Abbiamo visto che per realizzare una funzione è possibile utilizzare la sua tavola di verità, tuttavia ciò diventa poco praticabile quando il numero di ingressi aumenta (per esempio la somma di due interi a 32 bit avrebbe una tavola di verità orribile). Per questo motivo si opta per la realizzazione di moduli ripetibili a cui spetta costituire la funzione: ad esempio nel caso dell’addizione abbiamo visto il full adder, che riceve tre bit e ne manda in uscita 2, uno di risultato e uno di riporto. In questo modo basta avere tanti moduli quante sono le cifre binarie dei due numeri che vogliamo sommare per poter costruire un circuito sommatore (anche se ciò implica utilizzare 32 moduli per la somma di due interi). Un approccio di questo genere soffre di un problema legato alla velocità del dispositivo: la quantità di tempo cresce linearmente all’aumentare dei moduli.

Per costruire dispositivi più veloci si può comunque cercare di usare la modularizzazione, ma bisogna trovare un metodo che non blocchi il circuito aspettando ogni riporto. Se si utilizzano dispositivi diversi per calcolare i riporti si può ridurre il tempo di esecuzione della somma. I dispositivi che permettono ciò si chiamano “anticipo del riporto” (o “Carry Lookahead”) -> Logica a tre livelli. Se noi andiamo a vedere i percorsi dagli ingressi alle uscite vediamo che possiamo avere degli ingressi negati e non negati, delle funzioni and e poi delle funzioni di tipo or (Quindi possiamo al massimo avere 3 funzioni logiche elementari). È importante sapere quante funzioni logiche elementari si trovano lungo il circuito perché all’aumentare del numero di funzioni aumenta il tempo che ci mette un informazione ad arrivare alla fine del circuito. (Chiamiamo Tau () il tempo di esecuzione di una funzione logica elementare).

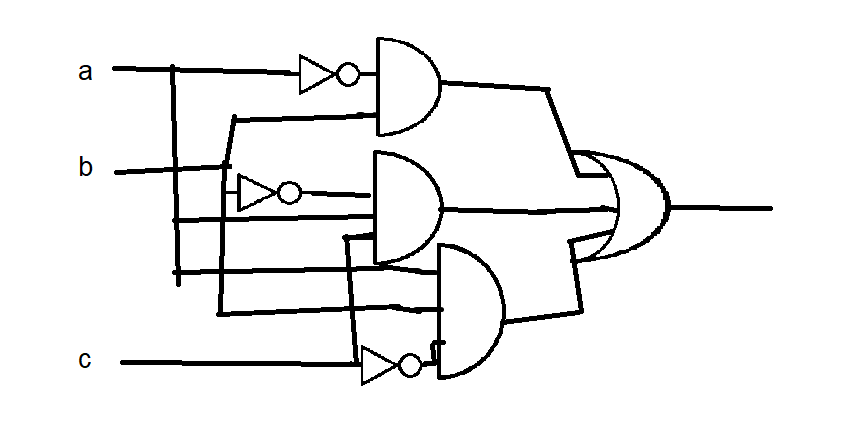
A determinare la crescita del tempo di propagazione dell’uscita, nel caso del circuito sommatore, è il fatto che il riporto continua a venir propagato per tutto il circuito e a passare attraverso ben più di 3 funzioni logiche (anche 6, 9, 12 ecc.). Se si utilizzano gli anticipi di riporto, ciò permette di tenere fisso a 3 il numero massimo di funzioni logiche che il circuito incontra tra ingresso e uscita (e quindi si ha la Logica a 3 Livelli).

Tramite tavole di verità e le funzioni AND, OR, NOT e applicando la modularità è possibile rappresentare qualsiasi funzione.

Esempio:

|  |  |  |  |
| --- | --- | --- | --- |
| a | b | c | U |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

U = -a\*b+a\*-b\*c+a\*b\*-c;

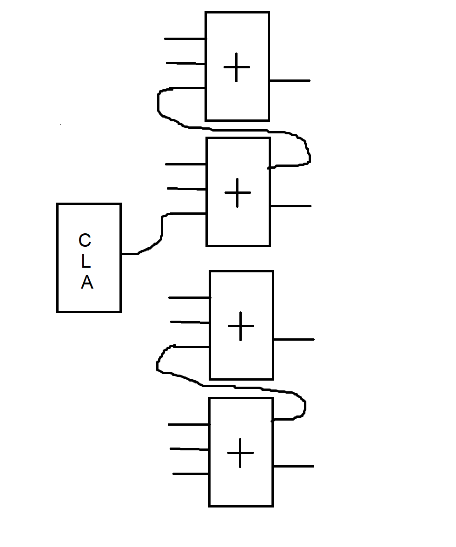


Quando si produce un circuito seguendo indicazioni di questo tipo bisogna però tenere conto dei tempi che nella realtà le funzioni impiegano a modificare il valore del segnale.

Nel circuito rappresentato, poiché il numero massimo di funzioni che il circuito incontra tra ingresso e uscita, il tempo da aspettare perché il valore in uscita sia giusto (cambiati i valori in ingresso) è di 3 Tau. Il dispositivo è realizzato in Logica a tre livelli.

Il Full Adder, al suo interno, è un dispositivo a logica a tre livelli. Il ritardo minimo che si può avere, dunque, è 3 Tau.

Utilizzando il Carry Lookahead è un dispositivo a sua volta in logica a tre livelli che permette di ri-azzerare la catena dei riporti (inserendo un Carry Lookahead prima di ogni modulo sommatore si potrebbe tenere costante il tempo di ritardo dell’intero circuito al numero minimo possibile di unità di ritardo). Per risparmiare (in fatto di spazio, calore, energia e costi di produzione), non si inserisce un Anticipo del riporto con elevata frequenza, ma ne basta uno ogni tanto per tenere bassi i tempi di ritardo del circuito (poiché ogni Anticipo del riporto, appunto, ri-azzera la catena dei riporti).



Inoltre, man mano che aumenta il numero di cifre, diventa anche più grande (e costoso) il Carry Lookahead poiché necessita di sempre più ingressi (quindi è meglio dosarne la quantità). Nella realtà si cercano le vie di mezzo.

Ipotizziamo di dover costruire un CL a 4 ingressi:

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

U = a\*-b\*-c+-b\*-c\*-d+-a\*b\*-c\*d+-a\*b\*c\*-d.

A questo punto si può realizzare fisicamente con 2 AND a 3 ingressi, 2 AND a 4 ingressi, un OR a 4 ingressi e 4 NOT (-> Ci troviamo in una situazione di logica a tre livelli).

Questa situazione è applicabile a qualsiasi funzione.

Sarebbe bello poter costruire un unico dispositivo che si possa comportare come ciascuna di queste 4 funzioni: Questo non è direttamente possibile, ma è possibile simulare una situazione di questo genere applicando le trasformazioni di De Morgan. La serie di funzioni AND tutte convergenti in un OR possono essere tutte sostituite da NAND (compreso l’OR). Inoltre, il NOT può essere visto come un NAND a un ingresso solo, quindi ipoteticamente è tutto realizzabile con un solo dispositivo generale.

Nota a caso: aggiungere un ingresso a una funzione costa solo un transistor in più.

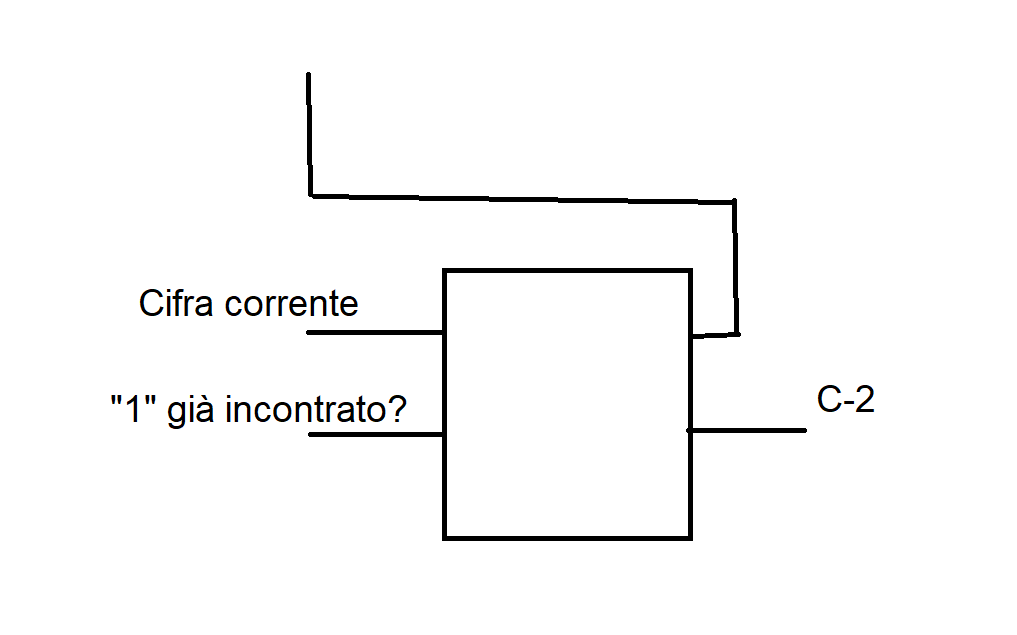
Come metodo alternativo per la realizzazione di funzioni è quello di trovare la forma normale disgiuntiva anziché quella congiuntiva (cercando gli zeri e scrivendola algebricamente come prodotto di somme anziché somma di prodotti). In questo caso se si cercasse un dispositivo universale si ricadrebbe nel NOR anziché nel NAND.

Consideriamo adesso degli altri esempi di dispositivi. Un’altra semplice operazione che ci può venire in mente è quella del cambiamento di segno. Il comportamento di tale operazione dipende però dal tipo di codice il dispositivo utilizza (modulo e segno/ complemento a 1/ complemento a 2/ eccesso 2^n). Nel caso di modulo e segno basta invertire il bit di segno -> è sufficiente una funzione NOT che prende in entrata il bit di segno e ne restituisce il valore contrario. Nel caso del complemento a 1, bisogna invertire tutti i bit, quindi servono tante funzioni NOT quanti sono i bit della rappresentazione che hanno in entrata e in uscita un bit ciascuna.

Per il complemento a 2 (che è tra le più frequentemente usate) si invertono i bit e poi si usa un circuito sommatore per sommare ai valori ribaltati 00…01. Ciò è un po’ uno spreco (perché si usa un circuito sommatore per aggiungere la costante 1). Si può quindi usare un Half adder anziché un Full adder (ha 2 ingressi anziché 3 e ha anch’esso due uscite, uno per la cifra significativa e l’altra per il riporto); di conseguenza per sommare la costante uno bastano tanti Semi-addizionatori quante le cifre binarie del numero (che costruiscono una specie di versione ottimizzata del circuito sommatore per la costante 1) e il valore 1 in ingresso alla cifra meno significativa.

Un modo alternativo di procedere è quello di usare una proprietà diversa per il calcolo del complemento a due (che permette di saltare il complemento a 1): si osservano le cifre binarie una alla volta da destra verso sinistra e si ricopiano tutte le cifre finché non si trova un 1, a quel punto si ricopia anche l’1 e si ricopiano ribaltate tutte le cifre successive ad esso.

Si può quindi creare un dispositivo che prende in ingresso due valori, uno che contiene la cifra corrente che si sta esaminando e l’altro che contiene 1 o 0 a seconda che si sia già trovato un 1 o no tra le cifre analizzate (l’inizializzazione del secondo ingresso è quindi ovviamente 0).



|  |  |  |  |
| --- | --- | --- | --- |
| a | b | c | d |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |

A = Cifra corrente;

B = “1” già incontrato;

C = C-2;

D = ;

Le funzioni C e D sono quindi uno XOR e un OR. L’OR è già una funzione elementare, per realizzare lo XOR in Logica a 3 livelli si può utilizzare l’espressione algebrica XOR = a\*-b+-a\*b.

Si può quindi costruire una struttura simile a quella dei semisommatori (che quindi occuperà una quantità linearmente crescente di tempo al crescere di cifre binarie). È quindi solo una via alternativa, che però permette di evitare di dover fare sia il ribaltamento di cifre che la somma. Provare a realizzare quest’operazione in logica a 3 livelli è possibile per quantità di bit relativamente piccole, tuttavia diventerebbe esponenzialmente più complicato all’aumentare delle cifre e quindi anche in questo caso è necessario cercare il compromesso.

