Componenti per l’aritmetica binaria

Engineering Department in Ferrara

Sommario

Codifica binaria di informazioni di tipo numerico e aritmetica binaria

Addizione

Sommatori binari

Sommatore CLA

Applicazioni di n-bit adder

Motivazioni

- **I sistemi di calcolo necessitano di componenti che realizzino operazioni di tipo aritmetico (somme, prodotti ....) su numeri interi e in floating point**
- **Dal punto di vista teorico, le conoscenze che abbiamo ci consentono di realizzare qualsiasi funzione e quindi anche quelle svolte da moltiplicatori e sommatori**
- **Nel caso di interesse, questo approccio non da peró risultati soddisfacenti**
- **Ad esempio la sintesi ottima di reti a due livelli da luogo a funzioni eccessivamente costose e non modulari**

Sommario

Codifica binaria di informazioni di tipo numerico e aritmetica binaria

Addizione

Sommatori binari

Sommatore CLA

Applicazioni di n-bit adder

Codifica delle informazioni di tipo numerico

- **In un sistema di calcolo la rappresentazione di informazioni numeriche riveste particolare importanza**
- **I codici utilizzati per rappresentarle determinano in parte la complessitá e le prestazioni delle unitá che elaborano i dati di tipo numerico**
- **Conviene notare la distinzione fra numeri e loro rappresentazione**

Codici numerici posizionali

- **Codici posizionali (numeri razionali positivi):**
  - $\mathcal{A} = (a_1, a_2, ..., a_b)$**: insieme ordinato di simboli;**
  - $b = |\mathcal{A}|$ **: base;**
- **Parola di codice (lunghezza** $n + k$**):**

$$X = (\overbrace{\alpha_{n-1}\alpha_{n-2}....\alpha_1\alpha_0}^{\textit{parte intera}}, \overbrace{\alpha_{-1}....\alpha_{-k}}^{\textit{parte frazionaria}})$$

- **Valore numerico (informazione):**

$$v(X) = \alpha_{n-1}b^{n-1} + \alpha_{n-2}b^{n-2} + ... + \alpha_1 b^1 + \alpha_0 b^0 + \alpha_{-1}b^{-1} + ... + \alpha_{-k}b^{-k}$$

- **Questa rappresentazione, a meno del segno, sta alla base della rappresentazione numerica detta in virgola fissa**

Esempi di basi

Codice	Base	Alfabeto
binario	**2**	$\{0, 1\}$
ottale	**8**	$\{0, 1, 2, 3, 4, 5, 6, 7\}$
decimale	**10**	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
esadecimale	**16**	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

Notazione: poiché alcuni simboli sono condivisi, occorre denotare i numeri con riferimento alla base. Esempio, $10_2 = 2_{10}$**,** $10_{10} = 1010_2$**, ...**

Rappresentazione dei numeri naturali

- **Ciascuna parola di codice ha n simboli ordinati in senso decrescente di peso**
- **Non si ha la parte frazionaria $X = (\alpha_{n-1}, ..., \alpha_0)$**

$$v(X) = \sum_{i=0}^{n-1} \alpha_i b^i \tag{1}$$

- **Si possono rappresentare numeri naturali $X \in [0, b^n - 1] \subset \mathbb{N}$**

Conversione di base

- **Dato X in base b, si vuole Y in base c in modo che $v(X) = v(Y)$**
- **Si puó utilizzare l'eq. 1, in cui si convertono i coefficienti di X e b nella nuova base c e si effettuano somme e prodotti nella nuova base**
- **Esempio: si vuole convertire 10100_2 in base 10,** $1_{10} \cdot 2^4_{10} + 0_{10} \cdot 2^3_{10} + 1_{10} \cdot 2^2_{10} + 0_{10} \cdot 2^1_{10} + 0_{10} \cdot 2^0_{10} = 20_{10}$
- **Per motivi di praticitá, questo algoritmo ha senso quando la nuova base é 10**

Conversione di base per divisioni ripetute

- **Metodo delle divisioni ripetute per la conversione da base 10 verso altre basi**
- **Sia X la parola di codice in base b che si vuole convertire in Y in base c**
- **Se $Y = [\beta_{m-1}, \beta_{m-2}, ..., \beta_1, \beta_0]$ il suo valore é:**
  $v(Y) = \beta_{m-1}c^{m-1} + \beta_{m-2}c^{m-2} + ... + \beta_1 c + \beta_0$
- **Dividendo (divisione intera) per c si ha: $v(Y) = Qc + R$ (dove Q é il quoziente e R é il resto):**

$$Q = \beta_{m-1}c^{m-2} + \beta_{m-2}c^{m-3} + ... + \beta_1$$
$$R = \beta_0 \quad (2)$$

- **Questo risultato é indipendente dal tipo di base utilizzato per eseguire le operazioni**

Conversione di base per divisioni ripetute

- **Si noti che essendo $R < c$, R ha la stessa rappresentazione in base b e c**
- **Questa operazione puó essere ripetuta in maniera iterattiva fino a calcolare tutti i coefficienti β_i**
- **Le operazioni possono essere svolte utilizzando l'aritmetica nella base di partenza b rappresentando quindi c in base b**
- **Questo algoritmo riveste particolare interesse nella conversione da base 10 a base 2**

Conversione di base per divisioni ripetute

Conversione di 27_{10} in base 2 (si divide per 2):

iterazione	Q	$R = \beta_i$	β_i
0	**27**	**1**	β_0
1	**13**	**1**	β_1
2	**6**	**0**	β_2
3	**3**	**1**	β_3
4	**1**	**1**	β_4
5	**0**		

Quindi $27_{10} = 11011_2$

Esercizi

1. **Si calcoli la rappresentazione in base 2 dei numeri decimali da 0 a 15**
2. **Si calcoli la rappresentazione in base 10 dei seguenti numeri naturali in base 2:** 1010_2, 110010_2, 100100111_2
3. **Si calcoli la rappresentazione in base 2 dei seguenti numeri naturali in base 10:** 1010_{10}, 87_{10}, 747_{10}
4. **Si eseguano le seguenti conversioni** $102_3 \rightarrow Y_{10}$, $643_7 \rightarrow Y_{10}$, $67_{10} \rightarrow Y_8$, $1419_{10} \rightarrow Y_3$
5. **Si converta** $145_6 \rightarrow Y_7$

Conversione base 2 $\Leftrightarrow$ base 16

- **Caso particolare in cui $c = b^k$**
- **Conversione diretta da binario a esadecimale:**
  - **suddividere i bit in gruppi di 4 (da destra) aggiundendo eventuali zeri a sinistra**
  - **sostituire ogni gruppo di 4 bit con il corrispondente simbolo esadecimale**
  - $X_2 = 1110011110 = 0011|1001|1110$ **da cui** $X_{16} = 39E$
- **Nel processo inverso si tratta semplicemente di sostituire i simboli esadecimali con gli equivalenti binari**

Conversione base 10 $\Leftrightarrow$ base 16

- **Si puó sempre passare per la base 2**
- **Conversione da base 16 a base 10:**

$$X_{16} = AB4 \ \rightarrow X_{10} = 10 \cdot 16^2 + 11 \cdot 16^1 + 4 \cdot 16^0 = 2740$$

- **Conversione da base 10 a base 16 (divisioni ripetute):**
- **Esempio** $X_{10} = 1410$

iterazione	Q	$R = \beta_i$	β_i
0	**1410**	**2**	β_0
1	**88**	**8**	β_1
2	**5**	**5**	β_2
3	**0**	**0**	

- **Da cui** $X_{16} = 582$

Sommario

Codifica binaria di informazioni di tipo numerico e aritmetica binaria

Addizione

Sommatori binari

Sommatore CLA

Applicazioni di n-bit adder

Addizione in base 2

- **Si puó applicare l'algoritmo di somma per colonne utilizzato in base 10**
- **Siano A e B i due numeri binari da sommare che vanno disposti su due righe**
- **Per ogni colonna i, partendo da quella di peso 0, é sufficiente usare $s_i = (a_i + b_i + r_i)_{mod2}$ e $r_{i+1} = (a_i + b_i + r_i)/2$ (riporto)**
- **Esempio**

r_i	**1**	**1**	**1**	**1**	**0**
A		**1**	**0**	**0**	**1**	**+**
B		**0**	**1**	**1**	**1**	**=**
$A+B$	**1**	**0**	**0**	**0**	**0**	

Aritmetica base 16

- **Le somme possono essere fatte convertendo gli operandi in base 10 o 2, oppure direttamente in base 16**
- **Altrimenti, se A e B sono i due numeri da sommare, é sufficiente usare $s_i = (a_i + b_i + r_i)_{mod16}$ e $r_{i+1} = (a_i + b_i + r_i)/16$ (riporto)**
- **Esempio:**

r_i	**0**	**1**	**1**	**0**	**0**	
A		**A**	**4**	**7**	**B**	**+**
B		**1**	**E**	**F**	**2**	**=**
$A+B$		**C**	**3**	**6**	**D**	

Sottrazione

- **Se $A \geq B$ si puó utilizzare un algoritmo di sottrazione per colonne con un prestito invece del riporto**
- **Siano A e B i due numeri binari da sottrarre che vanno disposti su due righe**
- **Per ogni colonna i, partendo da quella di peso 0, é sufficiente usare $s_i = (a_i + b_i + p_i)_{mod2}$ e $p_{i+1} = ((1 - a_i) + b_i + r_i)/2$ (riporto)**
- **Cosa succede se $A < B$?**

Sommario

Codifica binaria di informazioni di tipo numerico e aritmetica binaria

Addizione

Sommatori binari

Sommatore CLA

Applicazioni di n-bit adder

Esempio di funzione aritmetica: 3-bit adder

- **Si vuole realizzare un sommatore a 2 operandi per numeri interi positivi rappresentati su 3 bit.**
- **Siano $A = \{a_2, a_1, a_0\}$ e $B = \{b_2, b_1, b_0\}$ tali parole**
- **Il risultato é rappresentabile su 4 bit: $S = \{s_3, s_2, s_1, s_0\}$**
- **Si supponga di sintetizzare le funzioni s_i come espressioni SP a uscita singola**

3-bit adder

a1a0 \ b1b0	00	01	11	10
00	0000	0001	0011	0010
01	0001	0010	0100	0011
11	0011	0100	0110	0101
10	0010	0011	0101	0100

a2b2=00

a1a0 \ b1b0	00	01	11	10
00	0100	0101	0111	0110
01	0101	0110	1000	0111
11	0111	1000	1010	1001
10	0110	0111	1001	1000

a2b2=01

a1a0 \ b1b0	00	01	11	10
00	0100	0101	0111	0110
01	0101	0110	1000	0111
11	0111	1000	1010	1001
10	0110	0111	1001	1000

a2b2=10

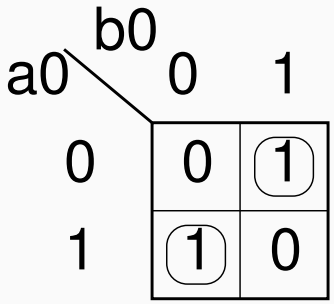
a1a0 \ b1b0	00	01	11	10
00	1000	1001	1011	1010
01	1001	1010	1100	1011
11	1011	1100	1110	1101
10	1010	1011	1101	1100

a2b2=11

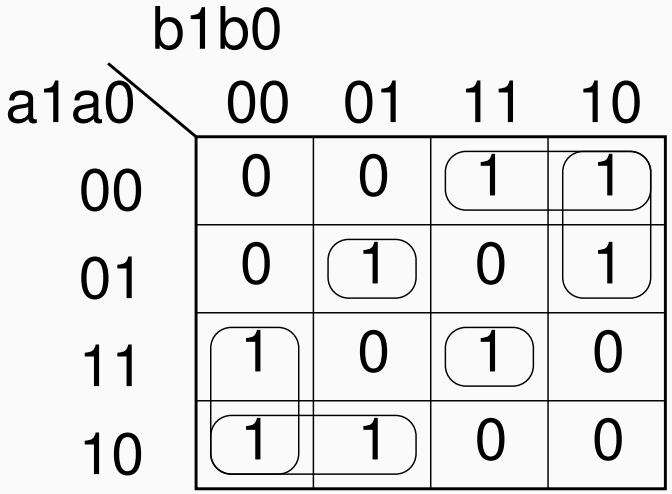
s3s2s1s0

3-bit adder

Partiamo dal bit di minore peso:



s0=a0b0'+a0'b0

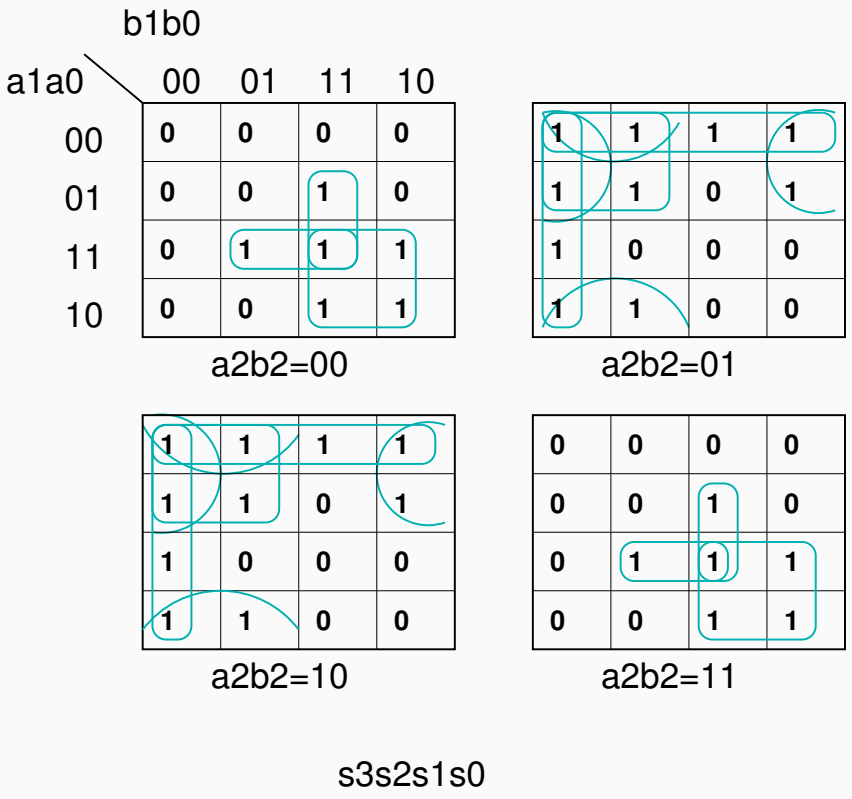


s1=a1'a0'b1+a1'b1b0'+
a1a0'b1'+a1'a0b1'b0
a1a0b1b0+a1b1'b0'

s2=.... (costo crescente)

3-bit adder

Le funzioni hanno un costo che a partire dal bit di minor peso (s_0) aumenta molto rapidamente



s3=a2'b2'a1b1+a2'b2'a1a0b0+a2'b2'a0b1b0+
a2'b2b1'b0'+a2'b2a1'b0'+a2'b2a1'a0'+a2'b2a1'a0'
a2b2'b1'b0'+a2b2'a1'b1'+a2b2'a1'a0'+a2b2'b0'b1'
a2b2a1b1+a2b2a1a0b0+a2b2b0a1a0

Problemi nella realizzazione di sommatori binari

- **Con l'aumentare della dimensione delle parole (n), il costo di sommatori binari realizzati come reti a 2 livelli aumenta molto rapidamente**
- **L'utilizzo dei metodi di sintesi multilivello da luogo a miglioramenti relativi**
- **Le soluzioni che si ottengono non risultano modulari**
- **Come alternativa si vedrá un metodo che é basato sulla realizzazione hardware dell'algoritmo di somma per colonne**

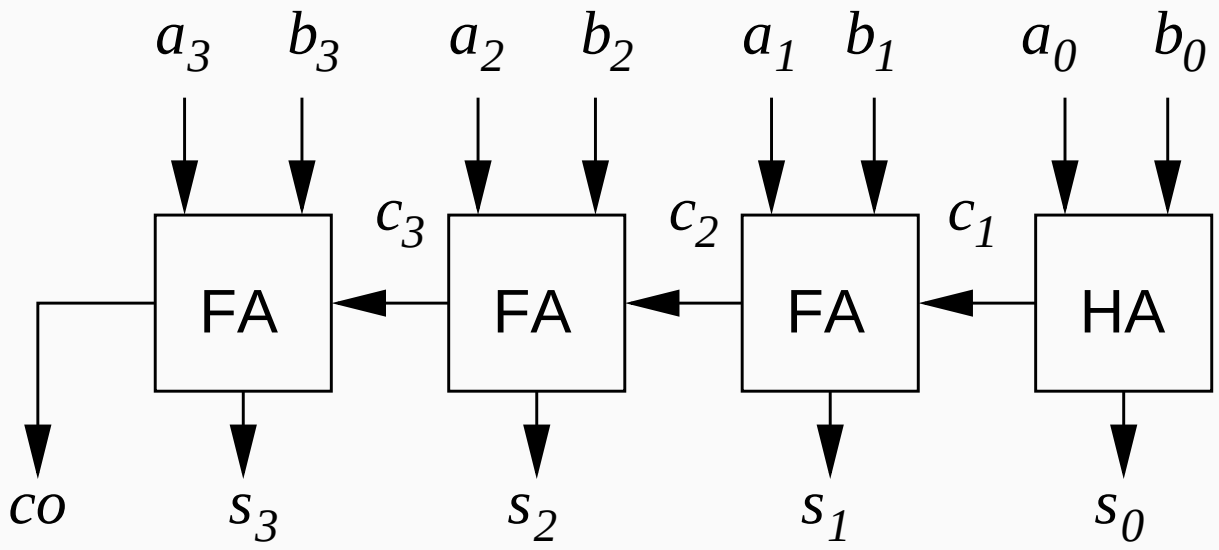
Ripple-carry adder

- **Algoritmo di somma per colonne di due parole di n bit basato sulla propagazione del riporto (carry)**
- $s_0 = (a_0 + b_0)_{mod2}$, $c_1 = (a_0 + b_0)/2$
- $s_i = (a_i + b_i + c_i)_{mod2}$, $c_{i+1} = (a_i + b_i + c_i)/2$
- $co = c_n$
- **ove $+$ é la somma aritmetica e $/$ é la divisione intera**

$$
\begin{array}{ccccc}
c_4 & c_3 & c_2 & c_1 & \\
 & a_3 & a_2 & a_1 & a_0 + \\
 & b_3 & b_2 & b_1 & b_0 = \\
\hline
co & s_3 & s_2 & s_1 & s_0
\end{array}
$$

Ripple-carry adder

La descrizione funzionale si traduce in questo schema:



- **I blocchi che gestiscono i bit di indice $i > 0$ sono detti full-adder (FA)**
- **Quello che gestisce il caso $i = 0$ é detto half-adder (HA) (vedremo poi il motivo per cui conviene sostuirlo con un FA)**

Half-adder

É un componente ampiamente utilizzato nell'aritmetica binaria.
Ingressi a e b, uscite s e c_{out}

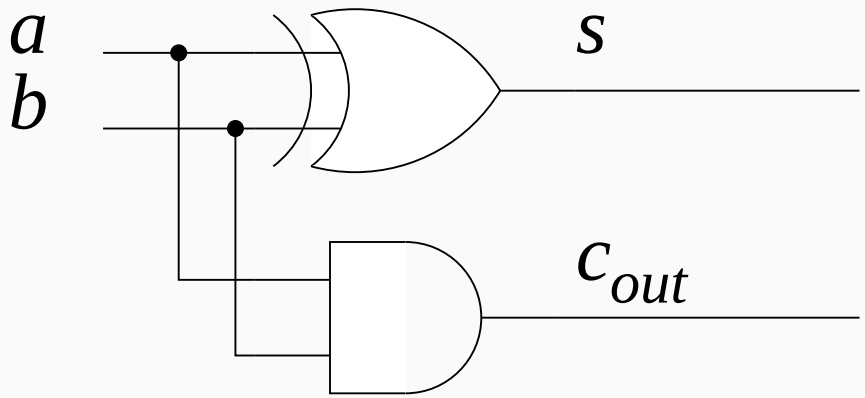
$a\,b$	$c_{out}\,s$
00	**00**
01	**01**
10	**01**
11	**10**

$$c_{out} = ab$$

$$s = ab' + a'b$$

Realizzazione di un HA

$c_{out} = ab$ $\qquad$ $s = ab' + a'b = a \oplus b$



Il gate EXOR che realizza la somma modulo 2 é un componente che puó essere realizzato in tecnologia CMOS al livello switch (in maniera píu complessa di NAND e NOR)

Full-adder

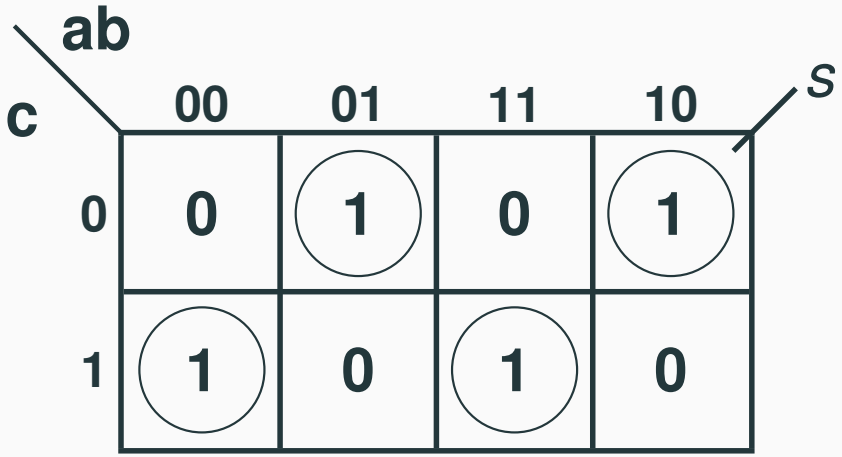
Il sommatore completo é anch'esso un componente fondamentale per l'aritmetica binaria. Ingressi a, b, e c_{in}, uscite s e c_{out}

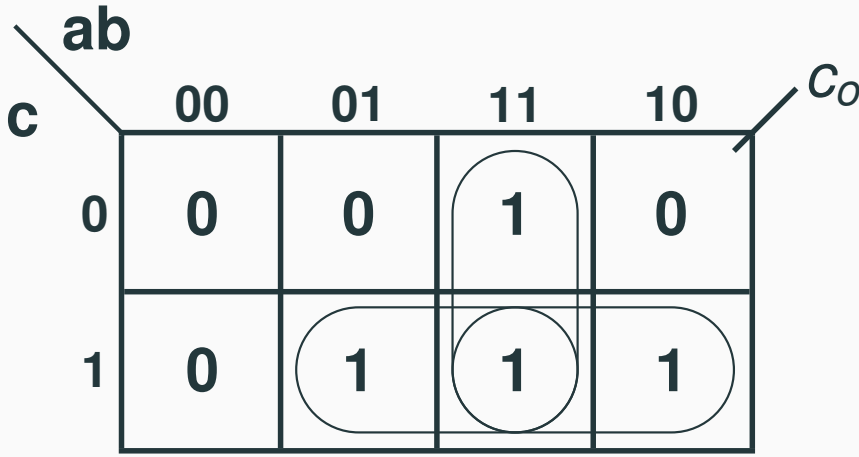
$a\,b\,c_{in}$	$c_{out}\,s$
000	00
001	01
010	01
011	10
100	01
101	10
110	10
111	11

$$c_{out} = ab + ac_{in} + bc_{in}$$

$$s = a'b'c_{in} + a'bc_{in} + ab'c'_{in} + abc_{in}$$

Realizzazione di un FA





$$s = a'b'c_{in} + a'bc'_{in} + ab'c'_{in} + abc_{in}$$

$$c_{out} = ab + ac_{in} + bc_{in}$$

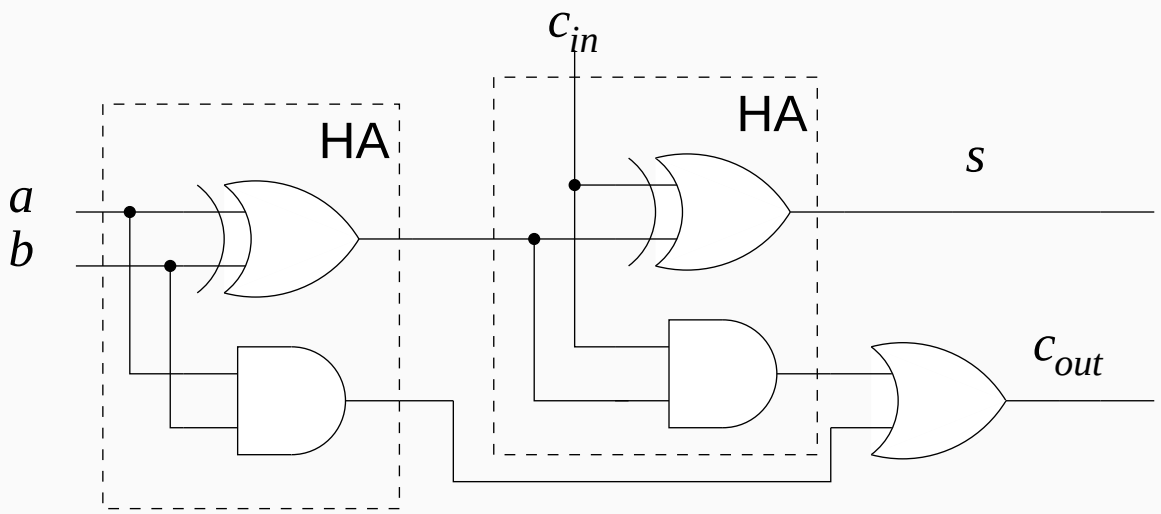
Realizzazione di un FA

$$
\begin{aligned}
c_{out} &= ab + ac_{in} + bc_{in} \\
&= ab + ac_{in}(b' + b) + bc_{in}(a' + a) \\
&= ab + ab'c_{in} + abc_{in} + a'bc_{in} + abc_{in} \\
&= ab + abc_{in} + abc_{in} + c_{in}(ab' + a'b) \\
&= ab + c_{in}(a \oplus b)
\end{aligned}
$$

$$
\begin{aligned}
s &= a'b'c_{in} + a'bc'_{in} + ab'c'_{in} + abc_{in} \\
&= c_{in}(a'b' + ab) + c'_{in}(ab' + a'b) \\
&= c_{in}(ab' + a'b)' + c'_{in}(ab' + a'b) \\
&= c_{in} \oplus (ab' + a'b) \\
&= c_{in} \oplus (a \oplus b)
\end{aligned}
$$

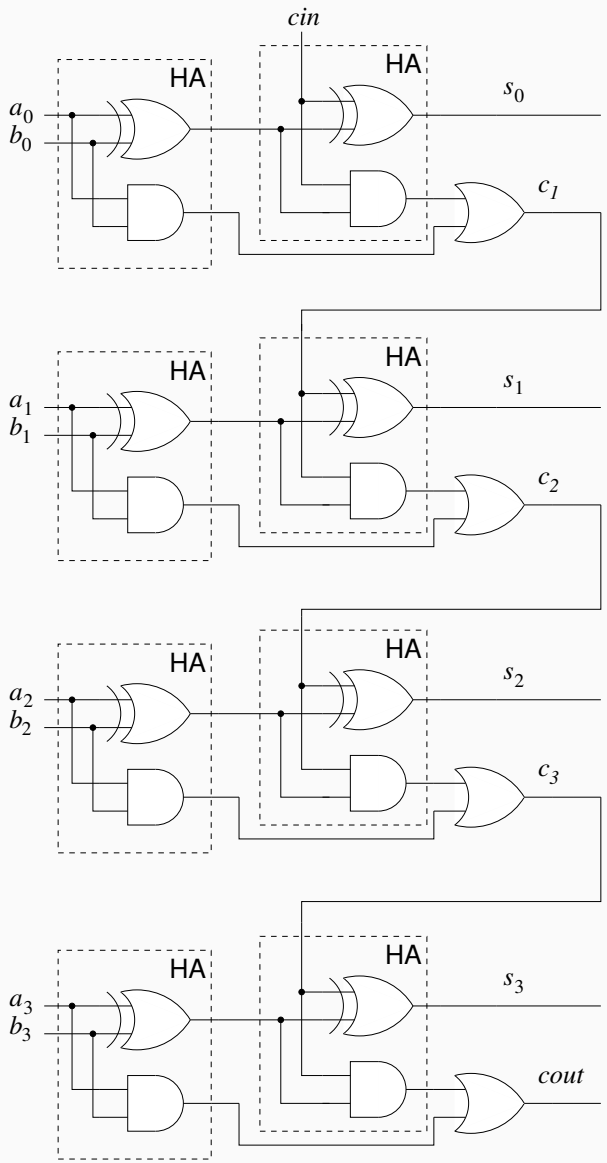
Realizzazione di un FA

La realizzazione delle equazioni viste in precedenza consente di riconoscere la presenza di due HA



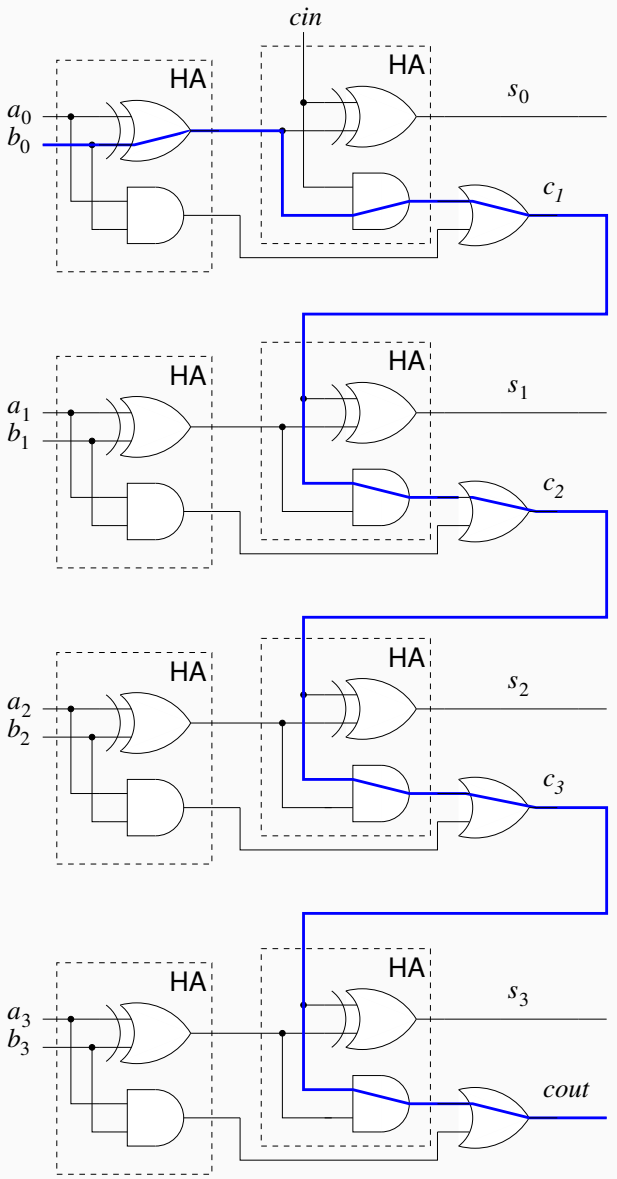
Struttura di un n-bit adder (n=4)

Struttura al livello gate di un n-bit adder. Si noti che l'half-adder che somma a_0 e b_0 é stato sostituito da un full-adder in modo da poter utilizzare un carry-in di ingresso.



Vantaggi e svantaggi

- **Vantaggi: modularitá e ridotto costo**
- **Svantaggi: ritardo**



Sommario

Codifica binaria di informazioni di tipo numerico e aritmetica binaria

Addizione

Sommatori binari

Sommatore CLA

Applicazioni di n-bit adder

Sommatore carry-look ahead (CLA)

- **Per superare i problemi dovuti alle prestazioni del sommatore ripple-carry sono stati proposti diversi sommatori**
- **Uno dei primi ad essere stato proposto é il sommatore carry look ahead (CLA)**
- **Il sommatore CLA utilizza una rete a 3 livelli che si occupa di calcolare i carry di un n-bit adder senza bisogno di propagare il riporto**
- **Il suo ritardo risulta quindi inferiore anche se al crescere di n tale vantaggio di riduce e il costo della rete puó diventare eccessivo**

Sommatore carry-look ahead

- **Nel sommatore ripple-carry, in uscita all'i-mo FA si ha riporto ($c_{i+1} = 1$) se a_i e b_i hanno valori tali da produrre un riporto in uscita indipendentemente da c_i o se il loro valore é tale da garantire la propagazione di $c_i = 1$**
- **Generazione del riporto per il bit di peso i (carry generate):** $g_i = a_i b_i$
- **Propagazione del riporto per il bit di peso i (carry propagate):** $p_i = a_i \oplus b_i$
- $c_{i+1} = g_i + p_i c_i$
- **L'applicazione iterattiva di questa formula porta alla logica di generazione dei riporti di un CLA**
- **I bit della somma sono calcolati come:** $s_i = a_i \oplus b_i \oplus c_i$

Sommatore carry look ahead (n=4)

$$
\begin{aligned}
c_1 &= g_0 + p_0 c_{in} \\
c_2 &= g_1 + p_1 c_1 \\
c_3 &= g_2 + p_2 c_2 \\
c_{out} &= g_3 + p_3 c_3
\end{aligned}
$$

Sostituendo iterattivamente

$$
\begin{aligned}
c_1 &= g_0 + p_0 c_{in} \\
c_2 &= g_1 + p_1 (g_0 + p_0 c_{in}) \\
c_3 &= g_2 + p_2 (g_1 + p_1 (g_0 + p_0 c_{in})) \\
c_{out} &= g_3 + p_3 (g_2 + p_2 (g_1 + p_1 (g_0 + p_0 c_{in})))
\end{aligned}
$$

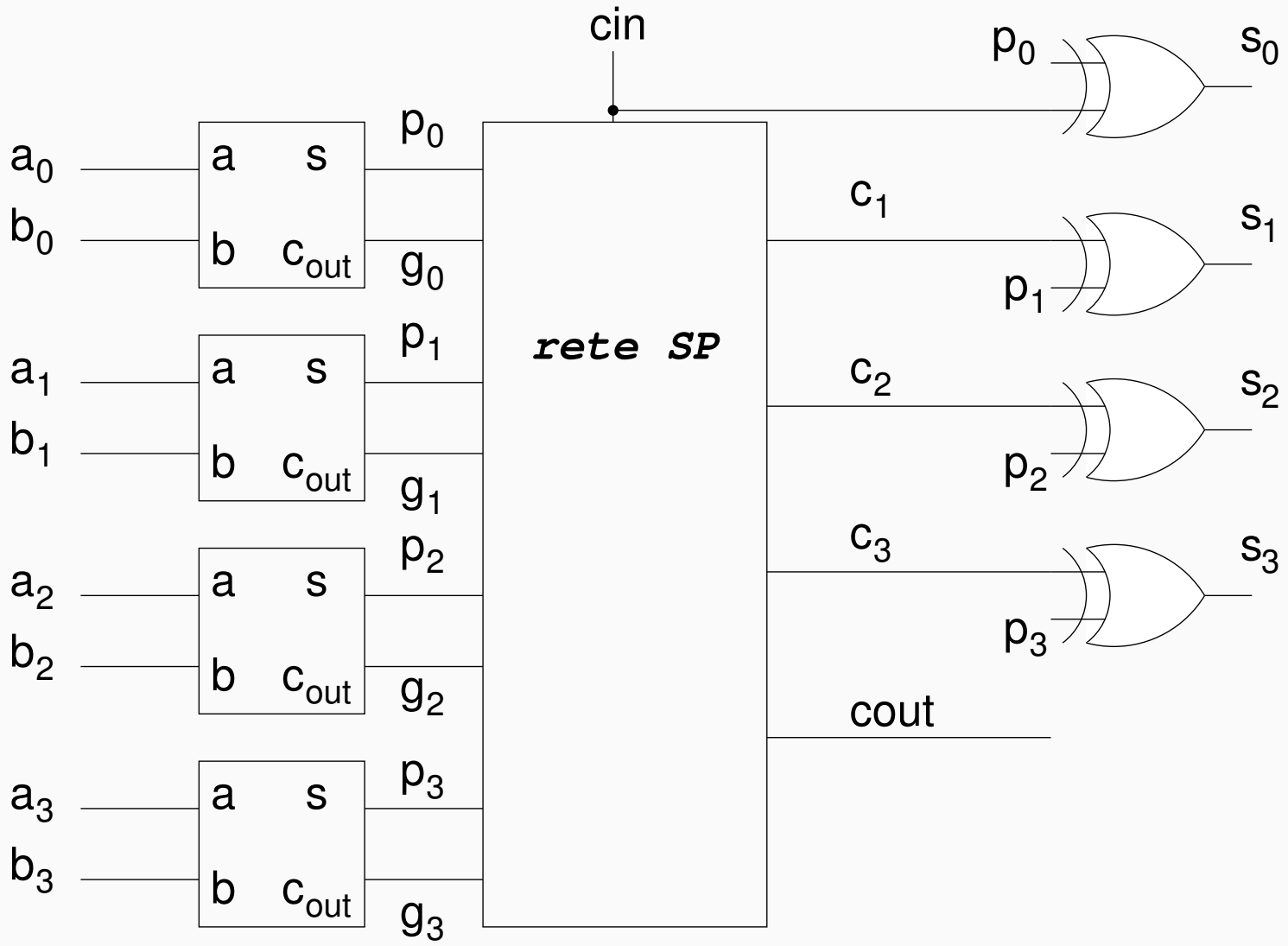
Sommatore carry look ahead (n=4)

Applicando la proprietá distributiva

$$
\begin{aligned}
c_1 &= g_0 + p_0 c_{in} \\
c_2 &= g_1 + p_1 g_0 + p_1 p_0 c_{in} \\
c_3 &= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_{in} \\
c_{out} &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_{in}
\end{aligned}
$$

Ciascun bit di carry viene espresso come un espressione SP a due livelli in funzione di dei bit di carry generate e propagate e del carry-in esterno

Struttura di un sommatore CLA per n=4



Confronto fra ripple-carry e carry look ahead adder

- **Il sommatore ripple-carry ha n FA in cascata e il suo cammino piú lento (cammino critico) attraversa $2n$ gate**
  - **supponendo che i gate abbiano ritardo d, il ritardo del cammino critico é pari a $2dn$ ($8d$ se $n = 4$)**
- **Nel sommatore CLA vengono attraversati $1 + 2 + 1$ gate**
  - **Il ritardo é apparentemente $4d$**
  - **In realtá la rete SP ha gate con fan-in elevato il cui ritardo é superiore a quello (d) dei gate a 2 ingressi utilizzati nei FA**
  - **Il ritardo di tali gate cresce con n**
  - **Il costo del ripple carry adder é proporzionale a n, quello del CLA aumenta con n^2**

Sommario

Codifica binaria di informazioni di tipo numerico e aritmetica binaria

Addizione

Sommatori binari

Sommatore CLA

Applicazioni di n-bit adder

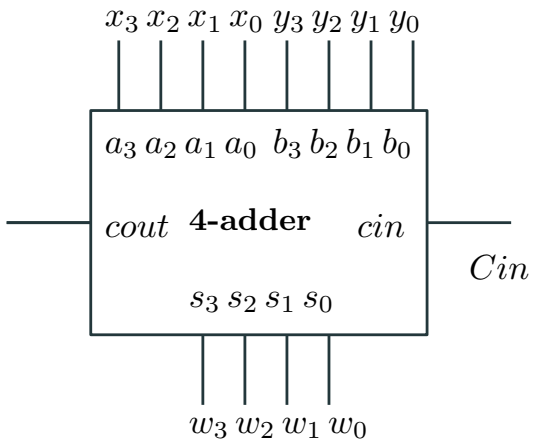
n-bit adder: applicazioni

- **Si mostrano qui alcune applicazioni di adder a n bit dove i singoli adder possono essere di qualsiasi tipo**
- **Sommatore a kn-bit** ▸ soluzione
- **Sommatori a piú operandi** ▸ soluzione
- **Valutazione di semplici espressioni aritmetiche**
- **Contatore di uni**

Sommatore a kn bit usando k sommatori a n bit

▸ return

Sia $k = 4$ e $n = 4$

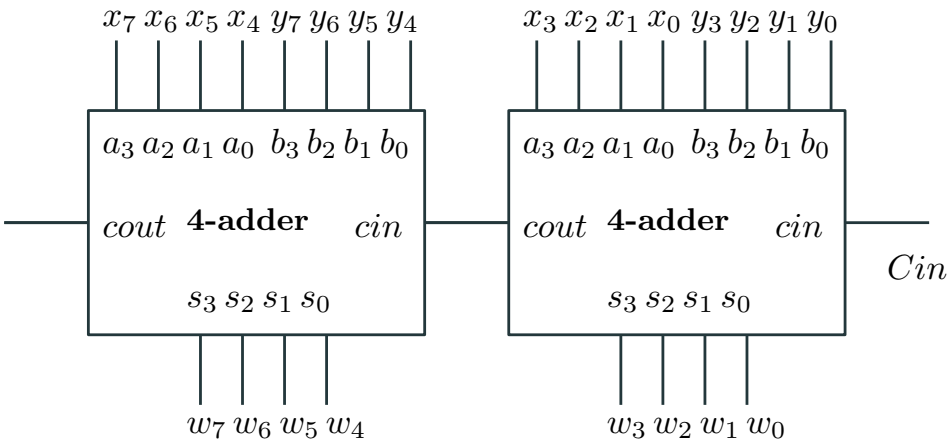


1. si prendono i 4 bit di minor peso di x e y e si connettono a un 4 bit adder

Sommatore a kn bit usando k sommatori a n bit

▸ return

Sia $k = 4$ e $n = 4$

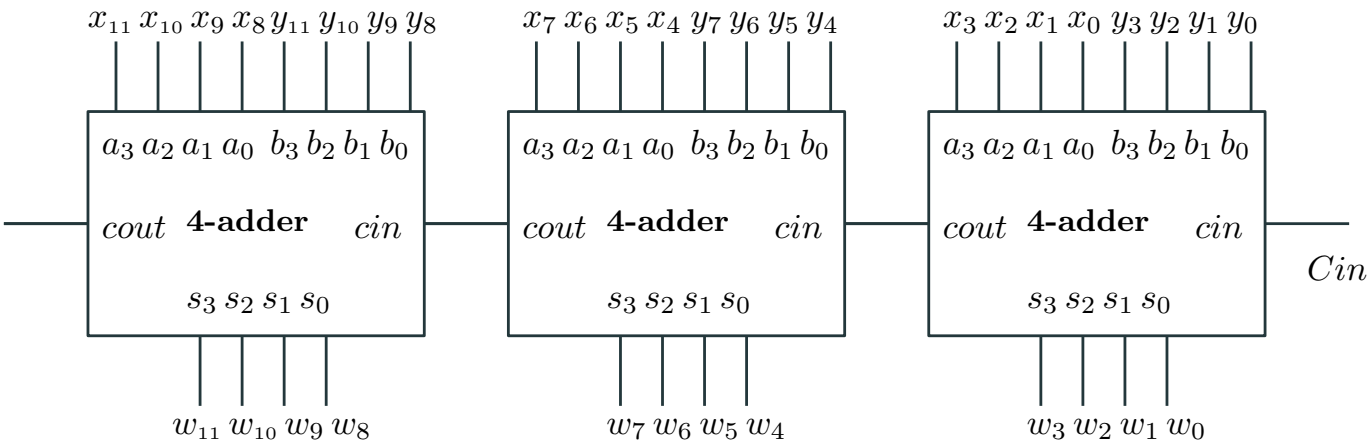


1. si prendono i 4 bit di minor peso di x e y e si connettono a un 4 bit adder
2. si procede aggiungendo 4 bit adder connessi in modo da avere il *cout* di uno connesso al *cin* del successivo

Sommatore a kn bit usando k sommatori a n bit

▸ return

Sia $k = 4$ e $n = 4$

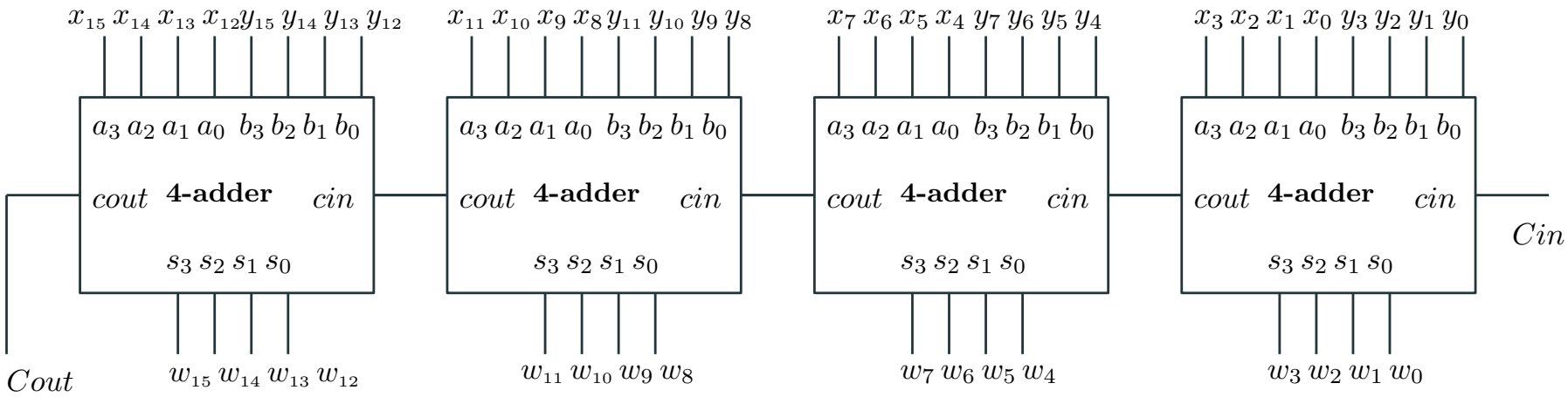


1. si prendono i 4 bit di minor peso di x e y e si connettono a un 4 bit adder
2. si procede aggiungendo 4 bit adder connessi in modo da avere il *cout* di uno connesso al *cin* del successivo

Sommatore a kn bit usando k sommatori a n bit

▸ return

Sia $k = 4$ e $n = 4$



1. si prendono i 4 bit di minor peso di x e y e si connettono a un 4 bit adder
2. si procede aggiungendo 4 bit adder connessi in modo da avere il *cout* di uno connesso al *cin* del successivo

Sommatore a 3 operandi da 4 bit ciascuno

▸ return

Si vuole calcolare $S = X + Y + W$

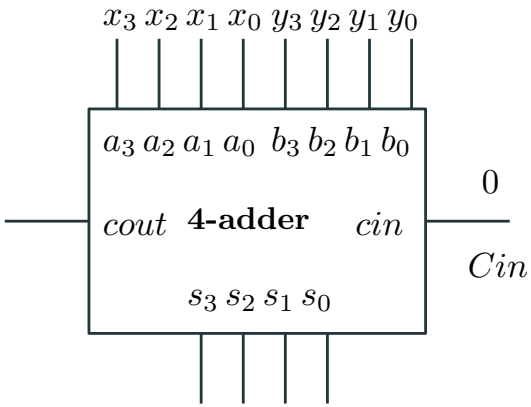
1. in questa soluzione non vengono perse informazioni: S ha un numero di bit sufficiente per contenere il risultato
2. $S \leq 45$ e quindi deve avere 6 bit

Sommatore a 3 operandi da 4 bit ciascuno

▸ return

Si vuole calcolare $S = X + Y + W$

1. in questa soluzione non vengono perse informazioni: S ha un numero di bit sufficiente per contnere il risultato
2. $S \leq 45$ e quindi deve avere 6 bit
3. per prima cosa si sommano X e Y con un 4 bit adder di cui vanno usate tutte le uscite

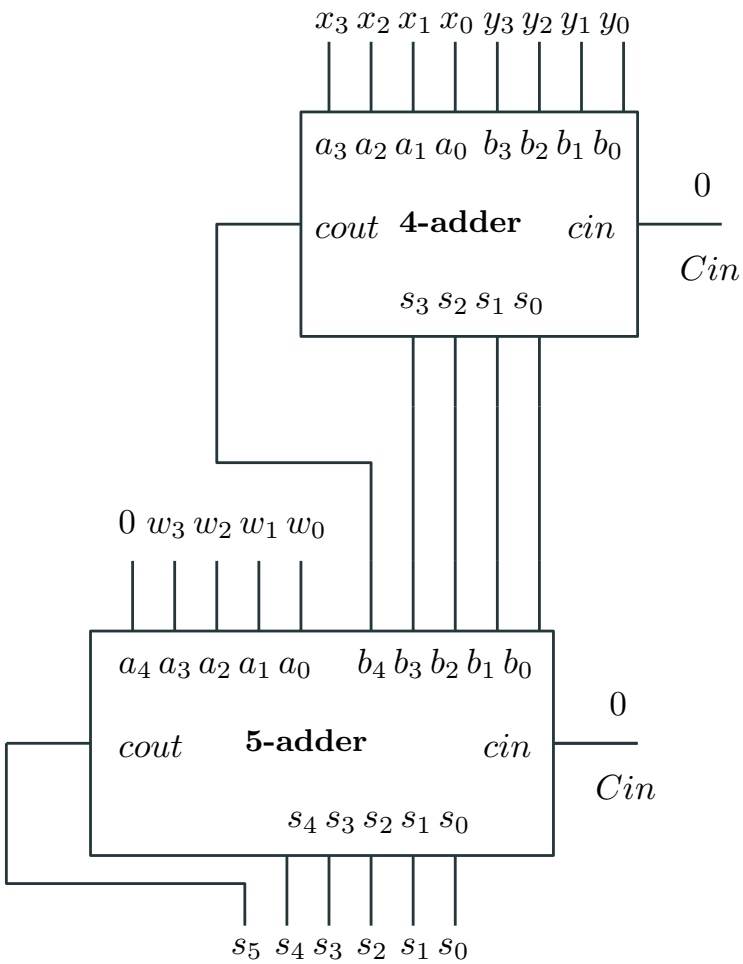


Sommatore a 3 operandi da 4 bit ciascuno

▸ return

Si vuole calcolare $S = X + Y + W$

1. in questa soluzione non vengono perse informazioni: S ha un numero di bit sufficiente per contnere il risultato
2. $S \leq 45$ e quindi deve avere 6 bit
3. per prima cosa si sommano X e Y con un 4 bit adder di cui vanno usate tutte le uscite
4. poi bisogna utilizzare un 5 bit adder per sommare W a $X + Y$ e anche in questo caso si utilizzano le 6 uscite



Conclusioni

- **I componenti per l'aritmetica binaria giocano un ruolo rilevante nelle prestazioni delle CPU e di altri sistemi digitali**
- **Siamo partiti dal caso piú semplice della somma di numeri naturali vedendo due possibili soluzioni per l'implementazione degli adder**
- **Ne esistono diverse altre**
- **Vedremo in seguito il loro utilizzo nelle CPU**