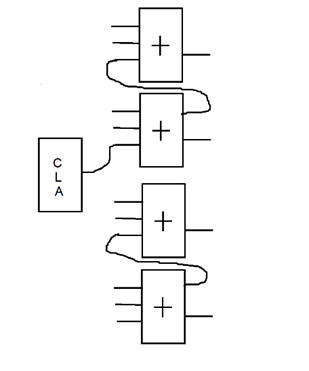
Abbiamo visto che per evitare di produrre dispositivi troppo lenti per le esigenze pratiche si può inserire un Carry LookAhead ogni tot. Moduli.

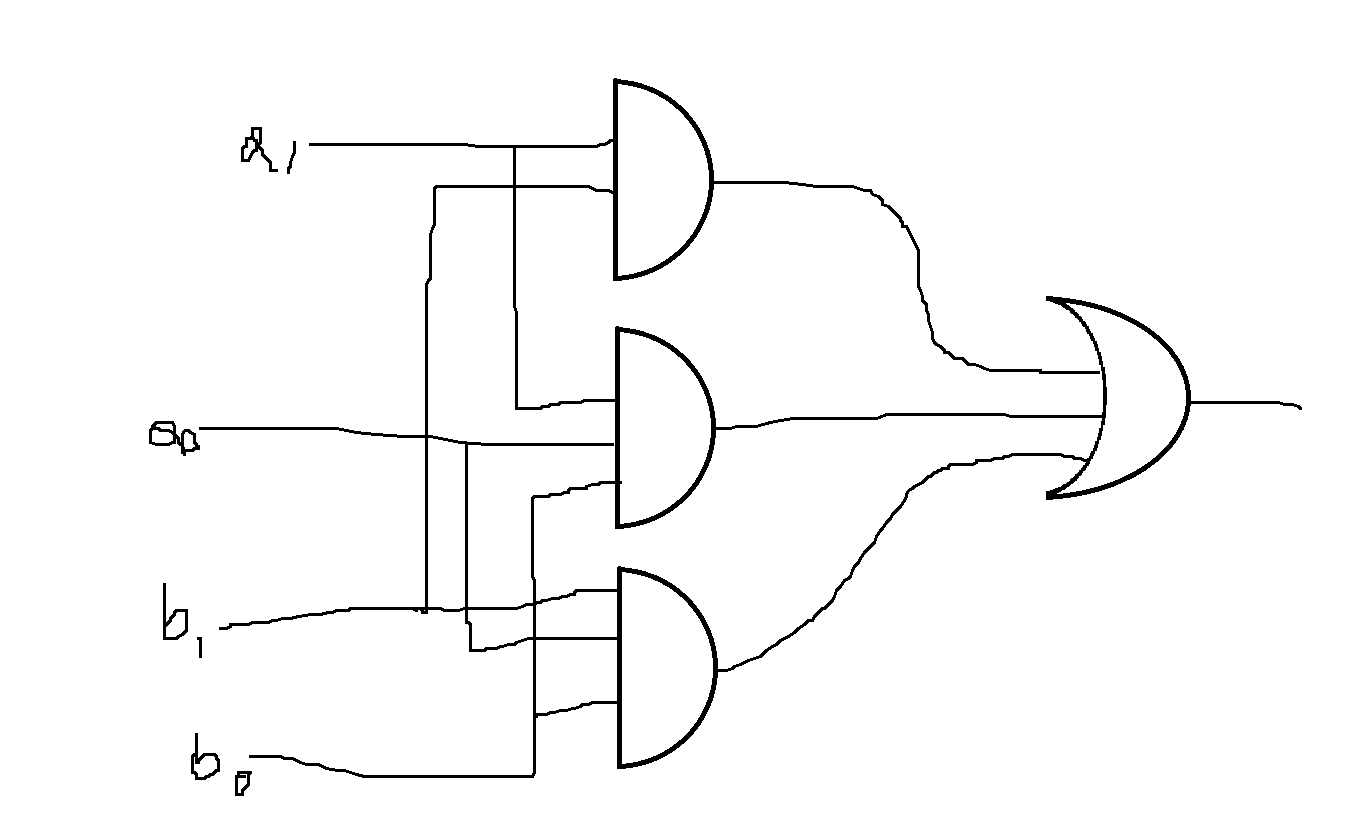


Nel caso di Carry LookAhead inserito dopo i primi due bit, la mappa di Karnaugh risulta:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a1a0\b1b0 | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 0 | 1 | 0 |
| 11 | 0 | 1 | 1 | 1 |
| 10 | 0 | 0 | 1 | 1 |

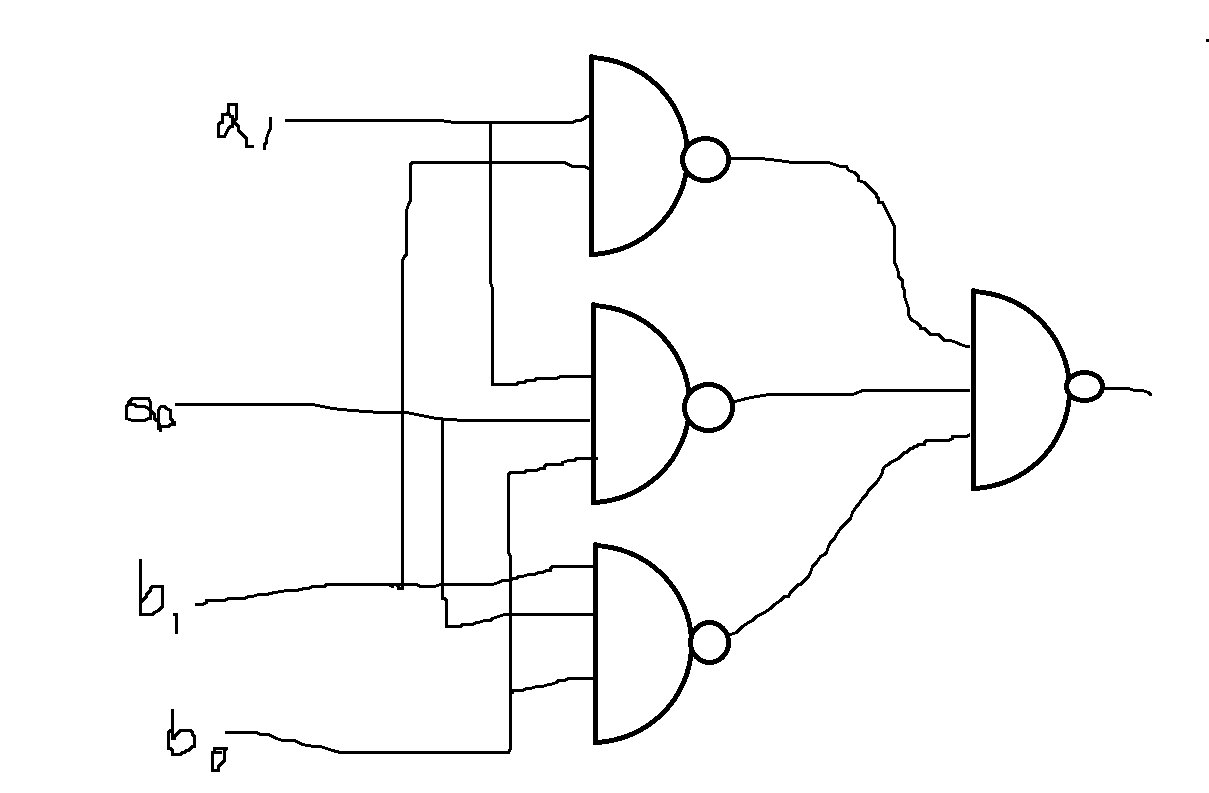
Questa è la funzione che vogliamo realizzare per generare in anticipo il riporto da utilizzare sulla terza cifra.

u = a1\*b1 + b1\*b0\*a0 + a1\*a0\*b0.



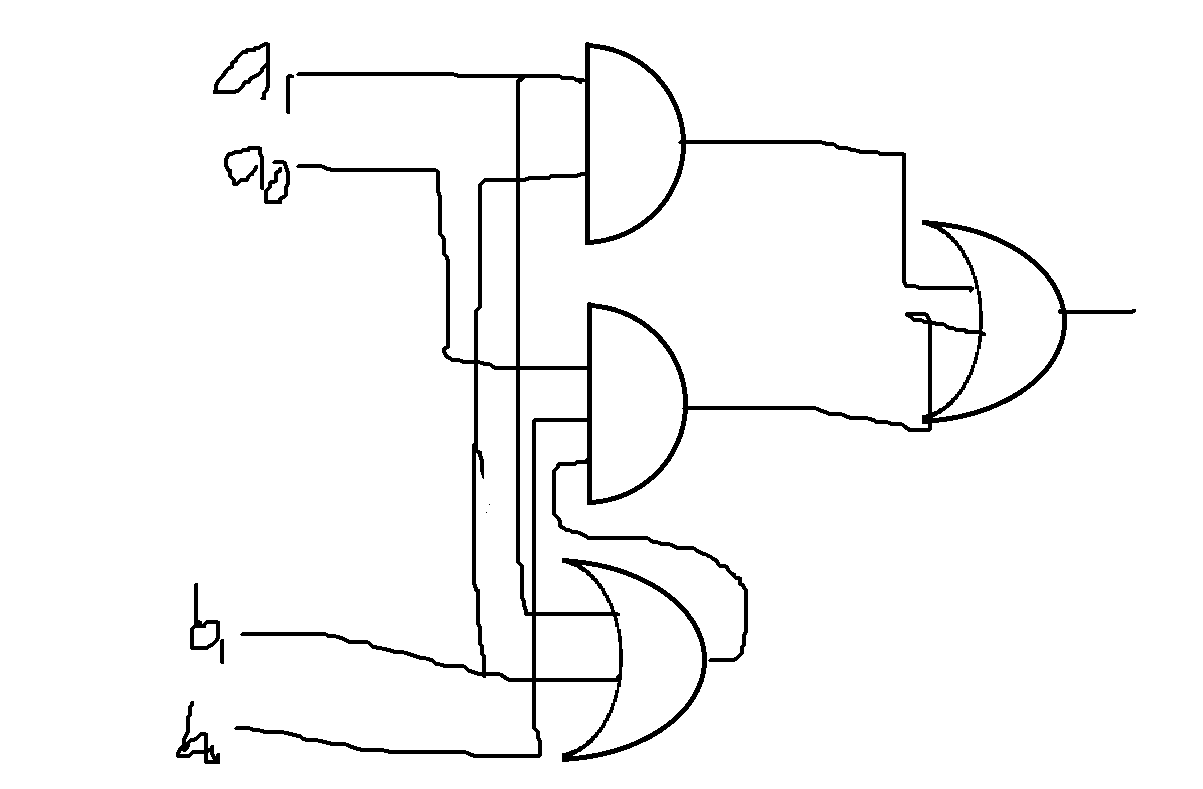
Per realizzarla servono 3 funzioni AND (1 a due ingressi e 2 a tre ingressi) e una funzione OR (a tre ingressi). In questo caso ci troviamo in un caso di logica a 2 livelli (quindi è più “facile” che a 3 da realizzare), poiché da d’ovunque si parta il circuito incontra al massimo 2 funzioni logiche. Ciò ci fa comodo perché ci produce con un minor ritardo il riporto da utilizzare nel full adder.

Trasformando il circuito in forma NAND, diventa perfino meno complicato: questo sia perché si utilizza un solo tipo di componente, sia perché il NAND è più facile da realizzare dell’AND.



C’è da dire però che funzioni con tanti ingressi consumano leggermente di più del normale. Sarebbe quindi conveniente, trovata una formula in forma normale, cercare di ridurre il numero di variabili in ingresso. Nel nostro esempio: u = a1\*b1 + b1\*b0\*a0 + a1\*a0\*b0 si può provare a raccogliere un a0\*b0:

u = a1\*b1 + (a0\*b0)\*(a1+b1). Ciò mi permette di realizzare questo tipo di dispositivo:



In questo caso abbiamo lo stesso numero di funzioni (sempre 4, ma 2 OR e 2 AND), tuttavia hanno in generale meno ingressi (soltanto una ha 3 ingressi, mentre prima erano 3). Questo vantaggio porta però lo svantaggio di non essere in logica a 2 livelli ma in logica a 3 livelli.

Il numero di ingressi di ogni funzione è detto FAN-IN (qui tre funzioni hanno FAN-IN di 2 e una ha FAN-IN di 3). Funzioni con FAN-IN più basso sono più facili da realizzare di altre dello stesso tipo ma a FAN-IN più alto.

Utilizzando un modulo generico di tipo **multiplexer** è possibile ottenere una forma generica di funzione a 4 ingressi. Si utilizza un multiplexer a 16 ingressi e a 4 variabili di controllo (i cui valori corrispondono a quelli dei 4 ingressi a0 a1 b0 b1). Si inseriscono invece nei 16 ingressi del multiplexer i valori (costanti) che si leggono all’interno della mappa di Karnaugh. Per esempio, nella casella corrispondente al numero 5 (ossia a1a0b1b0 = 0101) è contenuto il valore 0, perciò all’ingresso 5 è sempre in ingresso la costante 0. Se si prende la casella 1111 (=15) il valore contenuto è 1, quindi si inserisce nell’ingresso 15 la costante 1.

Questo dispositivo permette di realizzare qualsiasi funzione a 4 ingressi e a 1 uscita. Se si vuole modificare la funzione basta cambiare i valori costanti dei 16 ingressi: Ciò permette la realizzazione di dispositivi programmabili (basta inserire degli interruttori al posto di valori costanti nei 16 ingressi e, a seconda di come si girano questi interruttori, il dispositivo può rappresentare qualsiasi funzione).

I vantaggi e gli svantaggi di usare un dispositivo del genere per un Carry LookAhead sono i seguenti:

* È sufficiente comprare un solo tipo di dispositivo e riprogrammarlo a seconda della situazione per la quale lo si vuole usare (=vantaggio).
* È più lento del Carry LookAhead (poiché è in Logica a 3 livelli, mentre il CLA standard è in logica a 2 livelli = svantaggio). Se però la funzione che volessimo rappresentare ammettesse solo soluzioni a 3 livelli allora non ci rimetteremmo niente.

Da notare che se si cambiano i fili di controllo (a1b1a0b0 per esempio) vi è anche una permutazione negli ingressi (in quale ingresso vada inserito quale valore). C’è inoltre da dire che gli ingressi del multiplexer aumentano esponenzialmente all’aumentare delle variabili di controllo (se avessi una tastiera da 256 tasti sarebbe infattibile manualmente). È quindi necessario trovare un modo di generare la configurazione di un multiplexer in automatico (tramite programmazione -> BDD). Si possono usare dei dispositivi circuitali programmabili (che si differenziano in quelli che si possono programmare una volta sola e quelli che si possono riprogrammare più volte, anche se in questo caso il costo dei dispositivi aumenta -> vanno bene per dei prototipi ma non sono utilizzabili per produzione su larga scala).

Un esempio interessante di circuito numerico è il dispositivo comparatore. Un comparatore ha due ingressi a, b ed effettua diverse verifiche a seconda di che tipo di comparazione si vuole ottenere. Nel caso dell’eguaglianza possiamo far in modo che restituisca 1 in uscita se a=b e altrimenti 0.

Mappa di Karnaugh:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a1a0\b1b0 | 00 | 01 | 11 | 10 |
| 00 | 1 | 0 | 0 | 0 |
| 01 | 0 | 1 | 0 | 0 |
| 11 | 0 | 0 | 1 | 0 |
| 10 | 0 | 0 | 0 | 1 |

Qui di caselle adiacenti non ce n’è neanche una. La forma minimale, quindi, sembrerebbe essere:

u = -a1\*-a0\*-b1\*-b0 + a1\*-a0\*b1\*-b0 + a1\*a0\*b1\*b0 + -a1\*a0\*-b1\*b0.

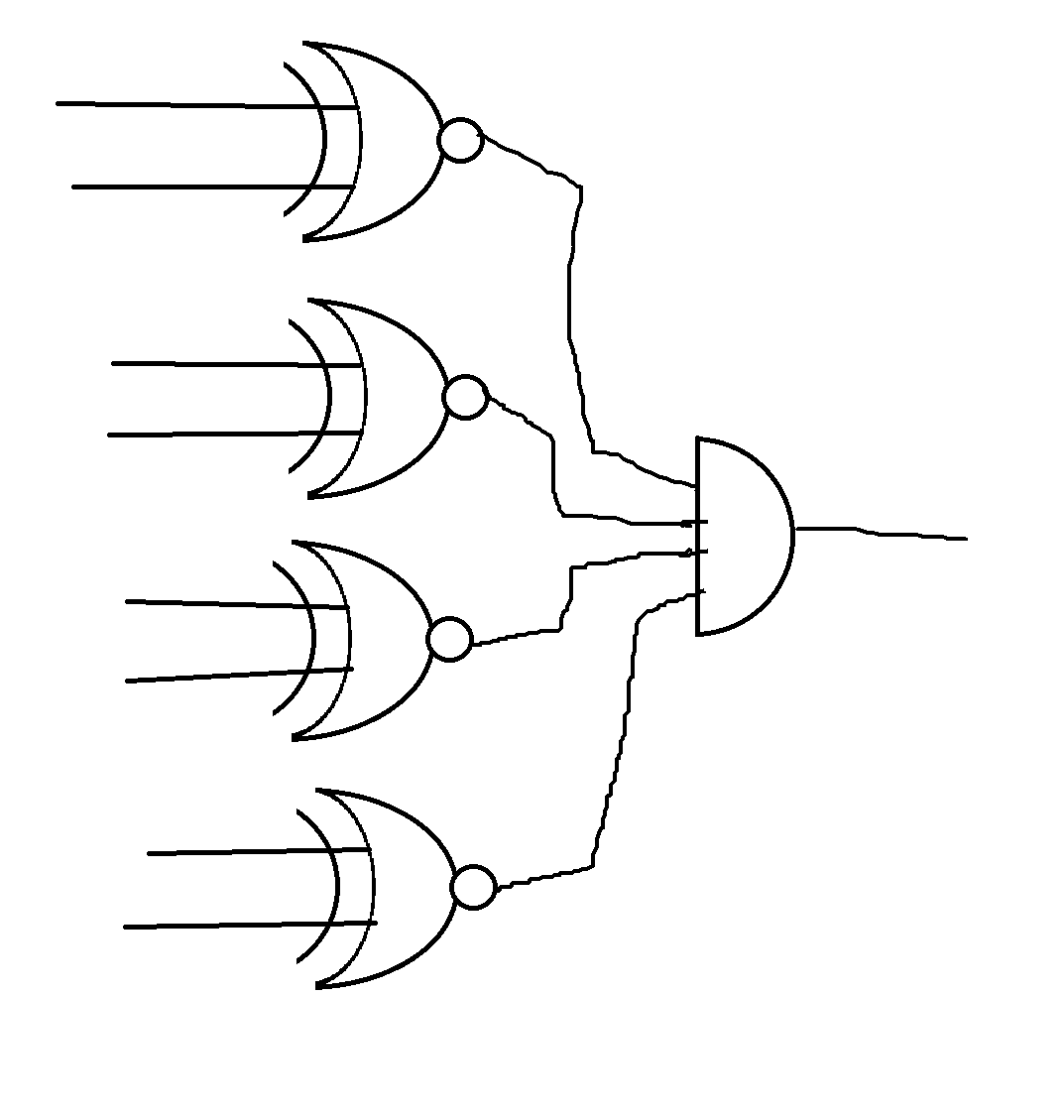
In questo caso abbiamo una situazione in logica a 3 livelli, con 4 AND a 4 ingressi e 1 OR a 4 ingressi.

Se aumentassimo il numero di bit gli 1 resterebbero comunque sempre sulla diagonale. Quindi saranno sempre necessarie tante funzioni AND quanto 2^ al numero di bit di a e b (se a e b sono rappresentati su 3 bit servono 8 AND e 1 OR a 8 ingressi). La complessità del dispositivo, quindi, cresce esponenzialmente all’aumentare del numero di bit, e tale crescita impatta sul numero di funzioni AND, incrementando notevolmente il costo del circuito.

Poiché la soluzione non è scalabile siamo costretti a cercare una soluzione di tipo modulare. Trovarla è abbastanza semplice: considero la rappresentazione su una sola cifra binaria (a0b0), la mappa è:

|  |  |  |
| --- | --- | --- |
| a0\b0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

In questa mappa di Karnaugh si può riconoscere come funzione la negazione dello XOR. Per estenderla a più bit basta confrontare i singoli bit tra loro e poi convogliare i risultati in una funzione AND.



Il vantaggio in questo caso è che si ha una crescita LINEARE di utilizzo di funzioni XNOR all’aumentare del numero di bit.

Qual è il ritardo però? Se si considera lo XNOR come una funzione elementare la funzione sarebbe una funzione in logica a 2 livelli, però nel dettaglio non si può sapere (potrebbe essere maggiore dell’AND). Però il lato positivo è che qualunque esso sia, tale ritardo rimane Costante all’aumentare del numero di bit.

Questo è un esempio in cui la rappresentazione modulare è su tutti i fronti più vantaggiosa della rappresentazione a 3 livelli basata su tavole di verità.

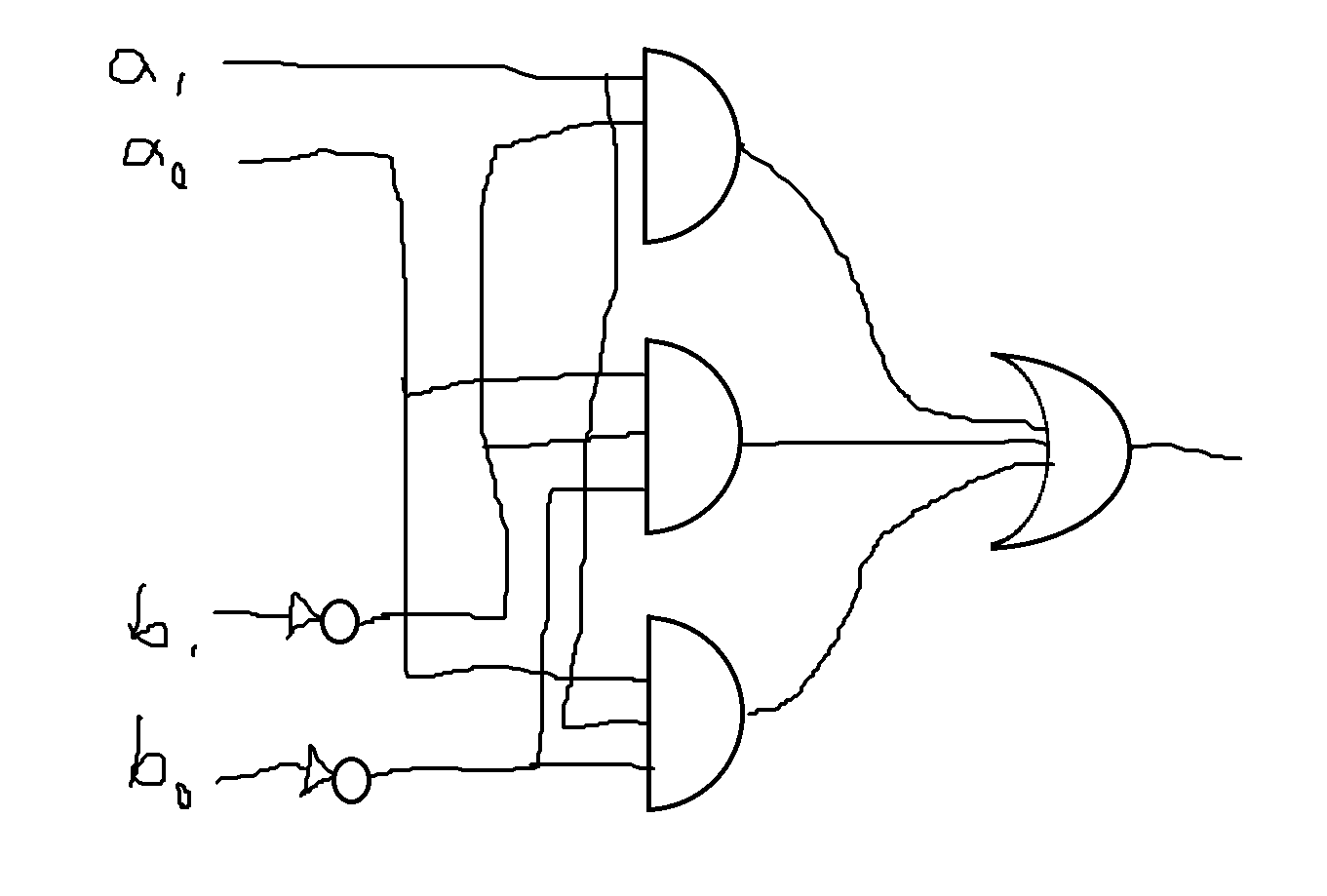
Ci si potrebbe chiedere come si comporti invece il comparatore >. Si assuma che se a>b la funzione restituisca 1 e se a <= b essa restituisca 0. La mappa di Karnaugh è:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a1a0\b1b0 | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 1 | 0 | 0 | 0 |
| 11 | 1 | 1 | 0 | 1 |
| 10 | 1 | 1 | 0 | 0 |

(In questo caso è importante sapere con quale codifica sono rappresentati i due numeri (a e b). Potrebbero essere rappresentati con segno, senza segno, ecc. Stiamo qui assumendo il caso più semplice: che siano entrambi interi senza segno).

La rappresentazione minimale che possiamo trovare è:

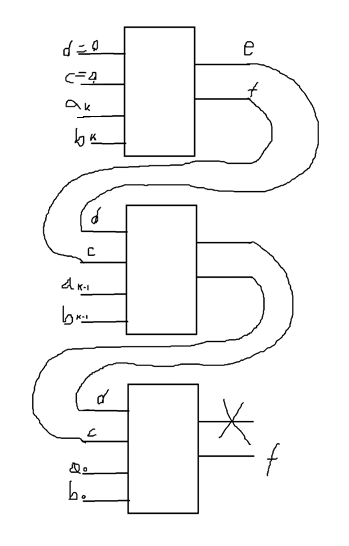
u = a1\*-b1 + a0\*-b1\*-b0 + a1\*a0\*-b0



In questo caso realizzare il comparatore usando la logica a 3 livelli tramite mappe è più semplice che rappresentare l’uguaglianza, ma ha comunque crescita ad andamento esponenziale. Quindi, bisogna anche in questo caso cercare una struttura di tipo modulare.

Quando si confrontano due numeri cercando il più grande tra i due, la cifra più importante (e da cui conviene quindi partire) è l’ultima (quella più a sinistra) poiché essa da sola rappresenta un numero di 1 maggiore alla somma di tutte le altre cifre. Quindi se le due cifre più significative sono diverse tra di loro si può già dare una risposta, se invece sono uguali bisogna procedere a guardare i valori delle cifre successive. Questa analisi preliminare ci permette di trovare un algoritmo che partendo dalle cifre più significative fa un confronto cifra per cifra tra i due numeri fintanto che non trova due valori diversi: a quel punto può dare una risposta.

Quindi il mio modulo ha bisogno di prendere in considerazione 4 ingressi e avere almeno 2 uscite (poiché deve poter codificare 3 possibilità in uscita: a>b, a<b, a?b). Per esempio, u=00/10 -> risultato non ancora trovato, u= 11 risultato trovato ed è a >b, u=01 risultato trovato ed è a <= b.



In questo caso la mappa di Karnaugh sarebbe:

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

Da notare che in alcune caselle al posto di 0,0 ci potrebbe essere 0,1 ( o in generale 0, ) tuttavia se si arriva all’ultimo modulo con quel valore il circuito si comporterà come se avesse emesso il valore 00.