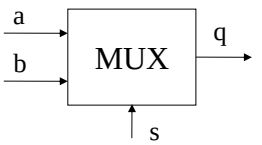
Componenti di un sistema digitale

Il Multiplexer 2x1

Dispositivo che permette di selezionare uno degli *n* ingressi e presentarlo in uscita

- Con n linee di ingresso un multiplexer richiede un numero di linee di comando pari a $\lceil \lg_2(n) \rceil$



Descrizione:

```
q <= a when s= '0' else
     b;
```

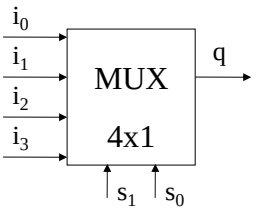
Tabella della verità

a	b	s	q
0	-	0	0
1	-	0	1
-	0	1	0
-	1	1	1

f(a,b,s)= a s'+b s

Il Multiplexer 4x1

Con 4 linee di ingresso sono richieste due linee di comando



Descrizione:

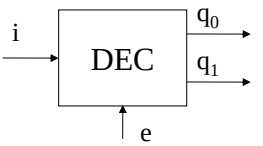
```
q <=    i0  when s1 = '0' and s0 = '0' else
        i1  when s1 = '0' and s0 = '1' else
        i2  when s1 = '1' and s0 = '0' else
        i3;
```

$f(i_3,i_2,i_1,i_0,s_1,s_0)= i_3s_1s_0+i_2s_1s_0'+i_1s_1's_0+i_0s_1's_0'$

Il decodificatore (in logica diretta)

Un decodificatore (1 su *m*) accetta in ingresso un codice di *n* bit e presenta in uscita *m*=2*n* linee, sulle quali asserisce solo quella che corrisponde alla codifica in ingresso

- Numerando le linee di uscita da 0 a 2*n*-1, viene asserita quella che corrisponde al numero presente in ingresso



Descrizione:

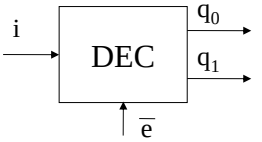
```
q0 <=  '1' when e ='1' and i='0' else
       '0';

q1 <=  '1' when e ='1' and i='1' else
       '0';
```

$q_0= ei'$

$q_1= ei$

Il decodificatore (in logica negata)



Descrizione:

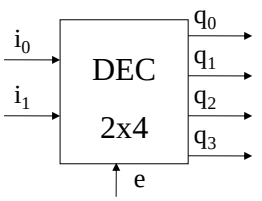
```
q0 <=  '0' when e ='0' and i='0' else
       '1';

q1 <=  '0' when e ='0' and i='1' else
       '1';
```

q_0'= e'i' $\qquad$ q_0 = e+i

q_1'= e'i $\qquad$ q_1 = e+i'

Il decodificatore 2x4 (in logica diretta)



Descrizione:

```
q0 <=   '1' when e ='1' and i0='0' and i1='0' else
        '0';

q1 <=   '1' when e ='1' and i0='0' and i1='1' else
        '0';

q2 <=   '1' when e ='1' and i0='1' and i1='0' else
        '0';

q3 <=   '1' when e ='1' and i0='1' and i1='1' else
        '0';
```

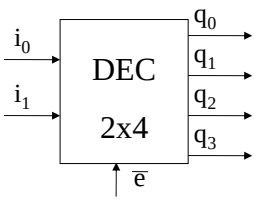
q_0= e i_1' i_0'

q_1= e i_1' i_0

q_2= e i_1 i_0'

q_3= e i_1 i_0

Il decodificatore 2x4 (in logica negata)



Descrizione:

```
q0 <=   '0' when e ='0' and i0='0' and i1='0' else
        '1';

q1 <=   '0' when e ='0' and i0='0' and i1='1' else
        '1';

q2 <=   '0' when e ='0' and i0='1' and i1='0' else
        '1';

q3 <=   '0' when e ='0' and i0='1' and i1='1' else
        '1';
```

$q_0' = e' i_1' i_0'$ $\quad q_0 = e + i_1 + i_0$

$q_1' = e' i_1' i_0$ $\quad q_1 = e + i_1 + i_0'$

$q_2' = e' i_1 i_0'$ $\quad q_2 = e + i_1' + i_0$

$q_3' = e' i_1 i_0$ $\quad q_3 = e + i_1' + i_0'$

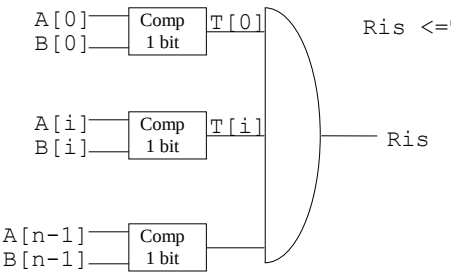
Comparatore a n bit (A==B)


```
Ris<= '1' when A=B else
      '0';
```

Descrizione del comparatore

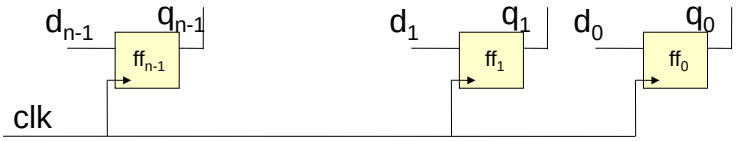
```
for i IN 0 to n-1 LOOP
 T[i]:=Comparatore_1Bit(A[i],B[i])
```

```
Ris <=T[0] and T[1] and ... and T[n-1]
```



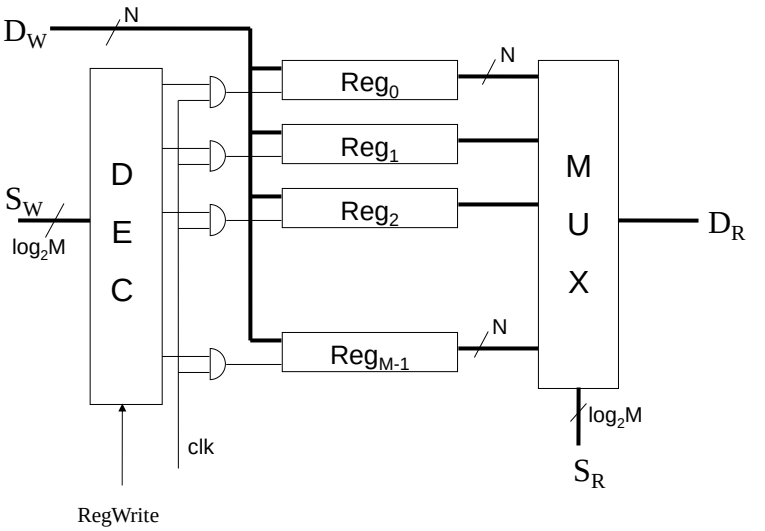
Registro

Un registro di n bit e' un vettore di n ff



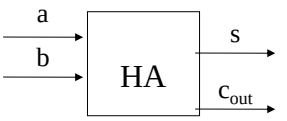
Il registro di n bit lo rappresenteremo come segue:

Il Register file



Sommatore 1 bit (Half/Full Adder)

Half adder

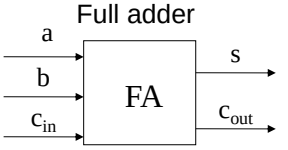


$s = a'b + ab'$

$c_{out} = ab$

a	b	s	c_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full adder

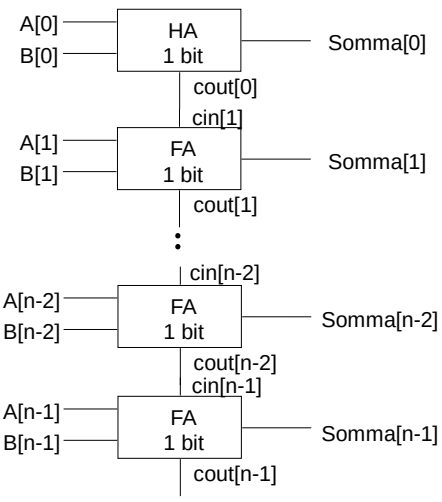


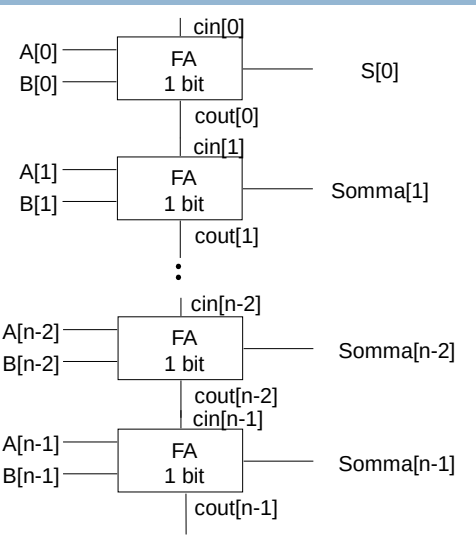
$s = a'b'c_{in} + a'bc_{in}' + abc_{in} + ab'c_{in}'$

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

a	b	c_{in}	s	c_{out}
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

Sommatore a n bit
Propagazione di riporto





Sommatori con anticipo di riporto

Nella generica cella *i*-esima di un sommatore a propagazione di riporto, il riporto in uscita c_{i+1} deriva da

$$C_{i+1} = a_i b_i + a_i' b_i C_i + a_i' b_i' C_i = a_i b_i + (a_i \oplus b_i) C_i$$

- Una componente generata localmente $a_i b_i$

Vale 1 se i bit *i*-esimi degli addendi valgono 1

- E una propagata dovuta al riporto di ingresso

Vale 1 se c_i=1 e se almeno uno dei bit *i*-esimi degli addendi è 1

$$c_{i+1} = G_i + P_i c_i$$

$$a_i b_i \qquad a_i \oplus b_i$$

Sommatori con anticipo di riporto

$$c_{i+1} = G_i + P_i c_i = a_i b_i + a_i \oplus b_i c_i$$

Il procedimento può essere iterato su c_i

$$c_i = G_{i-1} + P_{i-1} c_{i-1} = a_{i-1} b_{i-1} + a_{i-1} \oplus b_{i-1} c_{i-1}$$

e, riportando questa espressione in quella che fornisce c_{i+1} si ricava

$$c_{i+1} = a_i b_i + a_i \oplus b_i (a_{i-1} b_{i-1} + a_{i-1} \oplus b_{i-1} c_{i-1})$$

e così via fino ad esprimere il riporto c_{i+1} in funzione dei bit da *i* a 0 dei due addendi

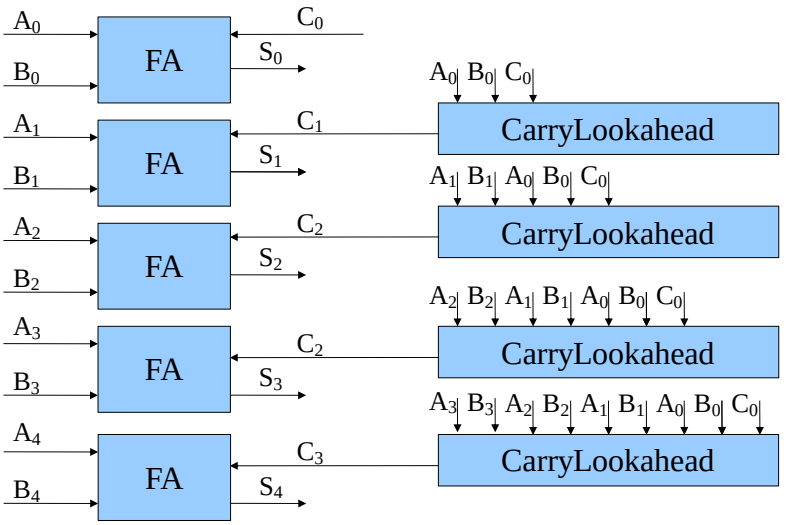
Sommatore con anticipo di riporto

$C_1=G_0+P_0C_0=A_0B_0+(A_0\oplus B_0)C_0$

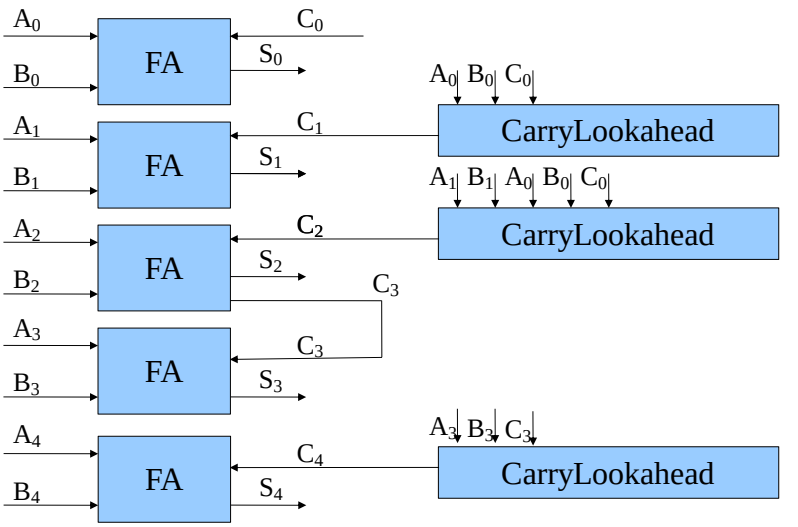
$C_2=G_1+P_1C_1=A_1B_1+(A_1\oplus B_1)(A_0B_0+(A_0\oplus B_0)C_0)=$
$=A_1B_1+(A_1\oplus B_1)(A_0B_0)+(A_1\oplus B_1)(A_0\oplus B_0)C_0$

$C_3=G_2+P_2C_2=$
$=A_2B_2+(A_2\oplus B_2)(A_1B_1+(A_1\oplus B_1)(A_0B_0+(A_0\oplus B_0)C_0)=$
$=A_2B_2+(A_2\oplus B_2)(A_1B_1)+(A_2\oplus B_2)(A_1\oplus B_1)(A_0B_0)$
$+(A_2\oplus B_2)(A_1\oplus B_1)(A_0\oplus B_0)C_0$

Sommatore con anticipo di riporto



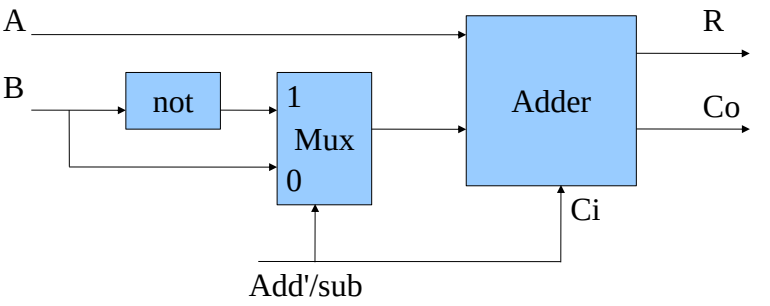
Sommatore con anticipo di riporto



Differenza

A - B = A + (-B) = A + C2(B)

C2(B) = NOT(B) + 1



Overflow

- Dati due numeri a *n* bit di segno diverso il risultato è sempre corretto e quindi occorre ignorare il riporto in uscita dello stadio più significativo
- Se i due numeri hanno lo stesso segno è possibile avere overflow
- Come capire se c'è stato overflow?
  - La somma di due numeri negativi è positiva oppure
  - La somma di due numeri positivi è negativa
- Overflow = $(a'_{n-1}b'_{n-1})s_{n-1} + (a_{n-1}b_{n-1})s'_{n-1}$

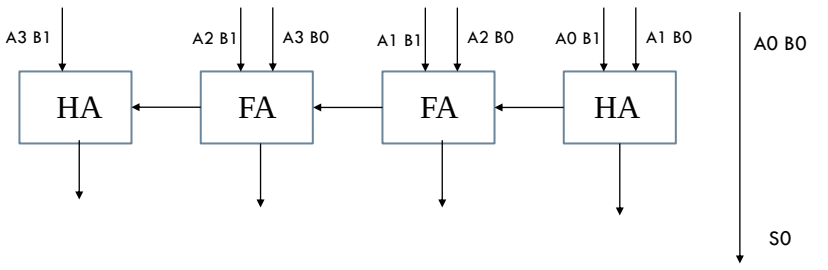
Prodotto

- Due fasi
- Calcolo dei prodotti parziali
  - Uguale per tutti i moltiplicatori
- Somma dei prodotti parziali
  - Somma per righe
  - Somma per diagonali

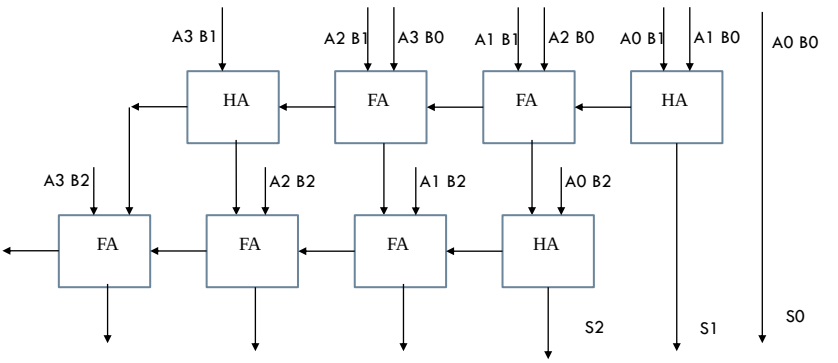
Prodotti parziali

				A3	A2	A1	A0
				B3	B2	B1	B0
				A2 B0	A2 B0	A1 B0	A0 B0
			A3 B1	A2 B1	A1 B1	A0 B1	
		A3 B2	A2 B2	A1 B2	A0 B2		
	A3 B3	A2 B3	A1 B3	A0 B3			
S7	S6	S5	S4	S3	S2	S1	S0

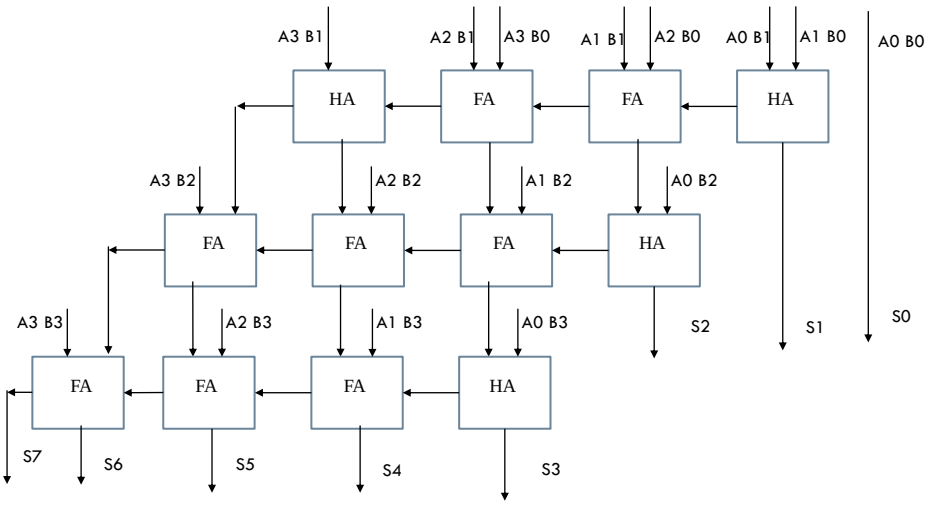
Somma per righe: prima riga



Somma per righe: seconda riga



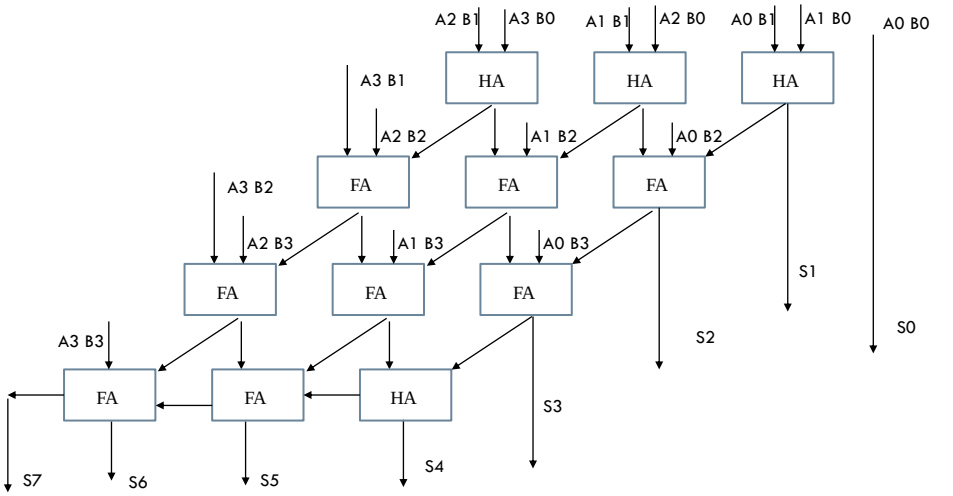
Somma per righe: terza riga



Ritardo somme per righe

- I tempi richiesti per la somma dei prodotti parziali dipendono *linearmente* dal numero dei bit dei fattori
- DelaySpR = 3*n* - 4

Somma per diagonale



Ritardo somme per diagonale

- I tempi richiesti per la somma dei prodotti parziali dipendono *linearmente* dal numero dei bit dei fattori
- DelaySpD = 2*n* - 2

Realizzazione di una ALU a n bit