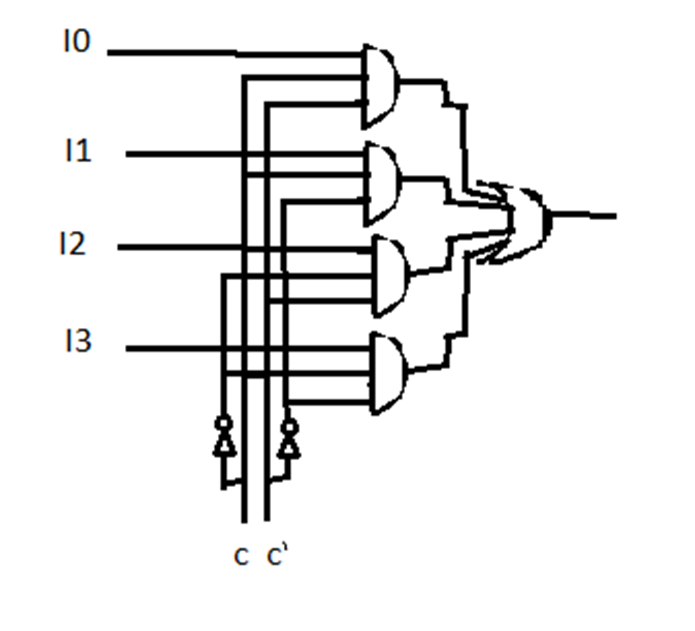
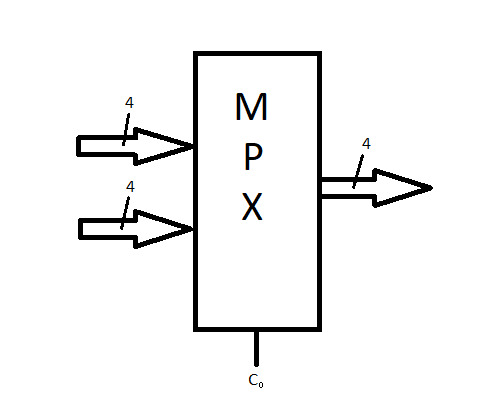
Un dispositivo multiplexer viene genericamente rappresentato come una scatola rettangolare, con vari ingressi, un uscita e un ingresso ausiliario di controllo.

Il valore di controllo deve essere rappresentato in tanti bit quanto il logaritmo in base 2 del numero di ingressi (approssimato per eccesso). Esempio di multiplexer.



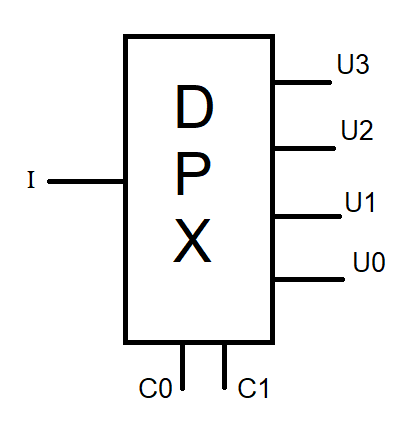
Un’alternativa all’estendere il multiplexer aumentando il numero di ingressi è quella di mantenere il numero di ingressi e aumentare il numero di bit che rappresentano ciascun ingresso. In tal caso bisogna moltiplicare il multiplexer di base (qui di seguito) per il numero di bit che rappresentano ciascuna variabile di ingresso (per esempio se ciascun ingresso è rappresentato su 3 bit bisogna triplicare la struttura rappresentata.

Se si vuole passare alla rappresentazione compatta (ignorando il funzionamento) si può usare:

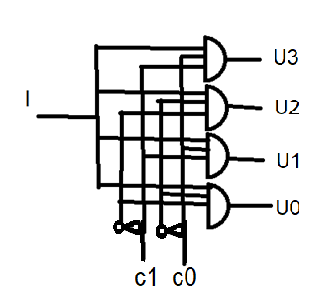


Dove MPX è la sigla che indica che è un multiplexer e i numeri sulle frecce indicano il numero di bit con cui sono rappresentati i valori di ingresso.

Si passa ora al Demultiplexer

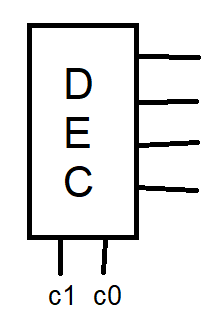


Esso rappresenta la funzione inversa del Multiplexer. In questo caso, può fare in modo di avere 3 uscite registrare sempre 0 e soltanto una il valore di I:

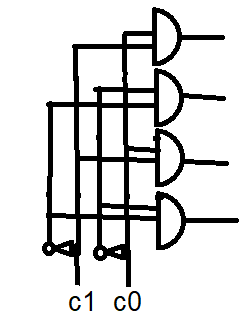


Il demultiplexer si differenzia dal multiplexer perché ha una funzione OR in meno e soltanto un ingresso. Anche il demultiplexer può essere espanso aumentando il numero di bit che codificano per le entrate e per le uscite (oppure aumentando il numero di uscite).

Introduciamo anche un terzo dispositivo: il circuito chiamato Decoder.



Esso si differenzia perché non ha ingressi, ha solo uscite. Esso invia in tutte le uscite tranne una il valore 0 e in quella rimanente il valore 1, basandosi sui valori di C0 e C1.

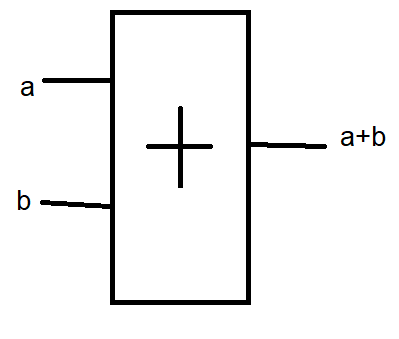


Un decoder è quindi una semplificazione del Demultiplexer (è come un demultiplexer il cui valore di ingresso è sempre 1).

Nota: Al crescere di ognuno di questi dispositivi esso consumerà più energia, si scalderà di più e occuperà più spazio.

Parliamo adesso di circuiti numerici: servono per fare dei calcoli su rappresentazioni binarie di numeri.

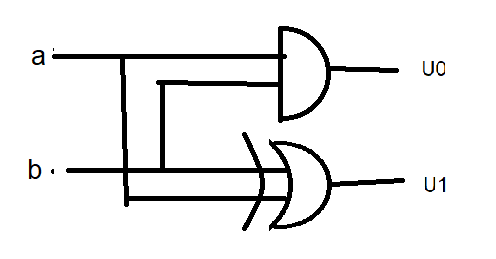
Pensiamo innanzitutto al circuito Sommatore:



Per evitare situazioni di errore bisogna controllare quali e quanti bit stiamo sommando, per non andare in overflow. Per esempio, se a e b sono rappresentati su 1 bit, avremo bisogno di due uscite (u0 e u1) per non andare in overflow. La tavola di verità di verità di QUESTA somma è:

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | u0 | u1 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

La colonna di u0 rappresenta la funzione AND, mentre la colonna di u1 rappresenta lo XOR. Quindi si può costruire una rappresentazione fisica che prevede l’uso di due funzioni.

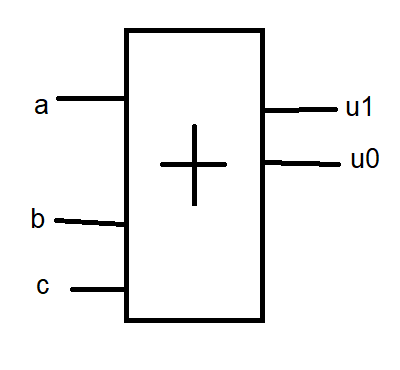


Se si aumentasse di 1 bit la rappresentazione di a e b (quindi 2bit e 2bit) ci servirebbero 3 bit in uscita (poiché il massimo risultato che si può ottenere è 6 < 8).

In quel caso la mappa di Karnaugh sarebbe:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a1a0\b1b0 | 00 | 01 | 11 | 10 |
| 00 | 000 | 001 | 011 | 010 |
| 01 | 001 | 010 | 100 | 011 |
| 11 | 011 | 100 | 110 | 101 |
| 10 | 010 | 011 | 101 | 100 |

In questo caso non è facile trovare una funzione elementare che permetta di effettuare la somma. L’alternativa sarebbe di aggiungere un terzo bit in ingresso anziché aumentare la rappresentazione binaria degli ingressi (e mantenere quindi la rappresentazione in ingresso a 1 bit e in uscita a 2 bit, che in questo caso sfrutta a pieno le sue capacità di rappresentazione).



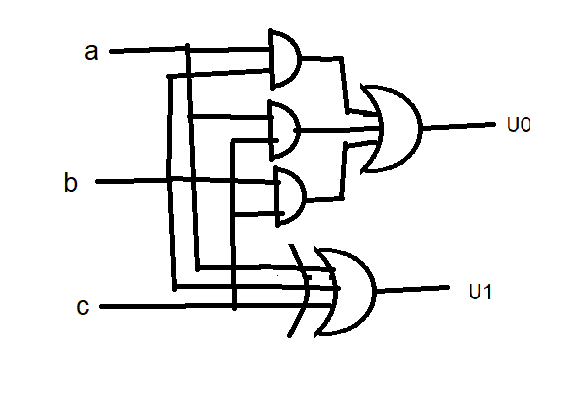
Mappa di Karnaugh:

|  |  |  |
| --- | --- | --- |
| ab\c | 0 | 1 |
| 00 | 00 | 01 |
| 01 | 01 | 10 |
| 11 | 10 | 11 |
| 10 | 01 | 10 |

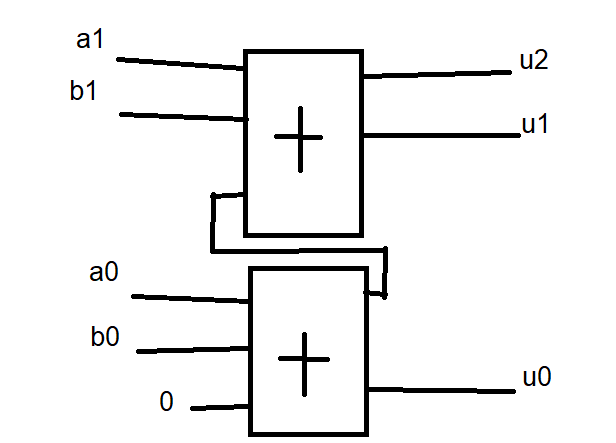
u1 = a\*b + b\*c + a\*c (ce la possiamo ancora cavare)

u0 però è un disastro (è uno zig-zag sulla tabella che sembra impedire ogni tipo di ottimizzazione), c’è tuttavia una funzione che ci permette di salvarci: lo XOR.

u0 = a XOR b XOR c;



Si può sfruttare ciò per ottenere risultati di somme anche a più bit di questi:



Si tratta il valore di uscita che normalmente sarebbe u1 come cifra di riporto e si effettua la somma tra gli altri due bit con in aggiunta il riporto. Questo tipo di circuito è definito Full Adder (o Sommatore Completo). Questo perché se vogliamo sommare numeri in 32 bit ci servono 32 Full Adder. Il vantaggio che offre questo genere di implementazione è che la complessità del circuito cresce LINEARMENTE all’aumentare dei bit in input. Lo svantaggio è che questi circuiti ci mettono una certa quantità di tempo (seppur normalmente piccola) per restituire il valore corretto in uscita. Quindi nel momento in cui si cambia la configurazione in ingresso (per un attimo) il valore in uscita non è corretto: c’è quindi un ritardo, che determina oltretutto la funzionalità massima del mio dispositivo (se il dispositivo impiega molto tempo per effettuare un’operazione risulta “lento” e poco utile). Dato ciò, questa configurazione fa si che all’aumentare del numero di bit cresce anche linearmente il ritardo (il sommatore a 64 bit è due volte più lento del sommatore a 32 bit, che è il doppio più lento di quello di 16 ecc.).

Dal puro punto di vista della velocità di funzionamento, questo sistema è tra i meno efficienti. Se vogliamo un circuito più veloce siamo costretti a crearne uno che consumi più energia, che occupi più spazio e che costi di più (sono quindi apprezzati i compromessi).