD-sette segmenti

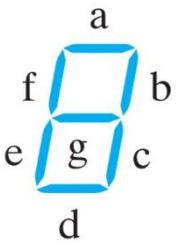
- **Tabella di verità:**

BCD Input				Seven-Segment Decoder						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
All other inputs				0	0	0	0	0	0	0

Ingresso: Codifica BCD delle cifre da 0 a 9

Uscita = ‘0’: segmento spento
Uscita = ‘1’: segmento acceso

Combinazioni 1010-1111: display spento



Es. 3-18: BCD-sette segmenti

- **Determinazione funzione booleana:**

  1^ implementazione: con porte logiche

  - Ottimizzazione: 7 mappe di Karnaugh, una per ogni uscita (a-g)
  - L'implementazione indipendente delle 7 funzioni richiede 27 porte AND e 7 porte OR

$$a = \overline{A}C + \overline{A}BD + \overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}$$

$$b = \overline{A}\,\overline{B} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD + A\overline{B}\,\overline{C}$$

$$c = \overline{A}B + \overline{A}D + \overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}$$

$$d = \overline{A}C\overline{D} + \overline{A}\,\overline{B}C + \overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C} + \overline{A}B\overline{C}D$$

$$e = \overline{A}C\overline{D} + \overline{B}\,\overline{C}\,\overline{D}$$

$$f = \overline{A}B\overline{C} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}B\overline{D} + A\overline{B}\,\overline{C}$$

$$g = \overline{A}C\overline{D} + \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C}$$

Es. 3-18: BCD-sette segmenti

- **Determinazione funzione booleana:**

  1^ implementazione: con porte logiche

  - Ottimizzazione: 7 mappe di Karnaugh, una per ogni uscita (a-g)
  - L'implementazione indipendente delle 7 funzioni richiede 27 porte AND e 7 porte OR
  - Condividendo i termini prodotto in comune tra le diverse uscite, il numero di porte AND si può ridurre a 15

$$a = \overline{A}C + \overline{A}BD + \overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}$$

$$b = \overline{A}\,\overline{B} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD + A\overline{B}\,\overline{C}$$

$$c = \overline{A}B + \overline{A}D + \overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}$$

$$d = \overline{A}C\overline{D} + \overline{A}\,\overline{B}C + \overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C} + \overline{A}B\overline{C}D$$

$$e = \overline{A}C\overline{D} + \overline{B}\,\overline{C}\,\overline{D}$$

$$f = \overline{A}B\overline{C} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}B\overline{D} + A\overline{B}\,\overline{C}$$

$$g = \overline{A}C\overline{D} + \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C}$$

Es. 3-18: BCD-sette segmenti

2^ implementazione: con decoder

- Usiamo un decoder 4-to-16 per realizzare tutti i mintermini corrispondenti ai 4 ingressi del circuito
- Colleghiamo i mintermini a 7 porte OR (una per ogni funzione in uscita)

BCD Input				Seven-Segment Decoder						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
All other inputs				0	0	0	0	0	0	0

$$a(A, B, C, D) = \Sigma m(0, 2, 3, 5, 6, 7, 8, 9)$$

$$b(A, B, C, D) = \Sigma m(0, 1, 2, 3, 4, 7, 8, 9)$$

$$c(A, B, C, D) = \Sigma m(0, 1, 3, 4, 5, 6, 7, 8, 9)$$

$$d(A, B, C, D) = \Sigma m(0, 2, 3, 5, 6, 8, 9)$$

$$e(A, B, C, D) = \Sigma m(0, 2, 6, 8)$$

$$f(A, B, C, D) = \Sigma m(0, 4, 5, 6, 8, 9)$$

$$g(A, B, C, D) = \Sigma m(2, 3, 4, 5, 6, 8, 9)$$

Es. 3-18: BCD-sette segmenti

3^ implementazione: con multiplexer

- Sette line mux 8-to-1 (uno per ogni uscita)
  - ingressi di selezione S_2 S_1 S_0 connessi ad A, B, C (attenzione all'ordine)
  - ingressi di dati I_0, I_1, I_2 …, I_7 connessi opportunamente a D, $\overline{D}$, 0, 1

BCD Input				Seven-Segment Decoder						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
All other inputs				0	0	0	0	0	0	0

Select Inputs	Multiplexer Data Inputs for Each Output Function						
$S_2S_1S_0$	a	b	c	d	e	f	g
000	$\overline{D}$	1	1	$\overline{D}$	$\overline{D}$	$\overline{D}$	0
001	1	1	D	1	$\overline{D}$	0	1
010	D	$\overline{D}$	1	D	0	1	1
011	1	D	1	$\overline{D}$	$\overline{D}$	$\overline{D}$	$\overline{D}$
100	1	1	1	1	$\overline{D}$	1	1
101	0	0	0	0	0	0	0
110	0	0	0	0	0	0	0
111	0	0	0	0	0	0	0

Es. 3-18: BCD-sette segmenti

3^ implementazione: con multiplexer

- Sette line mux 8-to-1 (uno per ogni uscita)
  - ingressi di selezione S_2 S_1 S_0 connessi ad A, B, C (attenzione all'ordine)
  - ingressi di dati I_0, I_1, I_2 …, I_7 connessi opportunamente a D, $\overline{D}$, 0, 1

BCD Input				Seven-Segment Decoder						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
All other inputs				0	0	0	0	0	0	0

Select Inputs	Multiplexer Data Inputs for Each Output Function						
$S_2S_1S_0$	a	b	c	d	e	f	g
000	$\overline{D}$	1	1	$\overline{D}$	$\overline{D}$	$\overline{D}$	0
001	1	1	D	1	$\overline{D}$	0	1
010	D	$\overline{D}$	1	D	0	1	1
011	1	D	1	$\overline{D}$	$\overline{D}$	$\overline{D}$	$\overline{D}$
100	1	1	1	1	$\overline{D}$	1	1
101	0	0	0	0	0	0	0
110	0	0	0	0	0	0	0
111	0	0	0	0	0	0	0

Esercizio 3-55: operazioni aritmetiche

- **Eseguire le operazioni aritmetiche tra i seguenti numeri con segno rappresentati in complemento a 2**. Verificare i risultati delle operazioni e indicare se c'è overflow.

  a) 110001 + 011101
  b) 0110111 + 0101111
  c) 0000 0111 - 1111 0100
  d) 0110111 - 0101111

Esercizio 3-55

a) 110001 + 011101

Riporto: 1 10001

110001 +
011101

001110

Verifica:
-15 + 29 = + 14

NOTE:

- Non c'è overflow dato che i bit di riporto nell'ultima e penultima posizione sono uguali tra loro
- C'è riporto in uscita (che non va considerato)

Esercizio 3-55

b) 0110111 + 0101111

Riporto: 0111111

```
  0110111 +
  0101111
-----------
  1100110
```

Verifica:
+55 + 47 = + 102

NOTE:

- Non c'è riporto in uscita
- **C'è overflow** dato che il riporto in uscita e il riporto nella posizione del segno sono diversi

Esercizio 3-55

c) 0000 0111 – 1111 0100

Per sottrarre numeri con segno, si fa il complemento a 2 del sottraendo e lo si somma al minuendo

Riporto: 0001 100

0000 0111 +

0000 1100

0001 0011

Verifica:

+7 – (- 12) = + 19

NOTE:

- Non c'è riporto in uscita
- Non c'è overflow

Esercizio 3-55

d) 0110111 – 0101111

Riporto: 1110111

```
  0110111 +
  1010001
-------------
  0001000
```

Verifica:
+55 – (+ 47) = + 8

NOTE:

- **C'è riporto in uscita e viene scartato**
- Non c'è overflow

Circuito Prime

- Realizzare il codice VHDL per descrivere il circuito così definito
  - Ingresso (4 bit): input
  - Uscita (1 bit): isPrime, ‘1’ se input è primo, ‘0’ altrimenti

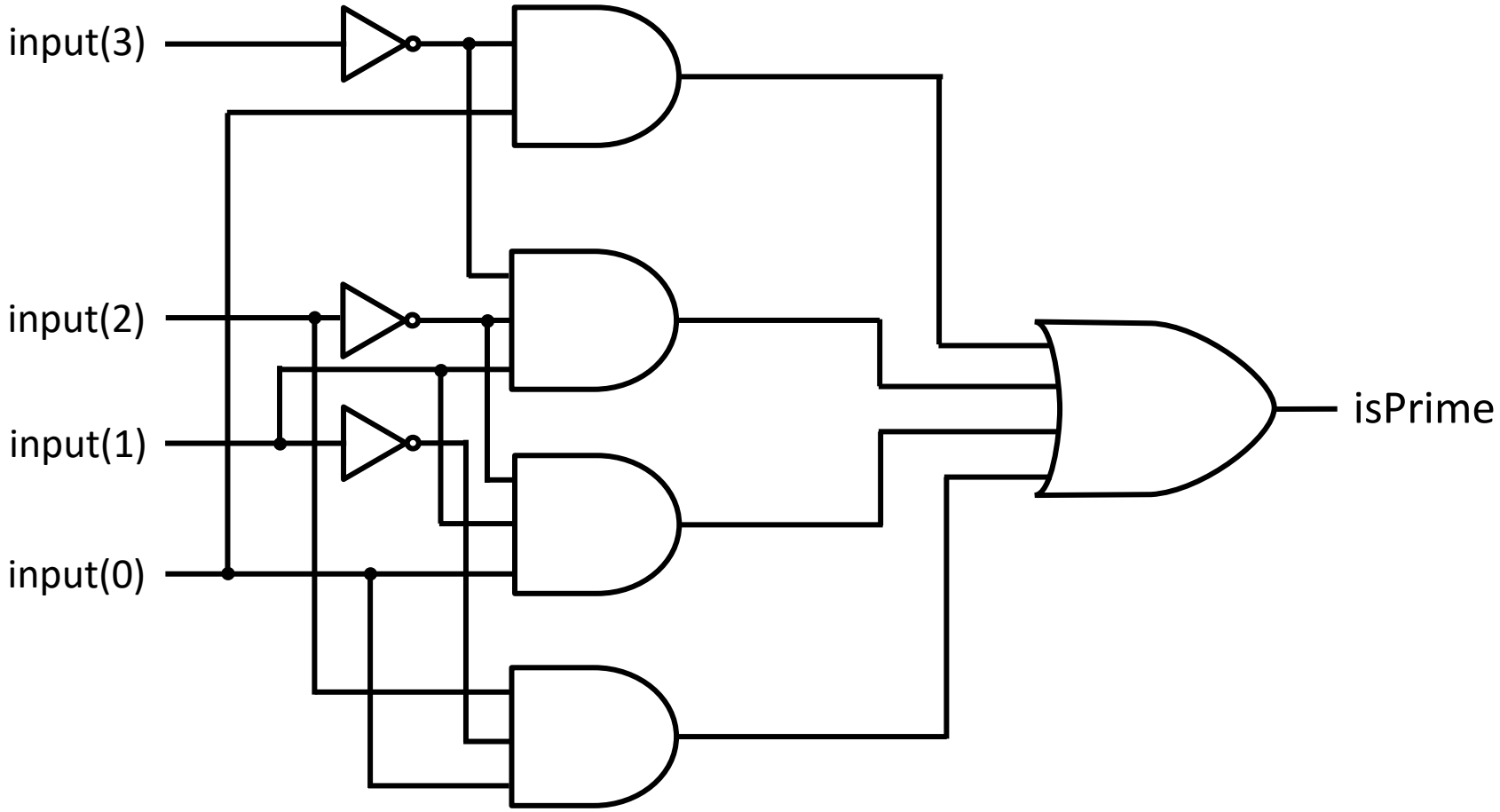
  nei seguenti modi:

  1) Descrizione structural
  2) Descrizione dataflow
  3) Descrizione behavioral

- Realizzare il testbench per verificare il funzionamento del circuito e simularlo con EDA Playground

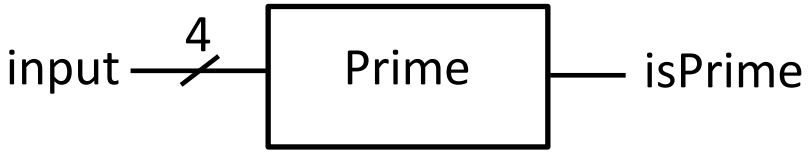
Circuito Prime

Diagramma logico del circuito (ottimizzato con mappa di Karnaugh):



Circuito Prime

1) Descrizione **structural-dataflow (1/2)**




```
library IEEE;
use IEEE.std_logic_1164.all;

entity Prime is
        port (input: in std_logic_vector (3 downto 0);
              isPrime: out std_logic);
end Prime;

architecture implStruct of Prime is
        signal a1, a2, a3, a4, n1, n2, n3: std_logic;
        begin
        AND1: entity work.andGate(andImpl) port map(a1, n3, input(0));
        AND2: entity work.andGate(andImpl) port map(a2, input(2), n1, input(0));
        AND3: entity work.andGate(andImpl) port map(a3, n2, input(1), input(0));
        AND4: entity work.andGate(andImpl) port map(a4, n3, n2, input(1));
        n3 <= NOT input(3);
        n2 <= NOT input(2);
        n1 <= NOT input(1);
        isPrime <= a1 OR a2 OR a3 OR a4;
end implStruct;
```

Component AND con 3 ingressi, impostati di default a '1'

Circuito Prime

1) Descrizione **structural-dataflow (2/2)**

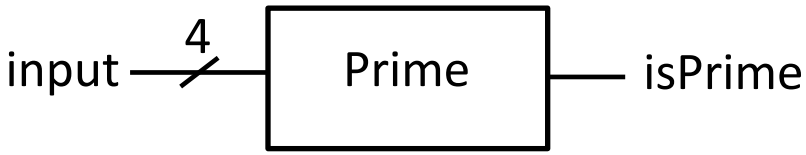
```
library IEEE;
use IEEE.std_logic_1164.all;

entity andGate is
       port (y: out std_logic;
             a, b, c: in std_logic:= '1');
end andGate;

architecture andImpl of andGate is
       begin
    y <= a and b and c;
end andImpl;
```

Circuito Prime

2) Descrizione **dataflow**



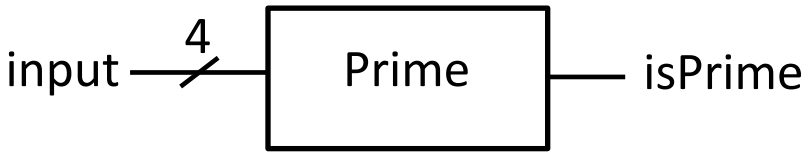

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Prime is
   port (input: in std_logic_vector (3 downto 0);
         isPrime: out std_logic);
end Prime;

architecture implLogic of Prime is
   begin
   isPrime <= (input(0) AND NOT(input(3))) OR
              (input(1) AND NOT (input(2)) AND NOT(input(3))) OR
              (input(0) AND NOT (input(1)) AND input(2)) OR
              (input(0) AND input(1) AND NOT (input(2)));
end implLogic;
```

Circuito Prime

3.a) Descrizione **behavioral** con costrutto **case**




```
library IEEE;
use IEEE.std_logic_1164.all;

entity Prime is
       port (input: in std_logic_vector (3 downto 0);
             isPrime: out std_logic);
end Prime;

architecture implCase of Prime is
  begin
    process(input) begin
    case input is
      when x"1" | x"2" | x"3" | x"5" | x"7" | x"b" | x"d" =>
            isPrime <= '1';
      when others => isPrime <= '0';
    end case;
    end process;
end implCase;
```

Case: Logica parallela

Sintassi per il costrutto case (**analogo sequenziale di "with-select"**):

```
case expression is
                when choice1 => {statements}
                when choice2 => {statements}
                when others => {statements}
end case;
```

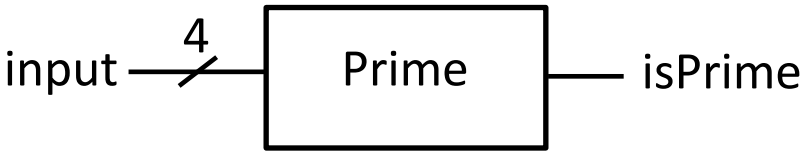
- Deve apparire dentro ad un **process**
- Permette di specificare il valore di uscita di una funzione logica per tutte le combinazioni degli ingressi
- Bisogna **coprire tutte le scelte possibili** (come "with select") e le varie scelte non si possono sovrapporre. Si può usare l'istruzione "null" per non eseguire alcuna azione in condizioni specifiche
- Nella sensitivity list di un process per generare un circuito combinatorio devono apparire tutti gli ingressi

Circuito Prime

3.b) Descrizione **behavioral** con costrutto **if**

```
entity Prime is
       port (input: in std_logic_vector (3 downto 0);
         isPrime: out std_logic);
end Prime;

architecture implIf of Prime is
       begin
    process (input) begin
          if input = 4d"1" then isPrime <= '1';
          elsif input = 4d"2" then isPrime <= '1';
          elsif input = 4d"3" then isPrime <= '1';
          elsif input = 4d"5" then isPrime <= '1';
          elsif input = 4d"7" then isPrime <= '1';
          elsif input = 4d"11" then isPrime <= '1';
          elsif input = 4d"13" then isPrime <= '1';
          else isPrime <= '0';
          end if;
    end process;
end implIf;
```



Decimale a 4 bit

If: Logica prioritaria

Sintassi per il costrutto if (**analogo sequenziale di "when-else"**):

```
if condition1 then
    {sequential_statement1}
elsif condition2 then
    {sequential_statement2}
else
    {sequential_statement3}
end if;
```

- If deve apparire dentro ad un 'process'
- Valuta ogni condizione in ordine, cioè genera una struttura di priorità (corrisponde al concurrent statement "when-else")
- Attenzione: dobbiamo specificare tutti i casi se vogliamo realizzare logica combinatoria usando if-else statement
- Nella sensitivity list di un process per generare un circuito combinatorio devono apparire tutti gli ingressi

Disclaimer

Figures from *Logic and Computer Design Fundamentals*, Fifth Edition, GE Mano | Kime | Martin