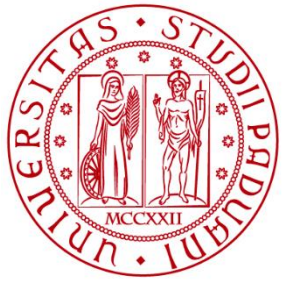
DEI
DIPARTIMENTO DI
INGEGNERIA DELL'INFORMAZIONE



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

# Sistemi Digitali

## Sommatori e sottrattori binari

Marta Bagatin, marta.bagatin@unipd.it

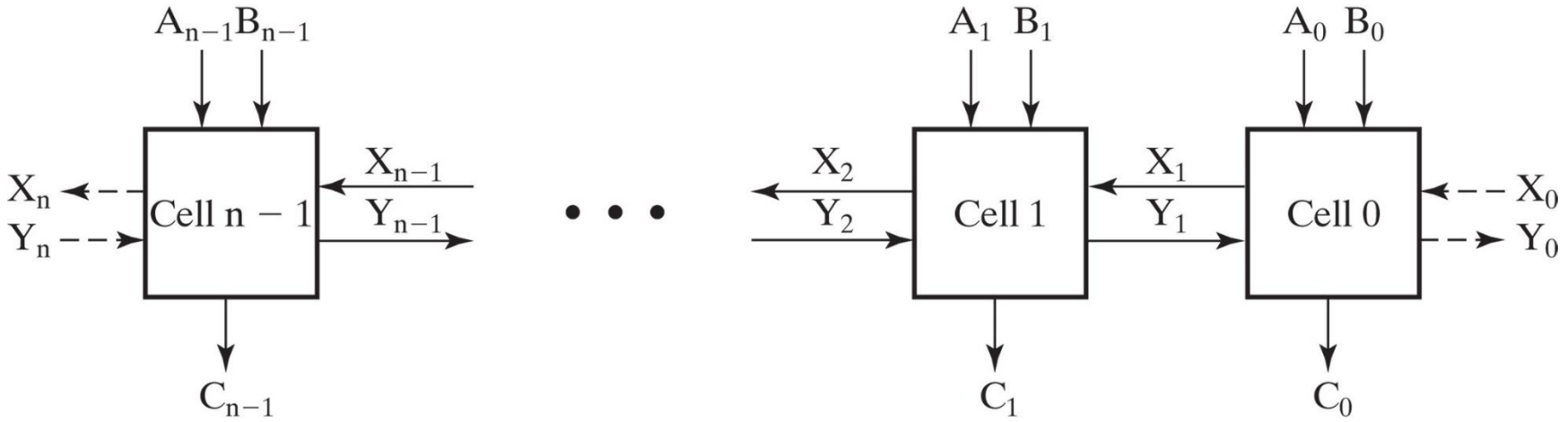
Corso di Laurea in Ingegneria dell'Informazione

Anno accademico 2022-2023

# Scopo della lezione

- Studiare i circuiti per somma e sottrazione di numeri binari a n bit
  - Half adder
  - Full adder
  - Ripple carry adder
  - Sottrazione di numeri senza segno
  - Rappresentazione di numeri con segno
  - Somma e sottrazione di numeri con segno
  - Gestione dell'overflow
  - Rappresentazioni VHDL del sommatore binario

# Circuiti combinatori iterativi




- Sono un esempio di **struttura gerarchica** e sono utili per funzioni che operano su bit multipli
- La stessa sottofunzione viene applicata ad ogni singolo bit. Il circuito viene realizzato combinando più blocchi (identici o simili), che si **passano iterativamente i valori che servono nel sottoblocco successivo**. Tali valori intermedi sono usati solo internamente e non sono presenti in uscita al circuito

# Circuiti aritmetici binari

- Un circuito aritmetico binario è un **circuito combinatorio che esegue operazioni aritmetiche** (somma, sottrazione, …) sui numeri binari
- Vedremo che è molto conveniente usare **design gerarchici iterativi** per sviluppare i circuiti aritmetici

# Sommatore binario

# Somma di due cifre binarie

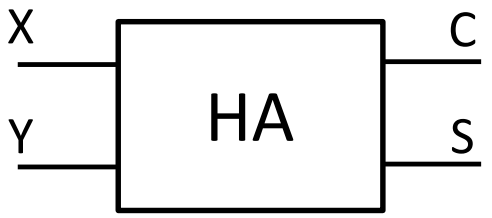
| X | Y | X + Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 10 |

- Se gli addendi sono entrambi ‘1’, servono 2 bit per il risultato!

# Half Adder

- E’ un circuito combinatorio che esegue la **somma tra 2 bit**

| Inputs | | Outputs | |
|---|---|---|---|
| X | Y | C | S |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |




X, Y: addendi

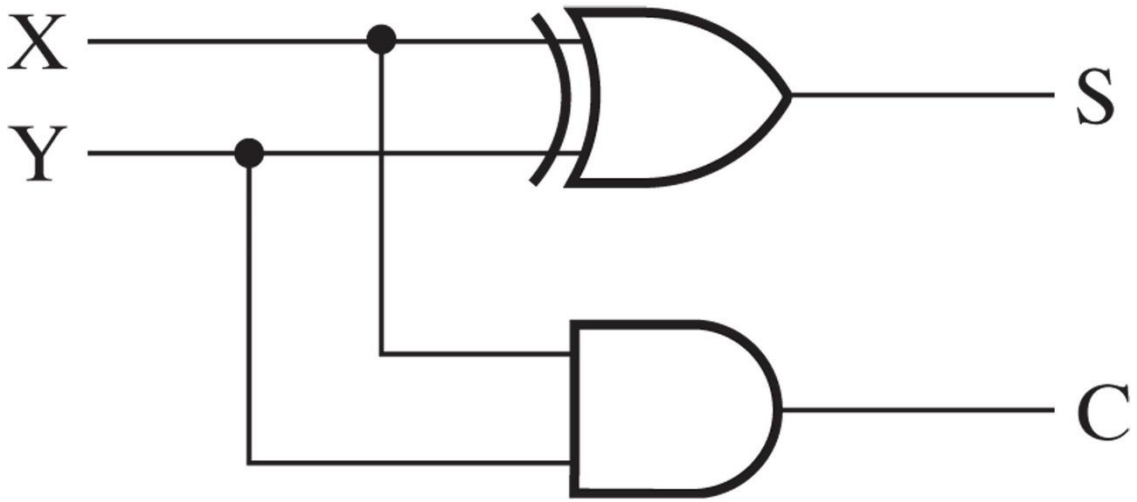
C (carry): riporto

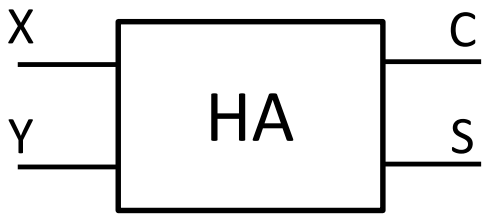
S: LSB della somma

# Half Adder

$S = X \oplus Y$

$C = X \cdot Y$




X, Y: addendi

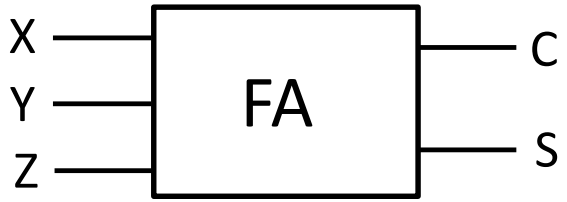
C (carry): riporto

S: LSB della somma

# Full Adder

- E' un circuito combinatorio che esegue la **somma di 3 bit**: 2 bit da sommare e 1 bit di riporto precedente

| Inputs | | | Outputs | |
|---|---|---|---|---|
| X | Y | Z | C | S |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

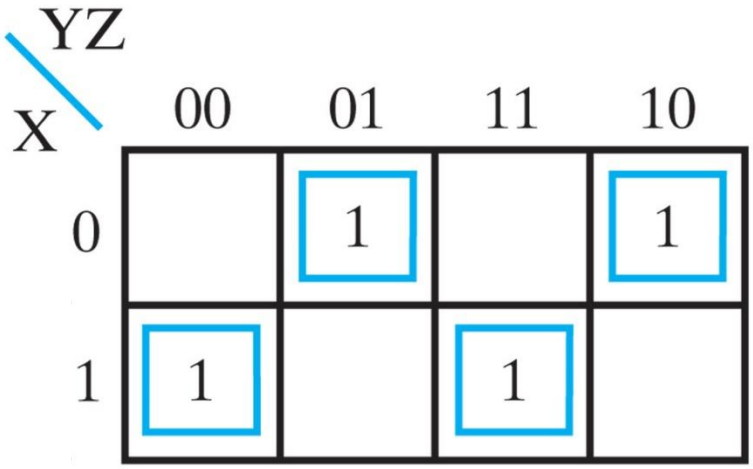


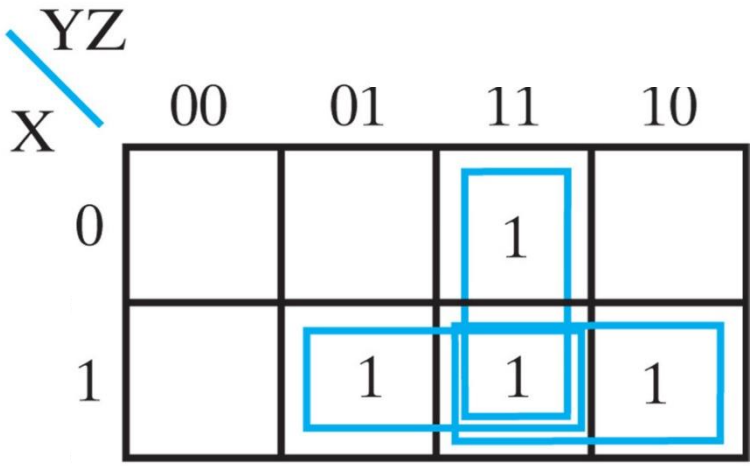

X, Y : addendi
Z : riporto precedente
S : LSB della somma
C (carry): riporto

$$S(X,Y,Z) = \sum m(1,2,4,7)$$

$$C(X,Y,Z) = \sum m(3,5,6,7)$$

# Full Adder




$$S = \overline{X}\,\overline{Y}\,Z + \overline{X}\,Y\,\overline{Z} + X\,\overline{Y}\,\overline{Z} + X\,Y\,Z$$
$$= X \oplus Y \oplus Z$$

XOR di 3 variabili: S è espressa dalla funzione disparità (='1' se ha in ingresso un numero dispari di '1')

$$C = XY + XZ + YZ$$
$$= XY + Z(X + Y)$$
$$= XY + Z(X \oplus Y + XY)$$
$$= XY + ZXY + Z\,(X \oplus Y)$$
$$= XY + Z\,(X \oplus Y)$$

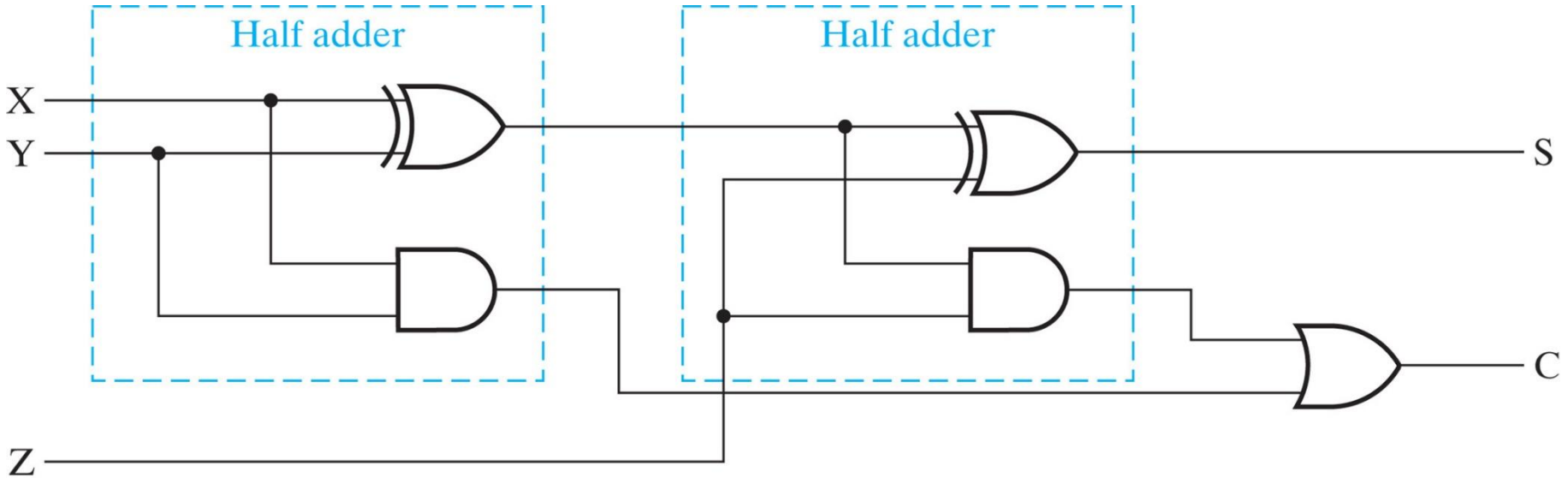
Abbiamo espresso C in modo che includa una porta XOR

# Full Adder

- Possiamo quindi realizzare il full adder con **due half adder e una porta OR a 2 ingressi**

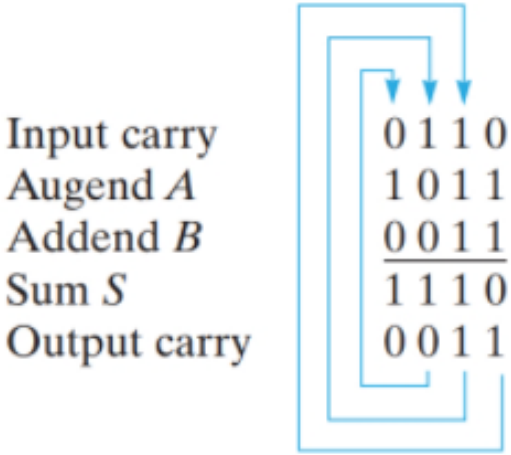
$$S = (X \oplus Y) \oplus Z$$

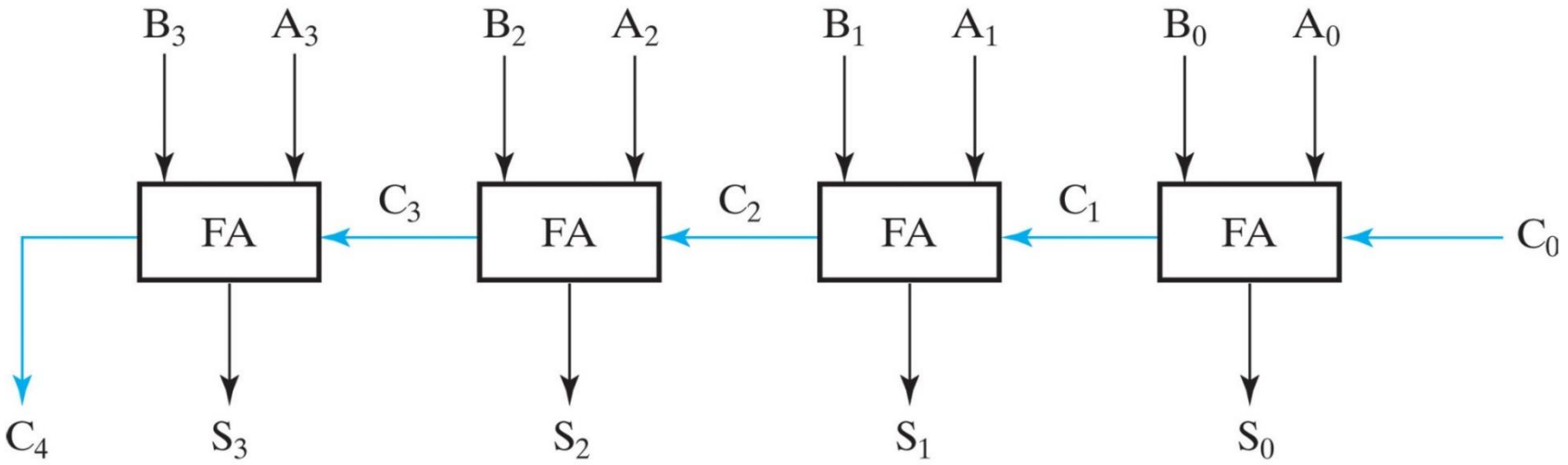
$$C = XY + Z\,(X \oplus Y)$$



# Ripple Carry Adder

- La **somma di due numeri binari a n bit** viene eseguita bit a bit, con **n full adder connessi (in catena) uno dopo l'altro**
  - Il riporto in uscita da ciascun full adder diventa il riporto in ingresso del FA successivo, partendo dal LSB e andando verso il MSB (ripple = propagare)




  - Esempio di circuito iterativo: blocco complesso è stato costruito riutilizzando blocchi semplici (tabella di verità con $2^9 = 512$ righe!)

# Sottrattore binario

# Numeri senza segno

- Finora abbiamo considerato numeri senza segno (**unsigned**): la rappresentazione binaria non include alcuna informazione sul segno del numero

# Sottrazione binaria con M > N

- Abbiamo visto la sottrazione di due numeri binari a n bit: quando il minuendo (M) è più grande del sottraendo (N), la sottrazione restituisce un valore corretto e non c'è prestito nella posizione più significativa
  - **Esempio con M ≥ N**

| | | | |
|---|---|---|---|
| Prestito | 0011 | | **-> Prestito 0 in uscita** |
| Minuendo | 11110 | $30_{10}$ | |
| Sottraendo | 10011 | $19_{10}$ | |
| Differenza | 01011 | $11_{10}$ | **-> Differenza è positiva e corretta!** |

# Sottrazione binaria con M < N

- Quando il minuendo (M) è più piccolo del sottraendo (N), il procedimento restituisce un valore errato

  – **Esempio con M < N**

| | | |
|---|---|---|
| Prestito | 11100 | |
| Minuendo | 10011 | $19_{10}$ |
| Sottraendo | 11110 | $30_{10}$ |
| Differenza (scorretta) | 10101 | $21_{10}$ |
| Differenza (valore assoluto corretto) | - 01011 | $-11_{10}$ |

**-> Prestito 1 in uscita**

Abbiamo erroneamente aggiunto $2^n$ (prestito nella posizione più significativa)

**-> Valore assoluto della differenza non corretto**: ottengo M-N+$2^n$ invece di M-N

-> Valore assoluto corretto è M-N

# Sottrazione binaria

- Due possibili alternative per realizzare la sottrazione di numeri senza segno in modo corretto sia con $M \geq N$ che con $M < N$
  a) Se $M < N$, scambiare M con N e invertire il segno del risultato: costoso e non conveniente in termini di realizzazione circuitale
  b) Se $M < N$, non scambiare M con N, sottrarre $2^n$ al risultato 'non corretto' e aggiungere un meno davanti

# Sottrazione binaria

- Metodo b) per sottrarre un numero N (n bit) da M (n bit)

  1) Eseguire la sottrazione M-N

  2) Se il prestito in uscita è 0, il risultato è positivo e corretto ($M \geq N$)

  3) Se il prestito in uscita è 1 ($M < N$), il risultato è negativo e il suo valore assoluto è $2^n$ meno il risultato ottenuto: come vedremo tra poco, questo è detto complemento a 2 del risultato ottenuto ($M - N + 2^n$)

  **Esempio**: realizzare la sottrazione 01100100 – 10010110 (n = 8)

# Sottrazione binaria

- Metodo b) per sottrarre un numero N (n bit) da M (n bit)

  1) Eseguire la sottrazione M-N

  2) Se il prestito in uscita è 0, il risultato è positivo e corretto ($M \geq N$)

  3) Se il prestito in uscita è 1 ($M < N$), il risultato è negativo e il suo valore assoluto è $2^n$ meno il risultato ottenuto: come vedremo tra poco, questo è detto complemento a 2 del risultato ottenuto ($M - N + 2^n$)

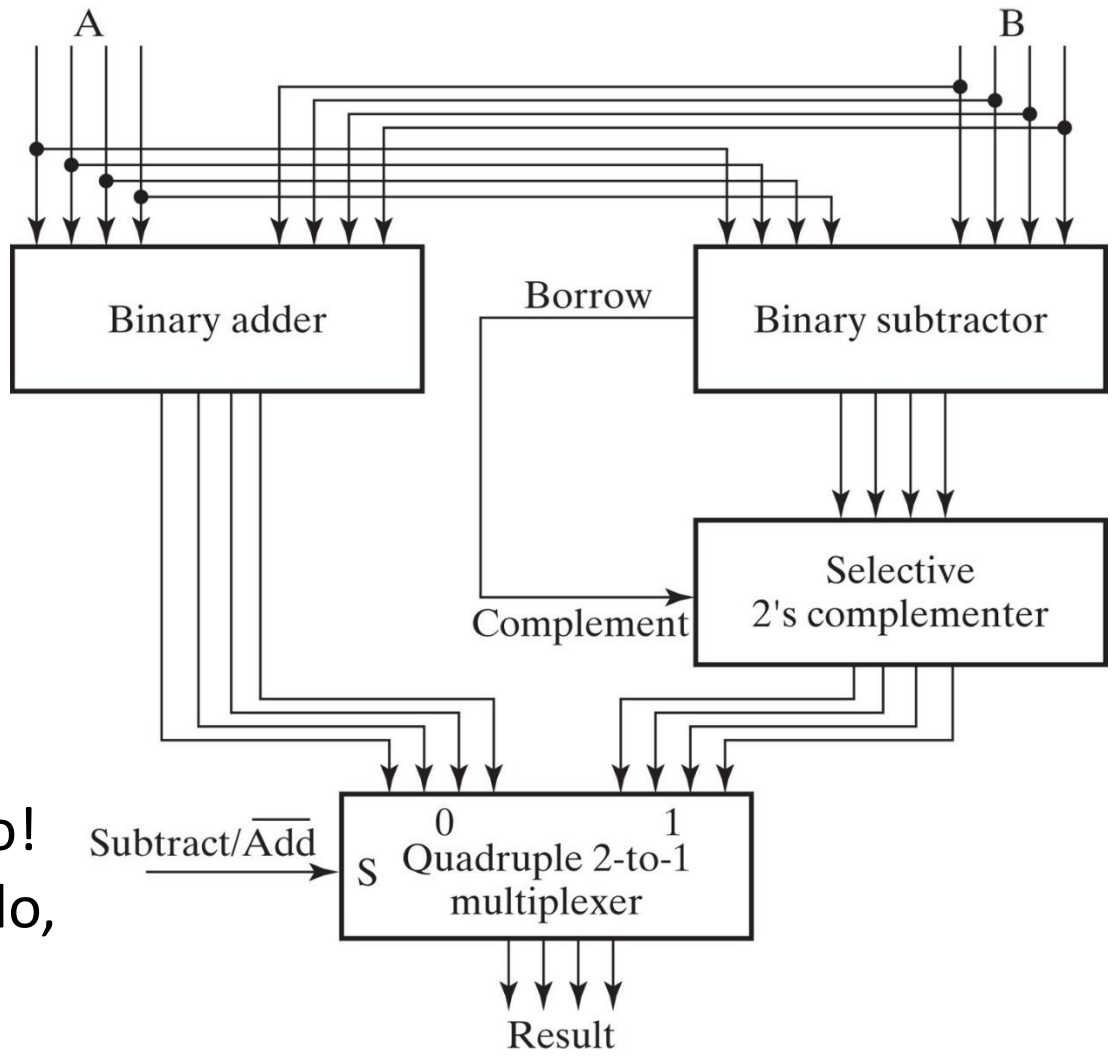
**Esempio**: realizzare la sottrazione 01100100 – 10010110 (n = 8)

| | | | |
|---|---|---|---|
| Prestito | 10011110 | | **-> Prestito 1 in uscita (M < N)** |
| Minuendo | 01100100 | $100_{10}$ | |
| Sottraendo | 10010110 | $150_{10}$ | |
| Differenza (scorretta) | 11001110 | $206_{10}$ | -> per ottenere il valore assoluto della differenza corretta dobbiamo fare $2^n$ meno il risultato ottenuto: 256–206=50 |
| Differenza (corretta) | -0110010 | $-50_{10}$ | |

# Addizionatore-sottrattore binario

Diagramma del circuito che realizza la sottrazione con il metodo b) richiede

- un sottrattore
- un addizionatore
- un circuito che esegue il complemento a 2 (selettivo: solo se il prestito in uscita vale 1)
- un mux che seleziona l'operazione da fare




Questo circuito è complicato!
→ Cerchiamo di semplificarlo, condividendo la logica tra addizionatore e sottrattore….

# Complementi

- Dato un **numero binario N** a **n bit**, definiamo
  - **Complemento a 1**: $(2^n - 1) - N$ → negazione logica di N (bit a bit)
    (fa il complemento di ogni bit)
  - **Complemento a 2**: $(2^n - 1) - N + 1$ → (complemento a 1) + 1
    **$= 2^n - N$**

| N | Complemento a 1 | Complemento a 2 |
|---|---|---|
| 1011001 | 0100110 | 0100111 |
| 0001111 | 1110000 | 1110001 |
| 101100 | 010011 | 010100 |

NB: il complemento del complemento di N è pari a N

# Addizionatore-sottrattore binario di numeri con segno

# Rappresentazione dei numeri con segno

- Vedremo due tipi di rappresentazioni dei numeri con segno
  - Rappresentazione con segno e modulo
  - Rappresentazione in complemento a 2

# Rappresentazione segno e modulo

- Al numero binario si antepone il bit di segno: **‘0’ per i numeri positivi (o nulli), ‘1’ per i numeri negativi**
  - Una stringa di n bit può rappresentare numeri diversi, a seconda che sia considerata come un numero senza segno o con segno (rappresentazione segno e modulo)

| N binario | Senza segno | Segno e modulo |
|---|---|---|
| 01001 | 9 | +9 |
| 11001 | 25 | -9 |

  - Nelle operazioni aritmetiche tra numeri così rappresentati, il bit di segno e i bit del valore assoluto sono processati separatamente, i bit del valore assoluto sono trattati come unsigned, quindi la sottrazione richiede una correzione del segno

# Rappresentazione complemento a 2

- Per evitare di dover correggere il segno in caso di sottrazione, viene adottata la rappresentazione in complemento a 2
  - Numeri **positivi**: sono indicati **dal bit di segno '0' (MSB) seguito dalla rappresentazione binaria** del numero
  - Numeri **negativi:** sono indicati con la **rappresentazione in complemento a 2 del corrispondente numero positivo** ($2^n - N$)

  $\Rightarrow$ In posizione MSB tutti i numeri positivi hanno '0', tutti i numeri negativi hanno '1' !

Esempio: rappresentazione binaria a 8 bit (n = 8)

| N decimale | Segno e modulo | Complemento a 2 |
|---|---|---|
| +9 | 00001001 | 00001001 |
| -9 | 10001001 | 11110111 |

# Numeri con segno

- I numeri positivi sono uguali nelle due rappresentazioni
- I numeri positivi hanno sempre MSB ’0’, i negativi sempre MSB ’1’
- Segno e modulo: 7 numeri positivi, 7 negativi, 2 zeri (0 positivo e 0 negativo)
- Complemento a 2: 7 numeri positivi, 8 negativi, uno zero (positivo)
- Segno e modulo è usata nell’ aritmetica ordinaria, complemento a 2 è usata nei computer, consentendo implementazioni più semplici di sommatore/sottrattore

**Signed Binary Numbers**

| Decimal | Signed 2s Complement | Signed Magnitude |
|---|---|---|
| + 7 | 0111 | 0111 |
| + 6 | 0110 | 0110 |
| + 5 | 0101 | 0101 |
| + 4 | 0100 | 0100 |
| + 3 | 0011 | 0011 |
| + 2 | 0010 | 0010 |
| + 1 | 0001 | 0001 |
| + 0 | 0000 | 0000 |
| − 0 | — | 1000 |
| − 1 | 1111 | 1001 |
| − 2 | 1110 | 1010 |
| − 3 | 1101 | 1011 |
| − 4 | 1100 | 1100 |
| − 5 | 1011 | 1101 |
| − 6 | 1010 | 1110 |
| − 7 | 1001 | 1111 |
| − 8 | 1000 | — |

# Somma di numeri con segno

- La somma di numeri binari con segno in rappresentazione complemento a 2 si ottiene **sommando i numeri (incluso il bit di segno)** e **scartando l'eventuale riporto in uscita del bit di segno**
- Si ottiene il risultato rappresentato in complemento a 2

- Esempi:

| | | | |
|---|---|---|---|
| 00000110 | $+6_{10}$ | 11111010 | $-6_{10}$ |
| 00001101 | $+13_{10}$ | 00001101 | $+13_{10}$ |
| 00010011 | $+19_{10}$ | 100000111 | $+7_{10}$ |
| 00000110 | $+6_{10}$ | 11111010 | $-6_{10}$ |
| 11110011 | $-13_{10}$ | 11110011 | $-13_{10}$ |
| 11111001 | $-7_{10}$ | 111101101 | $-19_{10}$ |

# Sottrazione di numeri con segno

- La sottrazione di numeri binari con segno in rappresentazione complemento a 2 si ottiene **sommando (incluso il bit di segno) al minuendo il complemento a 2 del sottraendo, e scartando l'eventuale riporto in uscita del bit di segno**
- Equivalente alla somma algebrica: $(\pm A) - (+B) = (\pm A) + (-B)$
  $(\pm A) - (-B) = (\pm A) + (+B)$
- Consente ad un calcolatore di avere un hardware comune per addizione e sottrazione
- Si ottiene il risultato rappresentato in complemento a 2
- Es: -6-(-13) = -6 +13

| | |
|---|---|
| 11111010 | $-6_{10}$ |
| 00001101 | $-(-13)_{10}$ |
| 100000111 | $+7_{10}$ |

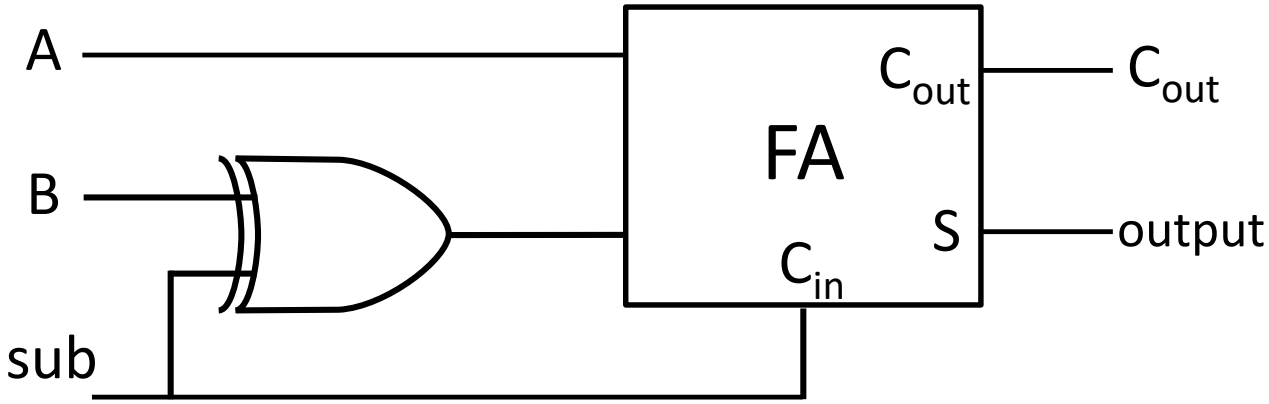
+6-(-13) = +6 +13

| | |
|---|---|
| 00000110 | $+6_{10}$ |
| 00001101 | $-(-13)_{10}$ |
| 00010011 | $+19_{10}$ |

# Addizionatore-sottrattore binario

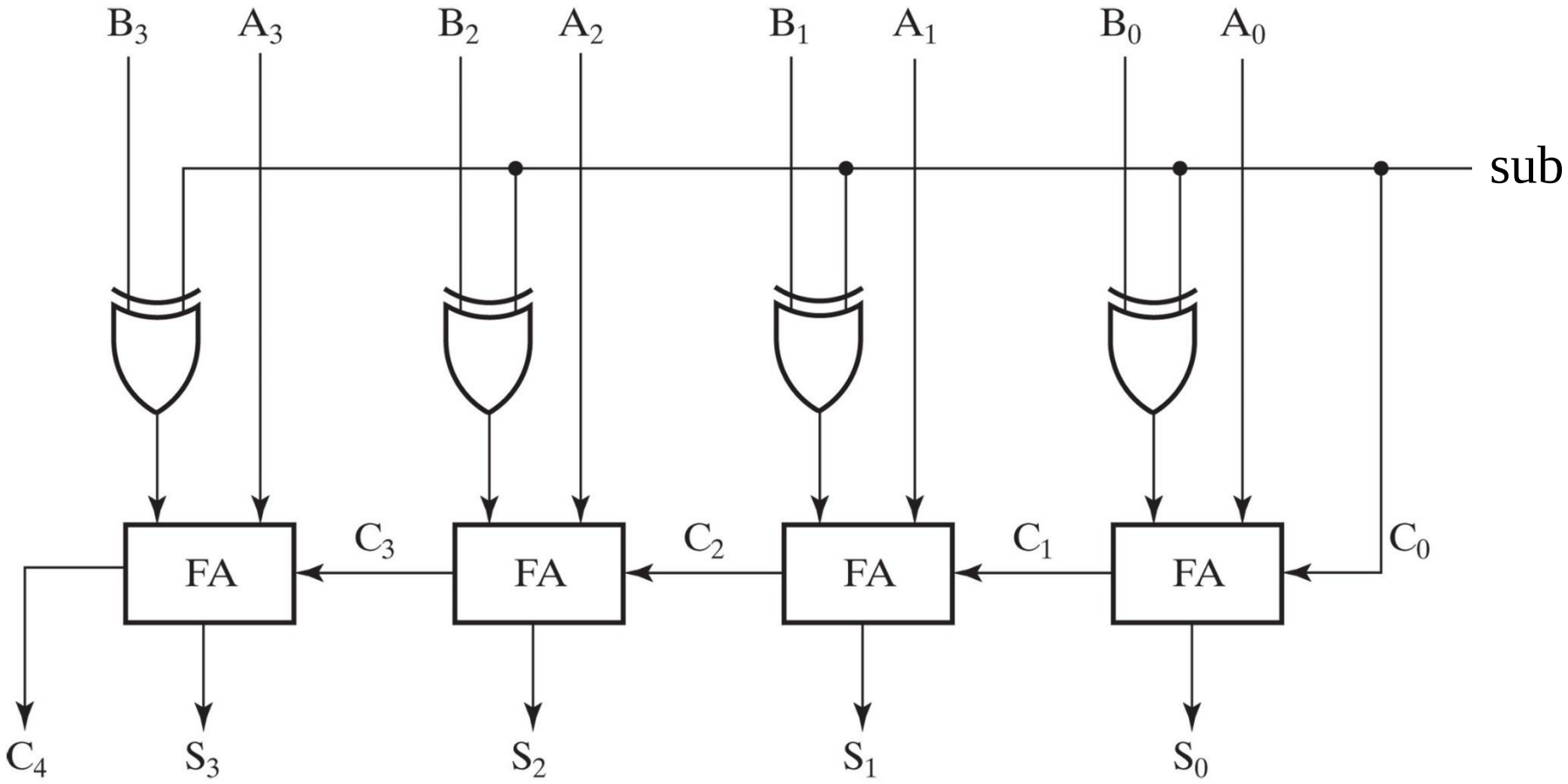
- Grazie al complemento a 2, combiniamo addizione e sottrazione in un sommatore (Full Adder, FA) comune e introduciamo una porta XOR, il cui scopo è fornire **selettivamente** il complemento a uno del secondo addendo (a cui va poi sommato 1), nel caso si debba fare una sottrazione

Inputs: A, B, sub
Outputs: output, $C_{out}$




- **se sub = 0, esegue la somma**: la porta XOR trasmette B e il FA esegue A + B (con riporto 0 in ingresso)
- **se sub = 1 esegue la sottrazione**: la porta XOR trasmette il complemento di B, e il FA esegue A + NOT(B) + 1 (somma di A con il complemento a 2 di B)

# Addizionatore-sottrattore binario: n=4




Input: $A_i$, $B_i$, sub (se sub = 0: esegue la somma, se sub = 1 esegue la sottrazione)
Output: $S_i$, $C_4$

# Overflow

# Il problema dell'overflow

- L'overflow (=straripamento) accade quando **non ci sono abbastanza bit nel risultato per rappresentare accuratamente il valore** della somma o della sottrazione
  - Es: sommo tra loro due numeri da 4 bit e ottengo un risultato di 5 bit
- L'overflow è un problema in un calcolatore, dove il numero di bit allocato per la rappresentazione dei numeri è fissato a priori
- Per segnalare che il risultato non è corretto con il numero di bit a disposizione, il **calcolatore deve essere in grado di rilevare che è avvenuto un overflow**

# Rilevazione dell'overflow

- A seconda che si considerino numeri senza segno o con segno, la rilevazione dell'overflow è diversa
- **Numeri senza segno**
  - nell'addizione c'è overflow se c'è riporto in uscita nella posizione più significativa
  - nella sottrazione non può esserci overflow (il risultato è sempre minore o uguale al maggiore dei 2 numeri)
- **Numeri con segno**
  - in entrambe addizione e sottrazione [A – B = A + (-B)], non c'è overflow se i numeri hanno segno diverso (il valore assoluto del risultato è sempre minore o uguale al più grande). Può esserci overflow solo se i due numeri hanno lo stesso segno (e si ottiene un risultato di segno opposto)

# Rilevazione dell'overflow

- Es.: somma di numeri con segno in complemento a 2 (n = 8 bit)

a)

| | | |
|---|---|---|
| Riporto: | 01 | |
| + 70 | | 01000110 |
| + 80 | | 01010000 |
| + 150 | | 10010110 |

b)

| | | |
|---|---|---|
| Riporto: | 10 | |
| - 70 | | 10111010 |
| - 80 | | 10110000 |
| - 150 | | 01101010 |

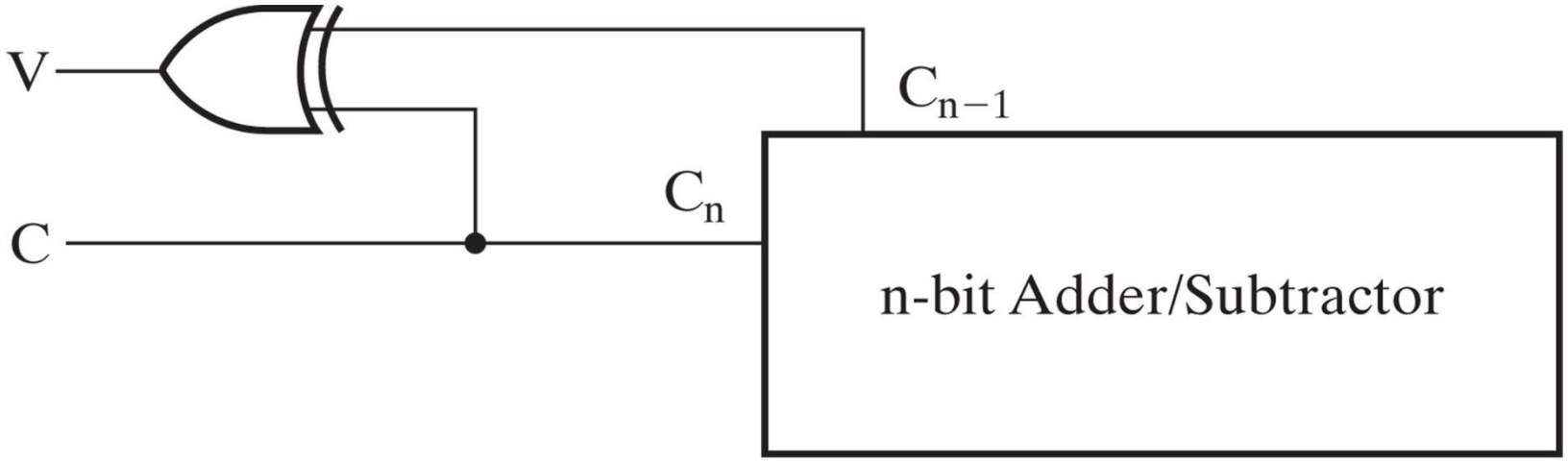
a) Il risultato che dovrebbe essere positivo è negativo (bit di segno è 1 a causa del riporto)

b) Il risultato che dovrebbe essere negativo è positivo

=> Il risultato sarebbe corretto se considerassimo il valore del riporto in uscita come il segno del risultato

- C'è **overflow** **se il bit di riporto nella posizione del segno e il bit di riporto in uscita sono diversi**: quindi mettendo questi due bit in ingresso ad una XOR possiamo rilevare un overflow!

# Circuito per la rilevazione dell'overflow




- **Numeri senza segno**
  - C = 0 indica che non c'è overflow nella somma (carry in uscita è nullo), mentre va corretto il segno per la sottrazione
  - C = 1 indica che c'è overflow per la somma (riporto in uscita nell'ultimo bit) e che il segno è corretto per la sottrazione
- **Numeri con segno**
  - V = 0 indica che il risultato è corretto
  - V = 1 indica overflow (avviene quando il bit di riporto in uscita e quello nella posizione più significativa sono diversi)

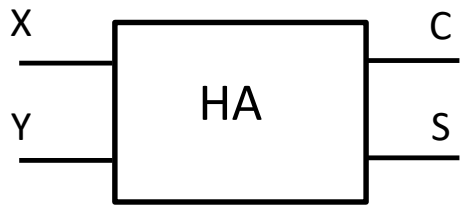
# Rappresentazioni VHDL del sommatore

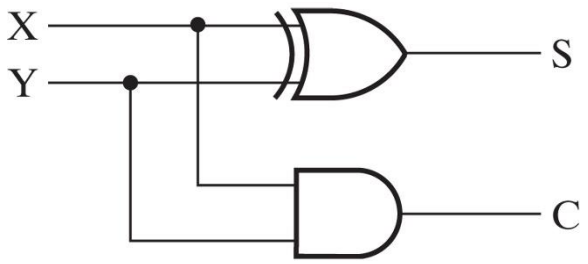
# Sommatore a 4-bit: VHDL gerarchico (1/3)

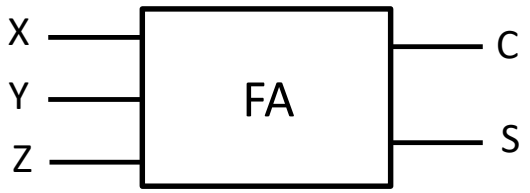
```
-- 4-bit Adder: Hierarchical Dataflow/Structural
-- (See Figures 3-42 and 3-43 for logic diagrams)
library ieee;
use ieee.std_logic_1164.all;
entity half_adder is
   port (x, y : in std_logic;
         s, c : out std_logic);
end half_adder;

architecture dataflow_3 of half_adder is
   begin
      s <= x xor y;
      c <= x and y;
end dataflow_3;

library ieee;
use ieee.std_logic_1164.all;
entity full_adder is
   port (x, y, z : in std_logic;
         s, c : out std_logic);
end full_adder;
```



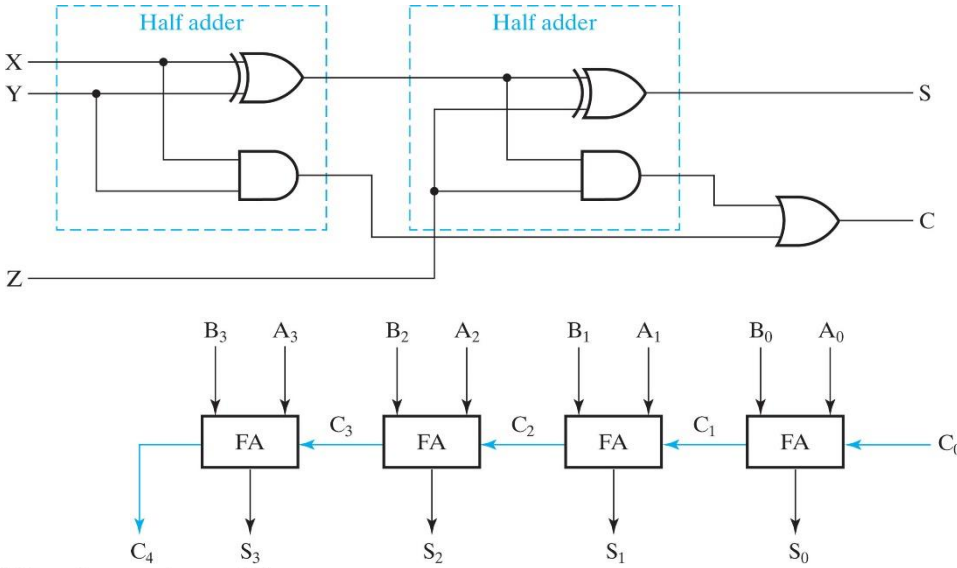



# Sommatore a 4-bit: VHDL gerarchico (2/3)

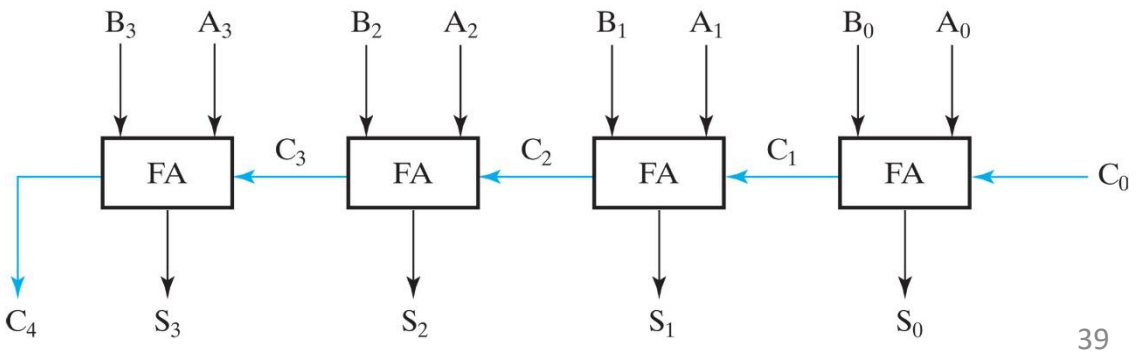
```
architecture struc_dataflow_3 of full_adder is
   component half_adder
      port (x, y : in std_logic;
            s, c : out std_logic);
   end component;
   signal hs, hc, tc: std_logic;
   begin
      HA1: half_adder
        port map  (x, y, hs, hc);
      HA2: half_adder
        port map (hs, z, s, tc);
      c <= tc or hc;
end struc_dataflow_3;

library ieee;
use ieee.std_logic_1164.all;
entity adder_4 is
   port(B, A : in std_logic_vector(3 downto 0);
        C0 : in std_logic;
        S : out std_logic_vector(3 downto 0);
        C4: out std_logic);
end adder_4;
```



# Sommatore a 4-bit: VHDL gerarchico (3/3)

```
architecture structural_4 of adder_4 is
   component full_adder
      port(x, y, z : in std_logic;
           s, c: out std_logic);
   end component;
   signal C: std_logic_vector (4 downto 0);
   begin
      Bit0: full_adder
       port map (B(0), A(0), C(0), S(0), C(1));
      Bit1: full_adder
       port map (B(1), A(1), C(1), S(1), C(2));
      Bit2: full_adder
       port map (B(2), A(2), C(2), S(2), C(3));
      Bit3: full_adder
       port map (B(3), A(3), C(3), S(3), C(4));
      C(0) <= C0;
      C4 <= C(4);
end structural 4;
```



# Sommatore a 4-bit: VHDL behavioral

```
-- 4-bit Adder: Behavioral Description
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity adder_4_b is
   port(B, A : in std_logic_vector(3 downto 0);
        C0 : in std_logic;
        S : out std_logic_vector(3 downto 0);
        C4: out std_logic);
end adder_4_b;

architecture behavioral of adder_4_b is
signal sum: std_logic_vector (4 downto 0);
begin
   sum <= ('0' & A) + ('0' & B) + ("0000" & C0);
   C4 <= sum(4);
   S <= sum(3 downto 0);
end behavioral;
```

# Sommatore a 4-bit: VHDL behavioral

```
-- 4-bit Adder: Behavioral Description
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity adder_4_b is
   port(B, A : in std_logic_vector(3 downto 0);
        C0 : in std_logic;
        S : out std_logic_vector(3 downto 0);
        C4: out std_logic);
end adder_4_b;

architecture behavioral of adder_4_b is
signal sum: std_logic_vector (4 downto 0);
begin
   sum <= ('0' & A) + ('0' & B) + ("0000" & C0);
   C4 <= sum(4);
   S <= sum(3 downto 0);
end behavioral;
```

Non descrive il circuito nel dettaglio, ma definisce il suo comportamento e lascia al sintetizzatore la scelta di come realizzare l'hardware!

Realizza il sommatore come somma di 3 numeri binari di 5 bit, ottenuti concatenando i due addendi da 4 bit con uno '0' iniziale e il bit del riporto in ingresso con 4 '0' iniziali: il riporto in uscita è dato dal MSB, la somma finale è data dai restanti 4 bit meno significativi

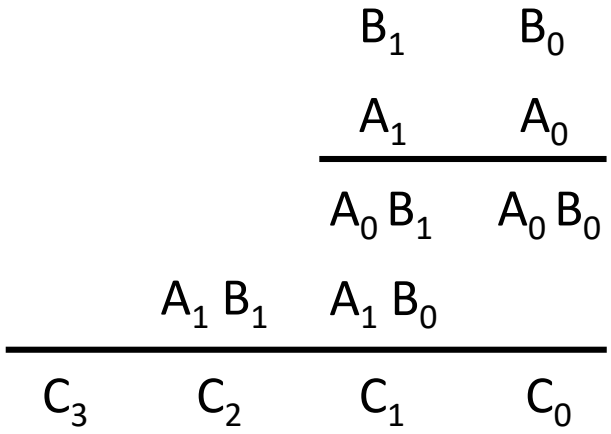
# Cenni al moltiplicatore

# Moltiplicazione binaria: cenni

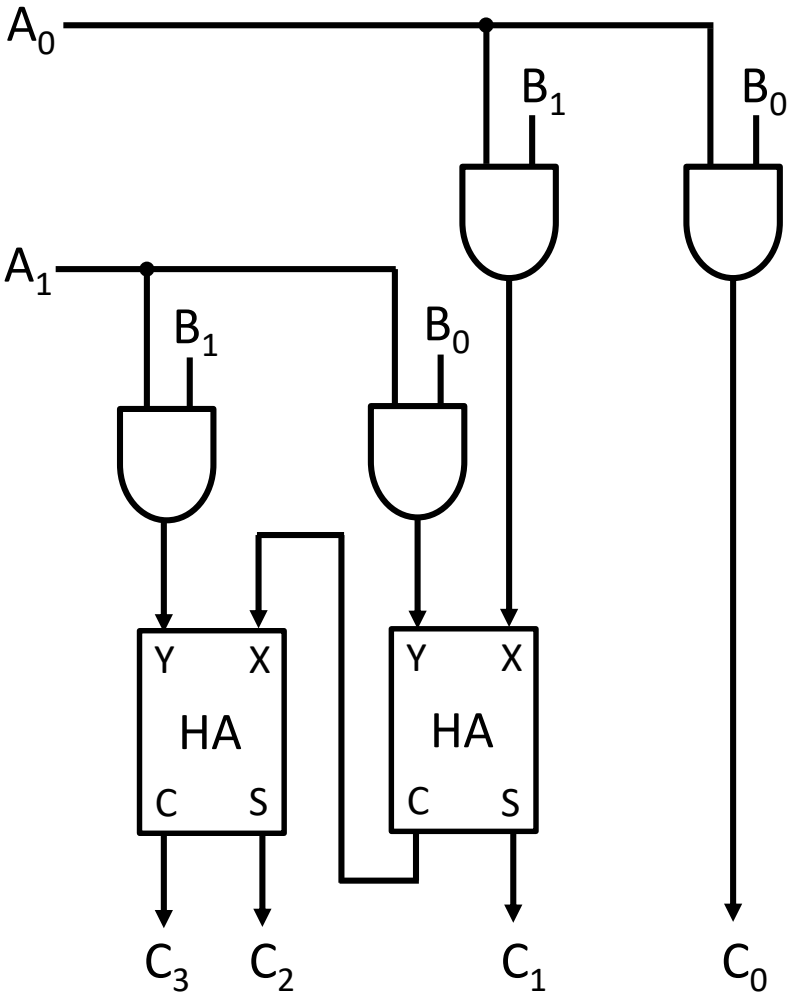
- La moltiplicazione si esegue in modo analogo al sistema decimale
  - Moltiplico il moltiplicando per ciascun bit del moltiplicatore:
    (0 x 0 = 0; 0 x 1 = 0; 1 x 0 = 0; 1 x 1 = 1 -> porta AND)
    partendo dal bit meno significativo. Ognuna di queste moltiplicazioni forma un prodotto parziale.
  - I prodotti parziali sono poi shiftati a sinistra di un bit e infine sommati tra di loro per ottenere il prodotto finale
- Esempio:

```
    1 0 1
  x 1 1 0
  -------
    0 0 0
  1 0 1
1 0 1
-------
1 1 1 1 0
```

# Moltiplicatore a 2 bit




- 4 Porte AND
- 2 Half Adder



# Disclaimer

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