CAPITOLO 6

LOGICA COMBINATORIA

6.1 Procedimento di progetto di circuiti logici

Il progetto di un circuito logico combinatorio parte da un enunciato verbale che descrive il problema e finisce con uno schema di circuito logico, o con un insieme di funzioni Booleane da cui può essere facilmente ottenuto lo schema logico. Il procedimento si svolge in sei passi:

1) *Viene stabilito il problema e definito il suo enunciato.*
2) *Vengono individuate il numero delle variabili sia di ingresso sia di uscita*.
3) *A ciascuna variabile di ingresso e di uscita viene assegnata un nome*.
4) *Viene derivata la tabella della verità che definisce le relazioni richieste dal problema tra le variabili di ingresso e ciascuna delle variabili di uscita*.
5) *Per ogni uscita si trova la funzione Booleana in forma semplificata*.
6) *Viene disegnato lo schema logico.*

Ricordiamo che una tabella di verità per un circuito combinatorio, con *n* ingressi ed *m* uscite, consiste di *n* + *m* colonne pari al numero delle variabili di ingresso più quelle di uscita. Gli 1 e gli 0 nelle colonne delle variabili di ingresso sono ottenute dalle 2^n combinazioni possibili delle *n* variabili. I valori binari delle variabili di uscita sono determinati dall'enunciato verbale del problema. Una uscita deve avere il valore 0 o 1 per ogni valida combinazione delle variabili di ingresso. Se dall'enunciato verbale del problema è chiaro che certe combinazioni non si possono verificare allora quelle sono combinazioni di don't care e quindi vanno segnate con una X.

Poiché le funzioni di uscita, determinate dalla tabella di verità, determinano univocamente il circuito combinatorio è importante che l'enunciato del problema sia interpretato correttamente. Una cattiva interpretazione dell'enunciato produce senz'altro un circuito combinatorio che non realizza la soluzione richiesta.

La semplificazione delle funzioni Booleane delle uscite deve senz'altro tenere conto dei metodi generali esposti nel Cap. 4 ma, poiché adesso abbiamo più funzioni Booleane che dipendono dalle stesse variabili di ingresso, possono essere effettuate delle semplificazioni globali. I criteri guida che bisogna tenere presente per una minimizzazione globale sono i seguenti:

- *minimo numero di porte,*
- *minimo numero di ingressi ad una porta,*
- *minimo tempo di propagazione del segnale attraverso il circuito,*
- *minimo numero di inter connessioni* e
- *verifica delle capacità di fan-out di ciascuna porta*.

Siccome tutti questi criteri non possono essere soddisfatti contemporaneamente, e siccome il criterio di minimizzazione più importante da applicare dipende dalla applicazione stessa, è difficile fornire delle regole generali. Nella maggior parte dei casi la semplificazione ha come obbiettivo primario la realizzazione delle funzioni richieste. Successivamente, tenendo conto dei parametri del problema che sono ritenuti più importanti (velocità, numero di porte, etc.), si procede, per tentativi, ad ottimizzare le semplificazioni.

6.2 Encoder

Gli elementi discreti di informazione sono rappresentati nei sistemi digitali da numeri binari o codici binari. Sul problema della codificazione e sulla standardizzazione dei codici, abbiamo parlato nel cap. 1. Sin dal quel momento sarebbe dovuto risultare chiaro che ad ogni modalità in cui si presenta una informazione è associato uno ed un solo codice. Inoltre, se si verifica una modalità, tutte le altre non si possono verificare: cioè esse sono *mutuamente esclusive*.
Un circuito logico *encoder* (codificatore) è un sistema che, accetta un numero *m* di stati logici, mutuamente esclusivi, uno per ogni sua linea di ingresso, e fornisce, su *n* linee di uscita, l'informazione codificata. Sappiamo che se le linee di uscita sono *n* il numero massimo di linee di ingresso sarà $m \leq 2^n$ e che la codificazione, se non è fissata dal problema, è completamente arbitraria. Per meglio capire che le modalità, in cui si presenta una osservazione, sono mutuamente esclusive e cosa questo comporta, in pratica, facciamo un esempio.
Si supponga di avere un encoder per i tasti di una macchina da scrivere, ad ogni tasto corrisponderà un unico codice (per es. lo standard ASCII). Se uno vuole scrivere un dato carattere batterà il tasto corrispondente e solo quello. Può accadere però che, involontariamente, l'operatore batte contemporaneamente due tasti vicini. Tutti sanno che in questo caso la macchina da scrivere, non potendo scrivere due caratteri nello stesso posto si rifiuta di azionare il meccanismo di stampa (rivela e corregge l'errore, non scrivendo alcun carattere). In un sistema di codifica deve esserci lo stesso tipo di protezione che blocca il trasferimento della richiesta di codifica. Noi tralasciamo, per il momento, di dire come questa protezione possa essere realizzata e supponiamo che sia funzionante. In definitiva possiamo dire che istante per istante arriva una ed una sola richiesta di codifica.

Es. 1 *Progettare un encoder di una informazione che si presenta in otto modi diversi e che fornisce un'uscita codificata in binario puro*.
Essendo il numero delle linee di ingresso = 8 il numero delle linee di uscita sarà uguale a 3 ($2^3 = 8$). Poiché è utile distinguere le variabili di ingresso e di uscita dobbiamo dare loro un

nome che li individua singolarmente: a quelle di ingresso diamo il nome f_j (con j = 0, 7) in cui l'indice j, usato per la distinzione delle 8 modalità, ne fissa l'ordine, mentre a quelle di uscita diamo il nome x, y , z.

Secondo le regole riportate in par. 6.1 dovremmo ora fare una tabella di verità che contiene tutte le combinazione delle variabili di ingresso che sono 2^8 = 256 ed assegnare per ogni una di queste combinazioni i valori alle variabili di uscita cioè il codice. Il problema, come si vede, si presenta arduo se non si tiene in conto che ci sono 256 - 8 = 248 combinazioni don't care. Queste sono tutte le combinazioni per cui più di un ingresso ha il valore 1 che, per la loro mutua esclusione, non possono certamente presentarsi. Ricordiamo che l'arbitrarietà con cui i codici possono essere definiti proviene da questo gran numero di condizioni di don't care. Dall'enunciato del nostro problema siamo vincolati ad una codifica binaria. In definitiva la nostra tabella avrà solo 8 entrate e 3 uscite. Scegliamo di codificare queste, rispettando l'ordine indicato dall'indice, secondo il codice di numerazione binaria pura (vedi Tab. 6.1) .

f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Tab. 6.1

Per la semplificazione delle funzioni di uscita x, y, e z, non è il caso di procedere attraverso il metodo tabulare (le variabili sono più di 4) anche perché la soluzione è immediata. Basta sommare logicamente tutte gli ingressi che rendono 1 la funzione di uscita da determinare. Per cui abbiamo: $x = f_4 + f_5 + f_6 + f_7$; $y = f_2 + f_3 + f_6 + f_7$; $z = f_1 + f_3 + f_5 + f_7$.

La realizzazione di un encoder in realtà ha bisogno di una quarta linea di uscita. Infatti, al numero delle combinazioni di ingresso da codificare bisogna aggiungerne un altra che qualifica la codifica. Essa indicherà con un 1 logico se almeno una delle linee di ingresso è attiva.

La Fig. 6.1 mostra la realizzazione di un encoder con 8 linee di ingresso e 4 bit in uscita. Come si vede essenzialmente un encoder è un circuito realizzato con porte OR con $m \leq 2^n$ linee di ingresso ed n uscite. Dal timing di Fig. 6.2 abbiamo che gli ingressi all'encoder (S0-S7) sono mutuamente esclusivi: ad ogni istante, soltanto una delle linee è attiva (all'uno logico). La quarta linea di uscita è stata ricavata facendo l'OR di tutti gli ingressi (U21A, U21B e U17D). L'uscita di quest'ultima è stata chiamata "Valid" in quanto con il suo

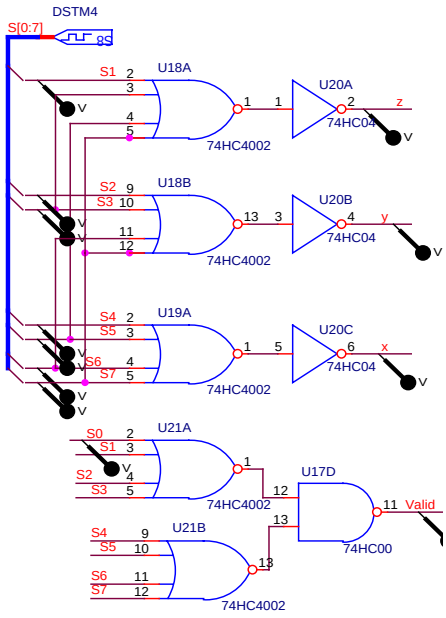


Fig 6.1

stato alto indica che l'indirizzo del numero di linea attiva, indicato nei tre bit ordinati x, y, z (z essendo il bit meno significativo) è valido.

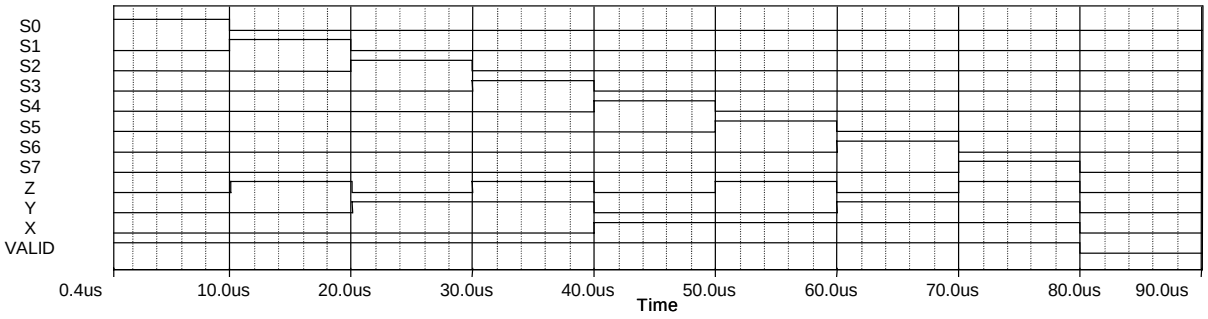


Fig. 6.2

Un codice binario con *n* bit e condizione di don't care rappresenta, per definizione, un numero inferiore a 2^n elementi di informazione. Il suo encoder userà un numero di ingressi inferiore a 2^n. Come esempio si consideri il codice BCD che è a 4 bit ma, per la presenza di condizioni di don't care nella sua definizione, rappresenta solo 10 elementi di informazione. Nel caso **non** vi vuole utilizzare una linea in più per validare la codifica si potrebbe utilizzare una delle combinazioni non usate per indicare che nessuna linea è attiva.

6.3 Rivelatore di Priorità e Codificatore

Un rivelatore di priorità è un circuito che accetta *m* linee di ingresso e fornisce l'indirizzo della linea attiva, con maggiore priorità, codificata in binario. Nello stesso tempo ci possono essere attive, tutte, in parte o nessuna delle linee di ingresso. Il circuito, in base ad un ordine prestabilito, e ad un prefissato criterio di importanza, determina quale delle linee è prioritaria e fornisce la codifica (d'ordine) della linea scelta. Questo problema si presenta nella gestione, da parte di un computer, delle periferiche di sistema. Queste, quando hanno bisogno di intervento, generano un segnale, chiamato "*Richiesta di Interruzione*". Per

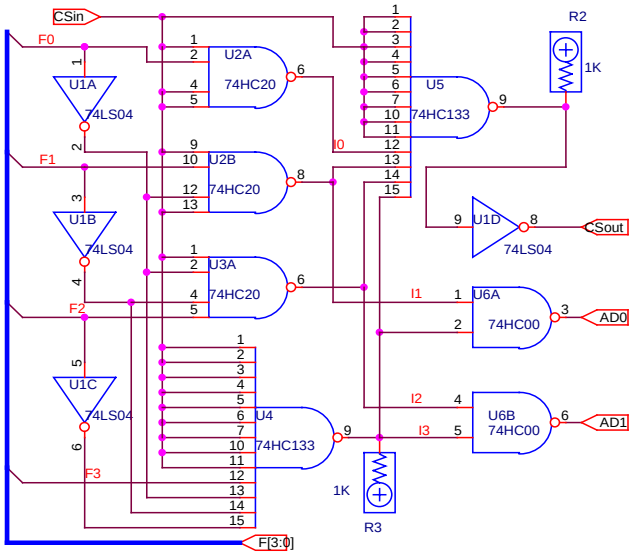


Fig. 6.3

far ciò utilizzano ciascuna una propria linea. Il circuito, gestore delle richieste di interruzione, le raccoglie tutte e, nel caso in cui, in un certo istante, c'è una sola richiesta, trasforma il numero d'ordine della linea attiva in un indirizzo codificato in binario (con un numero di bit opportuno), chiamato *vettore di interruzione*, se invece ci sono più richieste, determina quale tra di esse ha diritto ad avere la precedenza e ne codifica l'indirizzo.

Stabiliamo adesso che vogliamo *progettare un encoder di priorità di interruzione a quattro bit con la caratteristica che lo stesso circuito possa essere utilizzato in più esemplari, connessi tra di loro, in modo da realizzare rivelatori ed encoder di priorità con un numero maggiore di linee di interruzione(**espandibilità**).*

Per risolvere il nostro problema facciamo le seguenti definizioni.

- Csin (**C**hip **S**elect **I**n) variabile di controllo della funzione. Quando Csin= 0 l'indirizzo deve essere zero e Csout = 0.
- Csout (**C**hip **S**elect **O**ut) variabile di uscita di controllo della funzione; Csout= 1 se tutte le linee di ingresso sono a zero e Csin =1, Csout = 0 se almeno un ingresso è attivo o Csin= 0.
- F_3-F_0 le quattro variabili di ingresso, attive alte, sono le 4 linee di richiesta di interruzione. La linea con indice inferiore ha priorità maggiore.
- I_3-I_0 quattro variabili interne, solo una linea è attiva alla volta. I_i= 1 se la linea con indice *i* ha maggiore priorità di tutte le altre contemporaneamente attive.
- AD1-AD0 variabili che codificano l'indirizzo della linea di interruzione con maggiore priorità (indice inferiore)

Inputs					Ausiliary Out				Out		
Csin	F_3	F_3	F_1	F_0	I_3	I_2	I_1	I_0	AD1	AD0	CSout
0	X	X	X	X	0	0	0	0	0	0	0
1	X	X	X	1	0	0	0	1	0	0	0
1	X	X	1	0	0	0	1	0	0	1	0
1	X	1	0	0	0	1	0	0	1	0	0
1	1	0	0	0	1	0	0	0	1	1	0
1	0	0	0	0	0	0	0	0	0	0	1
I_3=Csin·F_3 F'_2·F'_1·F'_0					I_2=Csin F_2·F'_1·F'_0				I_1=Csin F_1·F'_0		
I_0=Csin F_0					AD1= I_3+I_2				AD0= I_3+I_1		
Csout =Csin·I'_3·I'_2·I'_1·I'_0											

Tab. 6.2

Le entrate nella tabella di verità (vedi Tab. 6.2) sono state ridotte, rispetto alle 32 necessarie, utilizzando le condizioni di don't care. Nella stessa tabella sono presentate le funzioni da realizzare sia per le variabili interne che per le uscite.

La Fig. 6.3 mostra la realizzazione del circuito. Notare che la realizzazione è fatta invece che con delle porte AND e OR con porte NAND (usando il teorema di De Morgan) per mancanza di funzioni integrate OR con numero sufficiente di ingressi.

CSin	F_0	F_1	F_2	F_3	AD1	AD0	Csout=CSin	F_4	F_5	F_6	F_7	AD1	AD0	CSout
1	0	0	1	X	1	0	0	X	X	X	X	0	0	0
1	0	0	0	0	0	0	1	0	1	X	X	0	1	0

Tab. 6.3

La Tab. 6.3 mostra l'uso di due delle funzioni qui descritte; una dedicato alle linee F_3-F_0 ed una dedicata alle linee F_7-F_4. Csin della prima è connesso costantemente ad 1 logico e quello della seconda è connesso all'uscita CSout della prima funzione. Le entrate in tabella illustrano due casi. Nella prima riga la linea F_2 è la linea con maggiore priorità attiva e nella seconda

riga è la linea F_5 con la maggiore priorità. Se come indirizzo si usano i bit Csout della prima funzione (bit più significativo) e le linee AD1 e AD0 della seconda funzione (bit ordinatamente meno significativi, caselle colorate) si vede come l'indirizzo rispecchia l'indice della linea con maggiore priorità.

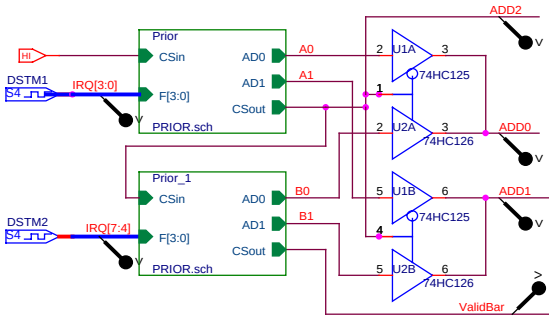


Fig. 6.4

La fig. 6.4 mostra una realizzazione di quanto dettato dalla Tab. 6.3. I due blocchi gerarchici *Prior* e *Prior_1* si riferiscono entrambi allo stesso circuito di Fig. 6.3. In questo modo viene realizzato un rivelatore/codificare di 8 linee di interruzione. Si noti che: Csin di Prior è stato posto ad 1 poiché questo è il primo circuito della serie e che l'indirizzo è formato da Csout del blocco che tratta le linee a maggiore priorità (indicato in Fig. 6.4 come ADD2) e dalle due linee ADD1, ADD0, scelte dalle uscite AD1 e AD0 di Prior se ADD3= 0 o dalle analoghe di Prio_1 se ADD2= 1. Una uscita, funzionalmente equivalente, avrebbe potuto essere ottenuta facendo l'OR delle due omologhe AD0 e AD1, fidando sul fatto che dette uscite sono a zero quando CSout è = 1.

Il timing di Fig. 6.5 riproduce, per il circuito realizzato come in Fig. 6.3 e usato come

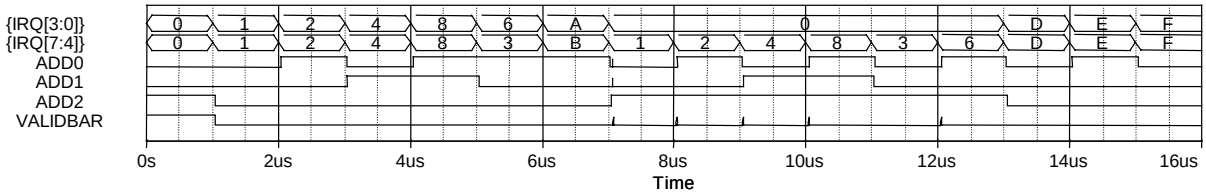


Fig. 6.5

rappresentato in Fig. 6.4, quanto descritto in Tab. 6.1. In particolare si nota che quando il contenuto del bus è zero (nessun uno presente) CSout del secondo blocco (chiamato ValidBar) è uguale ad 1 (significa praticamente che il valore 0 del bus indirizzo non è un indirizzo valido). Mentre in tutti gli altri casi in cui l'indirizzo è valido il suo livello logico è zero. I bus delle linee di interruzioni è letto in esadecimale. Considerare, quindi, la configurazione binaria per verificare che la priorità indicata si riferisce alla linea ad 1 logico con indice inferiore.

Per es. quando leggiamo il valore dei bus IRQ[7:4]= $(3)_H$=$(0011)_2$ e IRQ[3:0]= $(6)_H$=$(0110)_2$ il bit ad 1 con indice inferiore risulta essere in posizione 1 come è indicato dal timing.

Il lettore, si eserciti ad utilizzare *2n* circuiti così fatti e trovi le funzioni Booleane di interconnessione in modo tale da realizzare un encoder di priorità di *n x 8* linee di interruzione.

6.4 Decoder

Un circuito decodificatore (*decoder*) esegue, essenzialmente, una operazione logica inversa a quella del codificatore. I suoi *n* ingressi rappresentano i segnali logici con cui una certa informazione è codificata. Le sue *m* uscite sono una per ogni modo in cui l'informazione di ingresso si può presentare. Si noti che delle $m \leq 2^n$ linee (l'uguaglianza vale in assenza di condizione di don't care nel codice) *solo una è attiva*. La linea attiva, istante per istante, è quella corrispondente al codice presente, in quell'istante, in ingresso.

Es. 2. *Progettare un circuito decodificatore per la codifica dell'Es. 1.*

Il circuito è un decodificatore da 3 linee (o bit di ingresso) a 8 bit (o linee di uscita). La Tab. 6.1, che ci è servita per la codifica, ora può essere utilizzata per il progetto del decodificatore. Basta pensare che le variabili x, y, z sono gli ingressi e le f_j sono le funzioni Booleane di uscita desiderate. Ogni funzione di uscita dovrebbe essere minimizzata utilizzando la mappa relativa. Ma in questo caso, vista la mutua esclusione delle funzioni, un solo minterm contribuisce alla definizione di ogni una di esse. Pertanto non c'è possibilità di minimizzazione. Il minterm che determina la funzione si trova scrivendo in binario puro il valore del relativo indice.

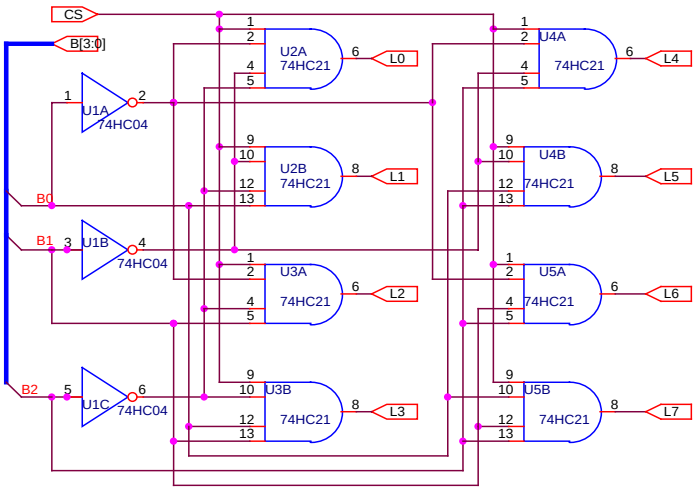


Fig 6.6

Così f_0 = x'y'z'; f_1 = x'y'z; e così via. In generale un decoder è un circuito realizzato con porte AND che ha *n* ingressi ed $m \leq 2^n$ uscite.

Lo schema che realizza questo decodificatore è rappresentato in Fig. 6.6. In essa, per motivi pratici, gli ingressi sono chiamati B3-B0 e le uscite L7-L0. Nella sua realizzazione si è tenuto conto del fatto che, in generale, le funzioni integrate nei circuiti MSI prevedono la possibilità di interconnessione tra circuiti identici, per la soluzione di problemi richiedenti la gestione di un numero maggiore di bit in ingresso. Nel nostro caso abbiamo definito una sola linea ausiliare in più. La quarta linea è chiamata *CS* (Chip Select) e serve a mettere a zero (stato non attivo) tutte le uscite del decoder. Il segnale di ingresso a CS è internamente collegata a tutte le porte AND, che quindi sono a 4 ingressi. Se il livello logico di CS = 0 tutte le linee di uscita saranno zero mentre se CS = 1 il circuito si comporta come se la linea non ci fosse. In Fig. 6.7 è presentato un decoder a 4 bit di ingresso e 16 bit in uscita, realizzato usando due

decoder a 3 bit come quello presentato in Fig. 6.6. I tre bit meno significativi sono collegati, in parallelo, ai due decoder e connessi agli ingressi dati, mentre all'ingresso CS di DEC_1 è collegato il bit più significativo del dato di ingresso (è destinato a decodificare gli indirizzi che vanno da 8 a 15), e all'analogo ingresso di DEC è collegato il suo valore complementare (è destinato a decodifiare gli indirizzi che vanno da 0 a 7). Si ricorda che è convezione generale, per rendere più chiari gli schemi logici, considerare gli ingressi indicati con lo stesso nome come collegati tra di loro. In Fig. 6.7. si sottintende che gli ingressi chiamati B[3:0] di entrambi i decoder sono collegati, ordinatamente, tra di loro ed con le linee del bus I[3:0]. Notare che un circuito integrato che dovesse contenere il decoder da 3 a 8 bit, ora presentato, userebbe tutte e 14 i piedini di un package dual in line. Dal timing di Fig. 6.8, si vede come le uscite sono mutuamente esclusive, cioè solo una linea di uscita è allo stato in un dato istante.

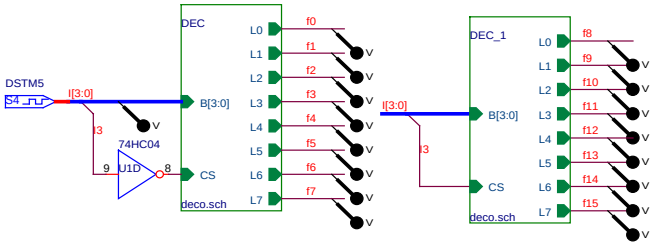


Fig. 6.7

Nei circuiti integrati è normale trovare, oltre agli ingressi necessari alla realizzazione della funzione dichiarata, degli altri ingressi di controllo (come il CS di cui si è parlato sopra). Essi sono aggiunti per permettere di collegare gli integrati tra di loro o in cascata o in parallelo. Il collegamento realizzato in Fig. 6.7 è in parallelo.

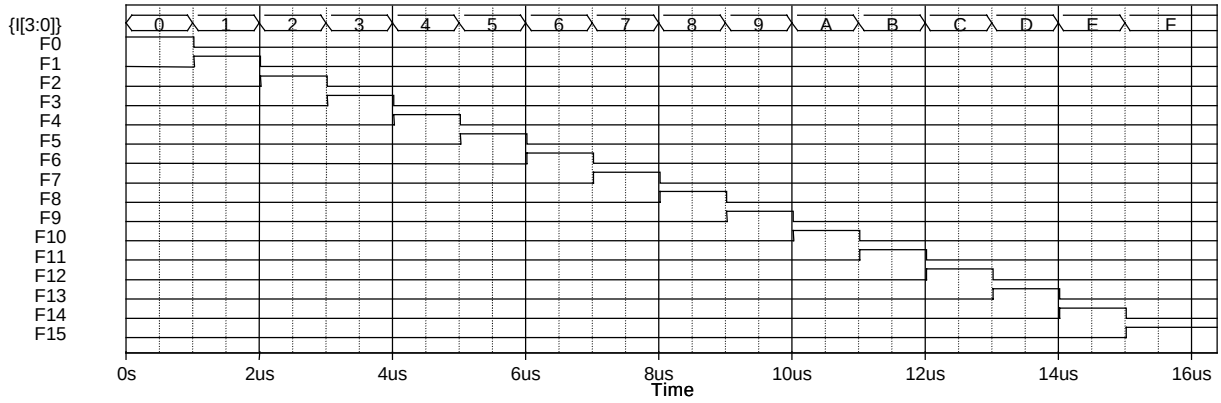


Fig 6.8

Il lettore si eserciti ad utilizzare da 3 a 9 decoder di questo tipo e provi che (con aggiunta solo di alcuni inverter), è possibile decodificare fino a 64 indirizzi diversi.

Oltre ad ulteriori *ingressi di controllo*, i circuiti MSI, spesso hanno anche delle *uscite di controllo* (in qualche modo legate alla funzione realizzata) e che sono *ausiliarie* al collegamento di più chip in cascata. Ne abbiamo già fatto un esempio quando abbiamo studiato l'encoder di priorità, ma più in là avremo modo di illustrarne altri.

Gli encoder e i decoder sono molto usati in quanto i sistemi digitali sono basati sulla codifica e decodifica delle informazioni. Una possibile applicazione, facile da capire, è quella del funzionamento di una telescrivente. Questa è una apparecchiatura che ha due modi di funzionare: uno locale e l'altro remoto. Nel funzionamento locale un operatore pigiando i suoi tasti la usa come una normale macchina da scrivere. In remoto quando l'operatore pigia i tasti, oltre alla scrittura su foglio di carta, avviene la trasmissione del codice binario del tasto pigiato (generalmente ASCII) ad un sistema digitale ad essa collegato (per es. un computer). Questa si chiama operazione di ingresso o di *input* per il sistema digitale di cui la telescrivente ne costituisce una periferica. Quando è invece il sistema digitale a volere fornire i dati, per esempio di misure o di calcoli effettuati (assieme alla loro identificazione), allora alla telescrivente, per ogni carattere che si deve stampare o di ogni altra azione che la stessa deve fare, viene inviato il relativo codice (ASCII nel nostro caso). Alla ricezione di un codice, il circuito decodificatore, che ha sede nella telescrivente, ne attiva la linea relativa e questa comanda per es. un relè che, attraverso altri mezzi elettromeccanici, fa si che il carattere venga stampato o l'azione eseguita. Questa si chiama operazione di uscita o di *output*. Abbiamo visto che il codice ASCII è un codice a 7 bit (a parte la parità) per cui il numero di uscite di un suo decodificatore debbono essere 128. Nel caso della telescrivente, alcune saranno utilizzate per la scrittura dei caratteri, come già detto, altre per andare a capo, altri per avanzare un rigo, per suonare il campanello, etc. etc.

Un'altra applicazione dei decoder, molto diffusa, è la decodifica degli indirizzi per es. delle memorie o delle periferiche di un calcolatore. Abbiamo accennato, quando abbiamo parlato di memorie, che queste sono organizzate a byte o word e che per accedere (leggere o scrivere) ad ogni una di esse è necessario indirizzarla. Gli indirizzi sono codificati in binario puro con un numero di bit che dipende dalla quantità di byte o word da indirizzare. Per es. una memoria da 64 Kbyte (1 Kbyte è = 1024 byte) servono 16 bit. Questo significa che dal decodificatore debbono uscire 64 x 1024 fili uno per ogni indirizzo. Per tale motivo i bit di ingresso di un decodificatore vengono spesso chiamati indirizzi. Chiaramente è una impresa piuttosto ardua diramare tutti questi fili esterni per i collegamenti (considerando che memorie di alcuni Mega byte sono ormai in uso in tutti i Personal Computer), pur potendosi, con la odierna tecnologia, fare collegamenti con fili sottilissimi (qualche micron), se non si utilizza qualche metodo per minimizzare il loro percorso. Nelle memorie a semiconduttore, il circuito integrato contiene la memoria organizzata in byte con, al suo interno, la decodifica necessaria. All'utente è richiesto di fornire la parola degli indirizzi codificati in binario. La memoria invece di un array lineare è pensata come una matrice. Per es. una memoria di 64 Kbyte diventa un array

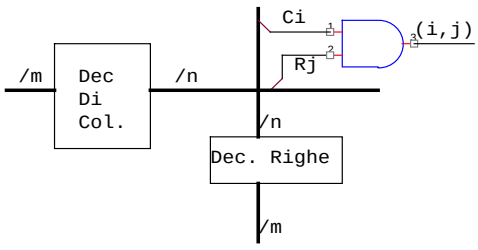


Fig. 6.9

quadrato di 256 x 256 byte. Per indirizzare una posizione di memoria allora basta conoscere l'indirizzo o coordinata X di riga e quella Y o di colonna. Per decodificare una riga o una colonna servono 256 fili e per indirizzare un byte serve solo utilizzare il filo di riga e quello di colonna attivo. L'AND di queste due linee formerà l'indirizzo decodificato di un byte. La Fig. 6.9 presenta due decoder uno di righe ed uno di colonne con *m* ingressi ed *n* uscite. L'AND dei segnali della riga *jesima* e colonna *iesima* produce il filo di comando per l'accesso del byte indirizzato da queste due coordinate.

6.5 Conversione di codici

La disponibilità di un gran numero di codici per gli stessi elementi di informazione permette l'uso di codici diversi nei diversi sistemi digitali. Delle volte è necessario usare l'uscita di un sistema digitale come ingresso di un altro. Se i due sistemi usano un codice diverso per lo stesso tipo di informazione è necessario inserire tra di essi un circuito di conversione. In questo modo un convertitore di codice rende i due sistemi compatibili.

Perché un convertitore di codice possa convertire da un codice binario A ad un codice binario B, è necessario fornire alle sue linee di ingresso la combinazione di bit relativo al codice A e ottenere dalle sue linee di uscita la corrispondente combinazione di bit per il codice B. Un circuito combinatorio esegue questa conversione per mezzo di porte logiche. La procedura di progetto di un convertitore di codice sarà illustrato per mezzo di un esempio specifico che riguarda la conversione da un codice BCD a quello eccesso di 3.

Le combinazioni di bit per i codici BCD ed eccesso di 3 sono listate in Tab. 1.2 di cap.1.

A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Tab. 6.4

Siccome questi codici usano 4 bit per rappresentare una cifra decimale, ci debbono essere 4 linee di ingresso e 4 di uscita. Decidiamo di chiamare A, B, C, D, le variabili di ingresso e w, x, y, z, le variabili di uscita. La tabella di verità relativa alla relazione delle variabili di ingresso con quelle di uscita è data in Tab. 6.4. La combinazione dei bit per gli ingressi e per le relative uscite è ottenuta direttamente dalla Tab. 1.2. Notiamo però che per 4 variabili di ingresso abbiamo 16 possibili combinazioni, mentre, nella tabella di verità, ne sono presentate soltanto 10. Le sei combinazioni non elencate per le variabili di ingresso sono combinazione di indifferenza (don't care). Poiché queste non si verificano mai abbiamo la libertà di assegnare alle variabili di uscita sia 0 che 1, a seconda della convenienza.

In Fig. 6.10 sono disegnate le mappe relative alle singole variabili di uscita per ottenere la minimizzazione delle relative funzioni Booleane.

Ciascuna delle 4 mappe rappresenta una delle 4 uscite di questo circuito in funzione delle 4 variabili di ingresso. I quadratini marcati 1 sono ricavati dalla tabella di verità. Le 6 combinazioni di don't care sono segnalati con una X. Un possibile modo di semplificare le funzioni in somma di prodotti è data sotto la stessa mappa.

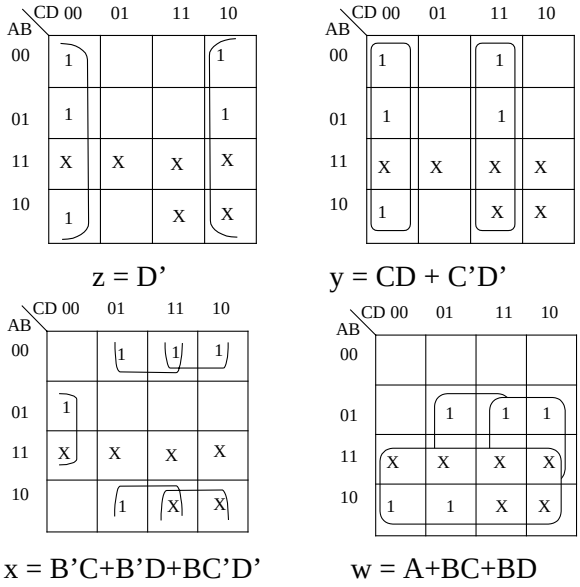


Fig. 6.10

Esistono, per questo circuito, diverse possibilità di realizzazione. Le espressioni mostrate in Fig. 6.10 possono essere manipolate algebricamente allo scopo di usare delle porte che sono comuni a più uscite. Questa manipolazione, data più in basso, mostra la flessibilità di cui si dispone se accettiamo di realizzare un circuito a più uscite con tre o più livelli di porte.

La Fig. 6.11 rappresenta lo schema logico di un circuito convertitore di codice da BCD ad eccesso di 3. Le funzioni logiche implementate sono nominate in modo diverso a quanto fatto precedentemente, per adattarla alla denominazione del bus. La corrispondenza è la seguente:

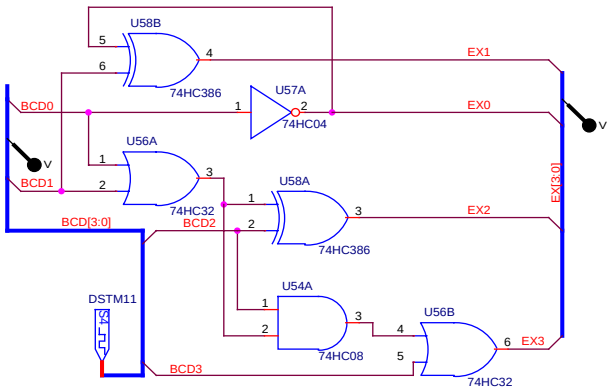


Fig 6.11

A->BCD3; B->BCD2; C->BCD1;
D->BCD0,
z->Ex0 = D';
y->Ex1 = CD + C'D' = C ⊙ D;
x->Ex2 = B'C + B'D + BC'D' = B'(C + D) + BC'D' =B'(C + D) + B (C + D)' = B ⊕ (C + D)
w->Ex3 = A + BC + BD = A + B(C + D)

Allo scopo di ridurre il numero di integrati necessari alla eventuale realizzazione si è anche cambiato l'EQ in XOR (U58B) avendo cura di fornire uno degli ingressi invertito (già presente, D') in modo da ottenere in ogni caso la funzione EQ desiderata.

La Fig. 6.12 mostra il timing della funzione realizzata con il circuito di Fig. 6.11.

Si ricorda che le funzioni per la conversione ricavate derivano da una semplificazione in cui sono state utilizzate condizioni don't care. La scelta fatta ha determinato le uscite che si leggono nel timing per quelle combinazioni. Notare che mentre le uscite D, E, F non possono

rappresentare numeri in eccesso di tre le successive uscite: 8, 9 ed A sono delle combinazioni possibili per tale codice.

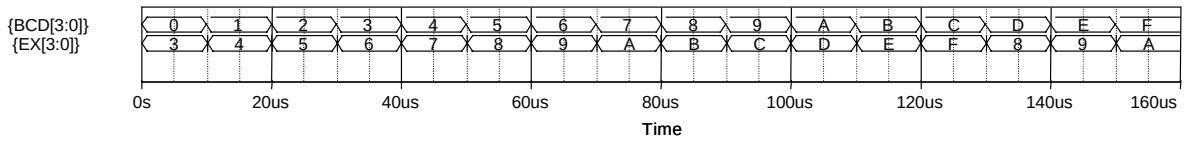


Fig 6.12

Una scelta più appropriata della minimizzazione potrebbe risultare dall'assegnazione, per quelle configurazioni, di un valore alle singole funzioni tali che le configurazioni corrispondenti fossero state una o più configurazioni impossibili per il codice eccesso di 3.

6.6 Comparator

Un *comparator* (paragonatore) è un circuito della classe MSI che paragona due numeri A e B e determina la loro relazione d'ordine. Le uscite del comparator debbono segnalare se: A > B, A = B o A > B, cioè ha tre uscite. Il numero degli ingressi dipende dal numero di bit che codifica l'informazione. Più precisamente se ogni numero è espresso in *n* bit, il numero di ingressi è il doppio: *2n*. I dati digitali che possono essere paragonati sono: i numeri binari, in numeri decimali rappresentati in codice binario, o qualsiasi altra informazione su cui è possibile definire una relazione di ordine. Arriveremo a trovare una soluzione che può essere facilmente utilizzata per un numero di bit qualsiasi partendo dalle soluzioni per il confronto di informazioni di 1 bit e di 2 bit.

Numero di bit =1. Chiamiamo A il primo numero e B il secondo numero e x, y e z le uscite riferite rispettivamente a A > B, A = B e A < B. Essendo questi codificati con un solo bit otteniamo la tabella di verità mostrata in Tab. 6.5. Notare che le funzioni di uscita sono mutuamente esclusive; infatti in ogni riga solo una delle funzione ha il valore 1. La minimizzazione è impossibile pertanto si ha: x = AB'; y = A ⊙ B; z = A'B.

A	B	A>B x	A=B y	A<B z
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Tab. 6.5

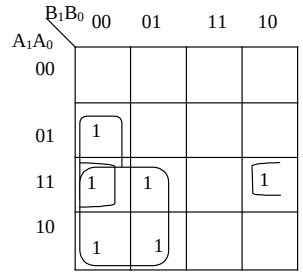
Numero di bit =2. Chiamiamo A_0, A_1 i bit rispettivamente meno significativo e più significativo del primo numero e B_0, B_1 i rispettivi bit del secondo numero e x, y e z le uscite riferite rispettivamente a A > B, A = B e A <B.
Applicando l'enunciato del problema abbiamo la tabella di verità mostrata in Tab. 6.6. Le quattro combinazioni di ingresso che rendono i due numeri uguali cioè y = 1 sono quelli con $A_1A_0 = B_1B_0$.

Le sei combinazioni di ingresso che rendono A > B cioè x = 1 sono quelli con $A_1A_0 > B_1B_0$.
Le sei combinazioni di ingresso che rendono A < B cioè z = 1 sono quelli con $A_1A_0 < B_1B_0$.
Notare che le funzioni di uscita sono ancora mutuamente esclusive mostrando essere una caratteristica generale indipendente dal numero di bit. La minimizzazione di ciascuna funzione di uscita è fatta utilizzando le tre mappe di Fig. 6.13 in cui è indicata anche la funzione minimizzata. Notare la simmetria delle mappe e delle funzioni Booleane: Gli 1 di y giacciono nei quadrati in diagonale, e le funzioni x e z sono simili: A e B sono intercambiati.

A_1	A_0	B_1	B_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

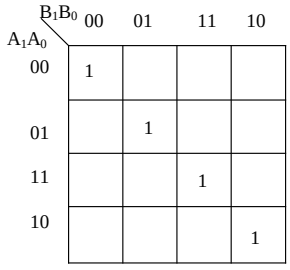
Tab. 6.6

La tabella di verità che paragona due numeri binari ad *n* bit richiede 2^{2n} righe e diventa non mangiabile per n > 2. Tuttavia i circuiti che posseggono



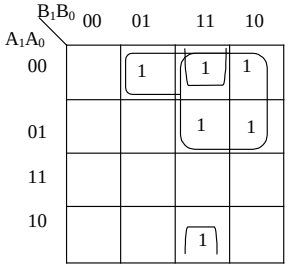
$x = A_1B'_1 + A_1A_0B'_0 + A_0B'_1B'_0$

A > B



$y = m_0 + m_5 + m_{10} + m_{15}$

A = B



$z = B_1A'_1 + B_1B_0A'_0 + B_0A'_1A'_0$

A < B

Fig. 6.13

una certa quantità di simmetria spesso possono essere progettati per mezzo di un procedimento algoritmico. Un *algoritmo* è un procedimento che specifica un insieme di regole iterative che forniscono la soluzione al problema. Come esemplificazione del metodo sarà derivato un algoritmo di progetto del nostro problema.
L'algoritmo consiste nella diretta applicazione del metodo normalmente utilizzato per il confronto di due numeri. Consideriamo due numeri A e B con 4 cifre. Scriviamo i coefficienti in ordine decrescente di significatività, cioè $A_3A_2A_1A_0$ e $B_3B_2B_1B_0$. I due numeri sono uguali se contemporaneamente $A_3 = B_3$; $A_2 = B_2$; $A_1 = B_1$ e $A_0 = B_0$. Quando le cifre sono binarie due cifre della stessa significatività sono uguali quando entrambi sono 1 o 0. Indicando, come in precedenza con x la funzione A > B, con y la funzione A = B e con z la funzione A < B, l'uguaglianza si esprime come:

$$y = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

Per determinare se è A ≠ B dobbiamo paragonare i due numeri a partire dalla coppia di cifre più significative. Se le due cifre sono uguali, viene paragonata la successiva coppia di cifre. Il paragone continua finché non si trova una coppia di cifre diverse. Se la corrispondente cifra di A è maggiore di quella di B, A > B altrimenti A < B. Nel caso di cifre binarie la sequenza di paragoni può essere espressa logicamente dalle seguenti funzioni Booleane.

$$x = A_3B'_3 + A_2B'_2(A_3 \odot B_3) + A_1B'_1(A_3 \odot B_3)(A_2 \odot B_2) + A_0B'_0(A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)$$

$$z = A'_3B_3 + A'_2B_2(A_3 \odot B_3) + A'_1B_1(A_3 \odot B_3)(A_2 \odot B_2) + A'_0B_0(A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)$$

La realizzazione delle tre funzioni di uscita è più semplice di quando sembra perché le uscite di disuguaglianza possono usare parte delle uscite parziali generate per l'uguaglianza. Lo schema logico per un comparatore a 4 bit è mostrato in Fig. 6.15. In parole, la funzione Booleana per x cioè A > B è uguale ad 1 logico se $A_3 = 1$ e $B_3 = 0$ o $A_2 = 1$ e $B_2 = 0$ (purché sia $A_3 = B_3$), o se $A_1 = 1$ e $B_1 = 0$ (purché sia $A_3 = B_3$ e $A_2 = B_2$) o se $A_0 = 1$ e $B_0 = 0$ (purché sia $A_3 = B_3$ e $A_2 = B_2$ e $A_1 = B_1$).

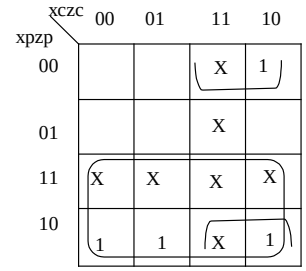
Il procedimento da applicare per ottenere un comparatore con più di 4 bit è ovvio. I comparatori per numeri decimali utilizza lo stesso algoritmo eccetto che bisogna paragonare 4 bit per ogni cifra decimale.

Notare che l'aumento del numero di bit da paragonare fa crescere il numero di ingressi alle porte AND e OR. Per questo motivo non sono disponibili circuiti integrati che risolvono il problema del paragone a più di 4 bit. I comparatori a 4 bit tuttavia sono modificati leggermente, rispetto allo schema logico, essenziale alla soluzione del problema, in modo che, (collegandone più di uno *in cascata*) possano essere utilizzati per paragonare un numero qualsiasi di bit senza usare porte esterne (*Espandibilità Completa*).

xp	zp	xc	zc	A>B	A<B
1	0	X	X	1	0
0	1	X	X	0	1
0	0	1	0	1	0
0	0	0	1	0	1

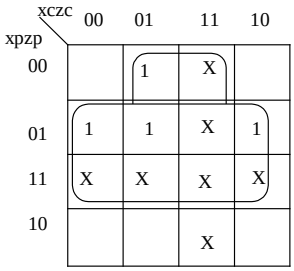
Tab. 6.7

Le uscite A> B, A= B e A< B di uno stadio che paragona 4 bit meno significativi sono connessi ai corrispondenti ingressi (*Cascading Inputs*)



A > B = xp + (zp)' xc

Fig. 6.14a



A < B = zp + (xp)' zc

Fig. 6.14b

A> B, A= B e A< B del prossimo stadio, che paragona i 4 bit più significativi. Lo stadio che paragona i 4 bit meno significativi deve avere il suo ingresso A = B collegato ad 1 logico e gli altri due ingressi allo 0.

Mentre è evidente che A= B è data dall'AND dei risultati parziali dei nible meno e più significativo, per le funzioni di disuguaglianza dobbiamo generare la tabella di verità, Tab. 6.7. In questa *xp* e *zp* indicano, rispettivamente, A> B e A< B per i bit presenti (in ingresso al

circuito) mentre *xc* e *zc* indicano, rispettivamente il risultato ottenuto, (A> B e A< B) dal confronto tra la coppia di nible meno significativa (ingressi in cascata al circuito).
La tabella è riempita considerando che se il risultato del confronto della coppia dei nible presente è la **non uguaglianz**a bisogna propagarlo, indipendentemente dal risultato della coppia precedente (meno significativa), se invece è **l'uguaglianza** bisogna propagare il risultato della coppia di nible meno significativa.

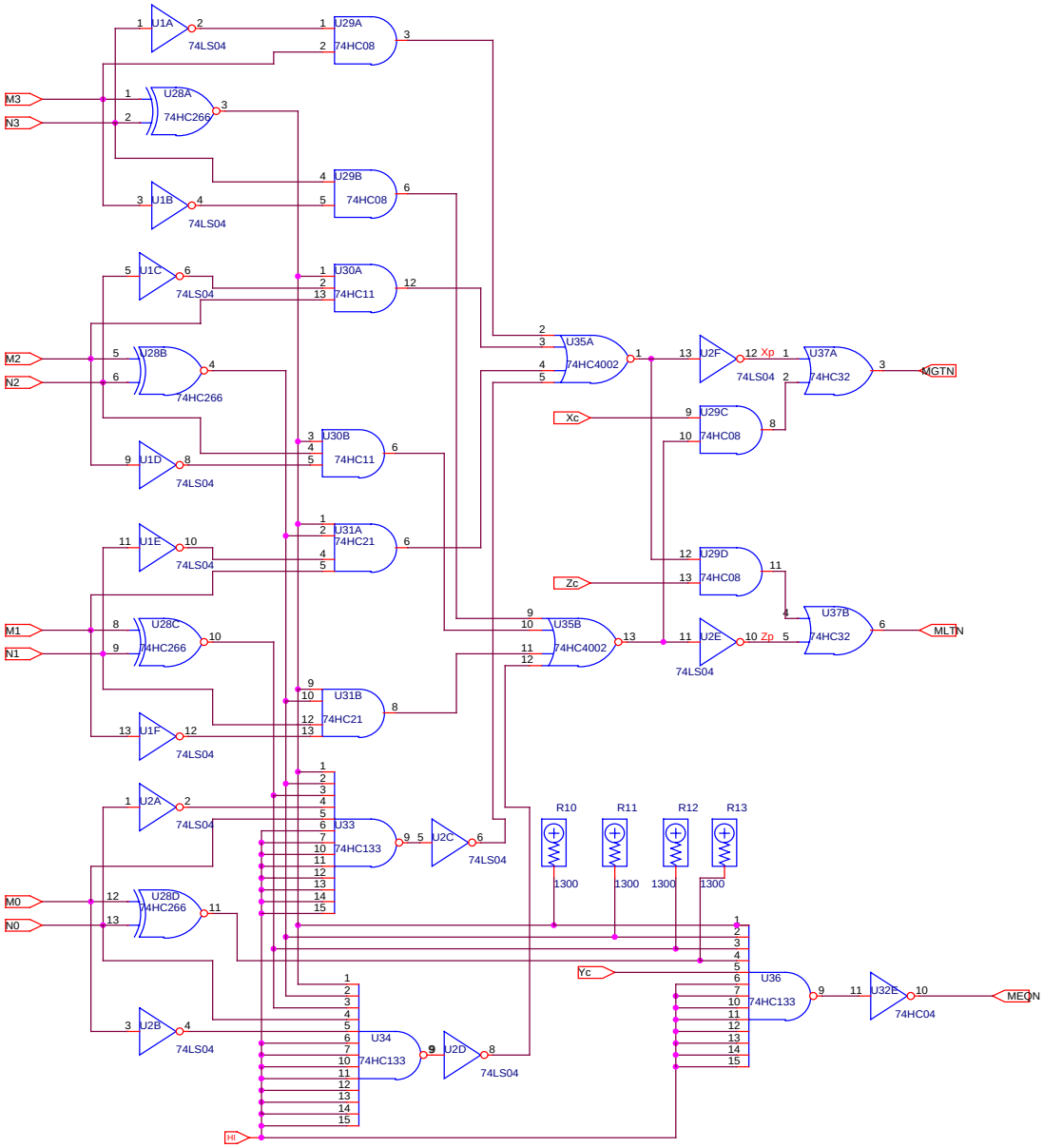


Fig. 6.15

La minimizzazione è fatta usando le mappe di Figg. 6.14a e 6.14b
Le condizioni di don't care sono relativi ai minterm in cui sono presenti due 1 nella coppia xp, zp o xc, zc che, per la mutua esclusività dei risultati, non possono presentarsi.

Il circuito di Fig. 6.15 rappresenta la realizzazione di un comparatore di due numeri M ed N a 4 bit. Notare che oltre ai due numeri da paragonare sono presenti tre ingressi di controllo Xc, Yc, Zc utilizzati come deriva dalle equazioni di Fig. 6.14a e 6.14b.

La Fig. 6.16 mostra l'uso di due di tali circuiti collegati in modo da poter paragonare due numeri interi ad 8 bit. La fig. 6.17 ne mostra il timing. Notare che le uscite sono indicate con

- MGTN (M Greater Then N) quando M>N (rimpiazza la variabile x)
- MEQN (M Equal N) quando M=N (rimpiazza la variabile y)
- MLTN (M Less Then N) quando M<N (rimpiazza la variabile z)

Si noti che nel paragone di $M=(03)_{16}$ e $N=(80)_{16}$ il nible meno significativo di M è maggiore del corrispondente nible di N (Xop=1) tuttavia il risultato finale mostra che M è minore di N, come deve essere.

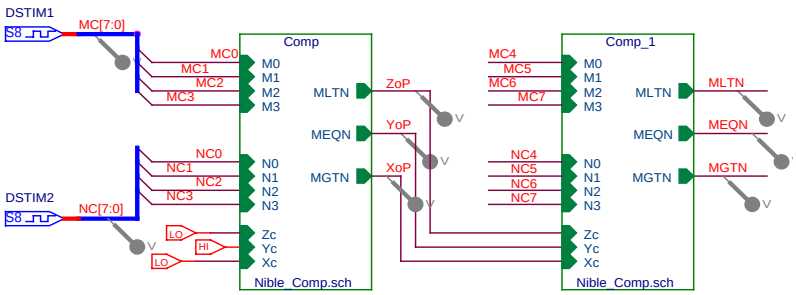


Fig. 6.16

Il lettore, supponga di dover paragonare due numeri interi per es. a 16 bit. Seguendo la traccia su esposta realizzi uno schema a blocchi suddividendo ciascuno dei due numeri in quattro nible e utilizzando un comparatore per ogni uno dei nible con lo stesso peso e quindi faccia i collegamenti in modo che il risultato finale del paragone sia corretto.

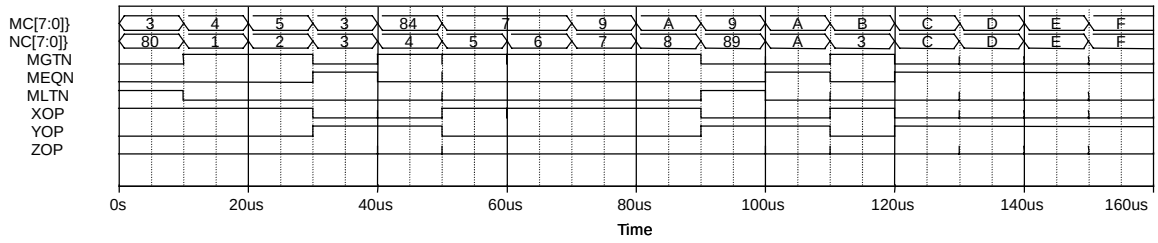


Fig. 6.17

Si noti che è possibile fare in modo che sia la coppia di nible più significativa ad essere connesso al primo paragonatore della catena mentre la coppia di nible meno significativo è connesso all'ultimo paragonatore. Si dimostri che questo può essere ottenuto scambiando, nella Tab. 6.7, xp con xc, e zp con zc. Si disegni quindi il nuovo modo di realizzazione delle equazioni di interfaccia. Con questo modo diverso di trattare il paragone tra nible più e meno significativo rifare i circuiti a blocchi relativi.

6.7 Analisi di un Circuito logico Combinatorio

Il ***progetto*** di un circuito combinatorio parte dalla specificazione verbale della funzione richiesta e finisce con un insieme di funzioni Booleane e/o un circuito logico. ***L'analisi*** di un circuito combinatorio è in qualche modo il processo inverso: parte da un circuito logico e finisce con un insieme di funzioni Booleane, una tabella di verità o con una spiegazione verbale del funzionamento del circuito. Se lo schema logico che deve essere analizzato è accompagnato dal nome di una funzione o da una spiegazione si assume che, come risultato dell'analisi, si vuole verificare la funzione dichiarata.

Il primo passo nel processo di analisi è di essere sicuri che il circuito proposto è combinatorio e non sequenziale. Lo schema di un circuito combinatorio presenta delle porte logiche senza alcun *collegamento di reazione* o elementi di memoria. ***Un collegamento di reazione è una connessione tra l'uscita di una porta e l'ingresso di un'altra porta la cui uscita è utilizzata dalla prima*** (vedi Fig. 6.18).

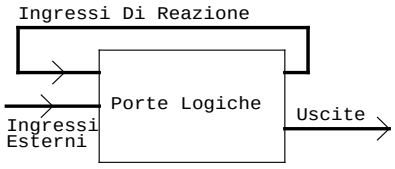


Fig. 6.18

Un collegamento di reazione o un elemento di memoria, in un circuito digitale, definisce un circuito sequenziale e deve essere analizzato con un procedimento diverso, che vedremo più avanti. Una volta che è stato verificato che lo schema si riferisce ad un circuito combinatorio, si può cominciare ad ottenere le funzione Booleane di uscita e/o la tabella di verità. Se il circuito è accompagnato da una spiegazione verbale della sua funzione allora per la verifica è sufficiente trovare le funzioni Booleane o la tabella di verità. Se bisogna trovare la funzione del circuito, allora è anche necessario interpretare l'operazione del circuito dalla tabella di verità che è stata trovata. Il successo di tale ricerca è favorito da una pregressa familiarità ed esperienza con i circuiti digitali. L'abilità di corredare una tabella di verità con una spiegazione del funzionamento è un'arte che si acquisisce con l'esperienza.

Per ottenere le funzioni Booleane da uno schema logico, procedere come segue:

1) Marcare con simboli arbitrari tutte le uscite delle porte che sono funzioni delle sole variabili di ingresso. Ottenere la funzione Booleana di ciascuna porta.
2) Marcare con altri simboli arbitrari le uscite di quelle porte che sono funzione sia delle variabili di ingresso che delle funzioni precedentemente trovate. Trovare le funzioni Booleane di queste porte.
3) Ripetere il processo delineato nel passo (2) fino a che non si ottengono tutte le uscite del circuito.
4) Attraverso ripetute sostituzione delle funzioni precedentemente definite, ottenere le funzioni Booleane di uscita in funzione delle sole variabili di ingresso.

L'analisi del circuito combinatorio di Fig. 6.19 illustra il procedimento ora enunciato. Notiamo che il circuito ha 3 ingressi binari, A, B, e C, e due uscite binarie, F_1 ed F_2. Le uscite delle varie porte sono marcate con simboli intermedi.

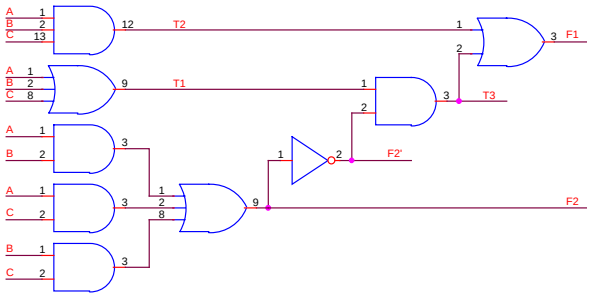


Fig. 6.19

Le uscite delle porte che sono funzione solo delle variabili di ingresso sono: F_2, T_1 e T_2. Le funzioni Booleane di queste 3 uscite sono:

$$F_2 = AB + AC + BC;$$

$$T_1 = A + B + C; \qquad T_2 = ABC;$$

Ora consideriamo le uscite delle porte che sono funzione dei simboli già definiti.

$$T_3 = F'_2T_1; \qquad F_1 = T_3 + T_2$$

La funzione Booleana F_2 espressa sopra è già funzione delle solo variabili di ingresso. Per ottenere F_1 in funzione di A, B e C facciamo la serie di sostituzioni successivamente indicata:

$$F_1 = T_3 + T_2 = F'_2T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC =$$
$$(A' + B')(A' + C')(B' + C')(A + B + C) + ABC =$$
$$(A' + B'C')(A\,B' + AC' + BC' + B'C) + ABC=$$
$$A'BC' + A'B'C + AB'C' + ABC = C(AB + A'B') + C'(\,A'B + AB') =$$
$$C\,(A \odot B) + C'(A \odot B)' = A \odot B \odot C$$

Se uno vuole proseguire nell'analisi per determinare il processo di informazioni realizzato da questo circuito, si deve derivare la tabella di verità direttamente dalle funzioni Booleane e cercare di riconoscere una operazione familiare. Questo esempio è un Full Adder con F_1 la funzione somma ed F_2 l'uscita del riporto. A, B e C sono i tre ingressi che vengono sommati aritmeticamente. Il Full Adder sarà descritto successivamente.

Una volta che sono note le funzioni Booleane la derivazione della tabella di verità del circuito è un processo immediato.

Per ottenere la tabella di verità direttamente dallo schema circuitale senza passare per il ritrovamento delle funzioni Booleane, procedere come segue:

1) Determinare il numero delle variabili di ingresso del circuito. Listare tutte le combinazione degli ingressi.
2) Fare tante colonne quanti sono i simboli scelti per le funzioni intermedie e finali.
3) Ottenere la tabella di verità per le uscite di quelle porte che dipendono soltanto dalle variabili di ingresso.
4) Ottenere la tabella di verità per le funzioni intermedie e finali le cui porte dipendono dai valori precedentemente trovati, finché non sono state determinate i valori per tutte le colonne.

Questo processo può essere illustrato usando il circuito di Fig. 6.19. Nella Tab. 6.8 formiamo le 8 combinazioni per le 3 variabile di ingresso. La tabella di verità per F_2 è determinata direttamente dai valori delle variabili di ingresso. $F_2 = 1$ per le combinazioni che hanno 2 o 3 ingressi = 1. La tabella di verità per F'_2 è il complemento di F_2. Le tabelle di verità per T_1 e T_2 sono rispettivamente funzione OR e AND delle variabili di ingresso. I valori per T_3 sono derivati da T_1 e F'_2: $T_3 = 1$ quando entrambe le funzioni sono 1. Finalmente $F_1 = 1$ per quelle combinazioni per cui o T_2 o T_3 o entrambi sono uguali ad 1.

A	B	C	F_2	F'_2	T_1	T_3	T_2	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	1	0	1
0	1	0	0	1	1	1	0	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	1	0	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	0	1	1

Tab. 6.8

Consideriamo ora un circuito combinatorio che ha delle combinazioni don't care. Quando si progetta un tale circuito, le combinazioni don't care sono segnate nella mappa con una X ed poi viene assegnato il valore 1 o 0 a seconda di cosa è conveniente per la semplificazione delle funzioni Booleane.

Quando viene analizzato un circuito che ha combinazioni don't care, la situazione è completamente diversa. Anche se noi ipotizziamo che le combinazioni don't care non si verificano mai. La realtà è che se una qualsiasi di queste combinazioni è applicata all'ingresso del circuito, (intenzionalmente o per errore), avremo certamente una uscita binaria. Il valore di questa uscita dipende dal valore che è stato assegnato alle X in fase di progetto. Parte dell'analisi di un circuito del genere potrebbe proporsi di determinare i valori in uscita per tutte le combinazioni di ingresso don't care. Come esempio consideriamo il convertitore BCD eccesso di 3 progettato nel Par. 6.4. Le uscite che si ottiene dalle 6 combinazioni non usate dal codice BCD sono date in Tab. 6.10. Queste uscite possono essere derivate per mezzo del metodo della tabella di verità appena spiegato. Nel caso particolare le uscite possono essere ottenute direttamente dalle mappe di Fig. 6.10. Dalle stesse mappe e dalla semplificazione usata possiamo determinare se ad ogni X è stato assegnato il valore 1 o 0. Per es. il quadrato 1010 è stato incluso come 1 per l'uscita w, x, e z ma non per y. Quindi le uscite per detta combinazione di valori è wxyz = 1101 come indicate in Tab. 6.9. Notiamo ancora che, le prime 3 uscite della tabella non corrispondono a nessuna combinazione ammessa per il codice eccesso di 3, e che le ultime 3 uscite corrispondono rispettivamente a 5, 6 e 7. Questa coincidenza è funzione soltanto della scelta fatta durante il progetto.

Ingressi BCD non usati				uscite			
A	B	C	D	w	x	y	z
1	0	1	0	1	1	0	1
1	0	1	1	1	1	1	0
1	1	0	0	1	1	1	1
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	1
1	1	1	1	1	0	1	0

Tab. 6.9

6.8 Multiplexer

Il *multiplexer* (MPX) è un circuito che è stato già presentato come applicazione dei buffer tri-state. Ricordiamo che l'operazione di *multiplexing* consiste nel trasmettere un gran numero di informazioni su un numero di linee inferiore. La Fig. 6.20 presenta, nella sezione (a), la funzione equivalente, nel caso in cui il numero degli ingressi è 4 ed una sola uscita, attraverso un commutatore di tipo meccanico manuale. A seconda della posizione del commutatore viene selezionata, in uscita, una diversa funzione.

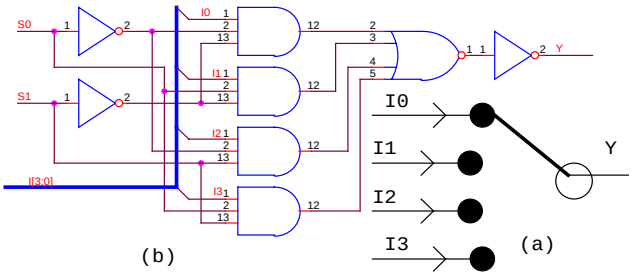


Fig. 6.20

Il multiplexer digitale consiste di un circuito combinatorio che seleziona i dati provenienti da 2^n linee di ingresso e li trasferisce su di una sola linea. La scelta della connessione di trasferimento ingresso-uscita è controllato da un insieme di *n* linee di selezione o di controllo. Un esempio di multiplexer è quello di Fig. 6.20b.

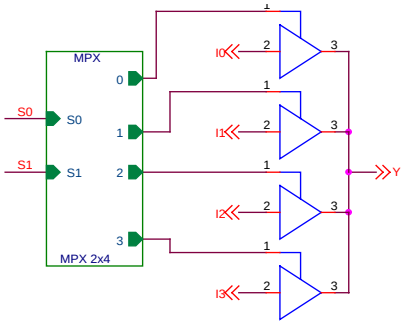


Fig. 6.21

Le quattro linee di ingresso I_0, I_1, I_2, I_3 sono applicate a 4 porte AND le cui uscite vanno ad un singolo circuito OR. Soltanto una linea di ingresso alla volta è collegata all'uscita. Le linee di controllo S_0 ed S_1 determinano quale degli ingressi deve essere connessa alla uscita. Le 4 AND assomigliano ad un circuito di decodifica ed in realtà decodificano gli indirizzi delle 4 linee di ingresso.

La funzione Booleana dell'uscita di un multiplexer a 4 ingressi mostra chiaramente come ciò può essere ottenuto.

$$Y = I_0(S'_1S'_0) + I_1(S'_1S_0) + I_2(S_1S'_0) + I_3(S_1S_0)$$

Il termine in parentesi è l'indirizzo della linea che si intende selezionare.

Un altro modo di realizzare un multiplexer è quello di utilizzare un decodificatore le cui linee di uscita controllano dei buffer tri-state. Per lo stesso multiplexer dell'es. precedente possiamo utilizzare il circuito di Fig. 6.21 in cui le funzione di decodifica sono chiaramente distinte dagli ingressi che debbono essere collegati in uscita.

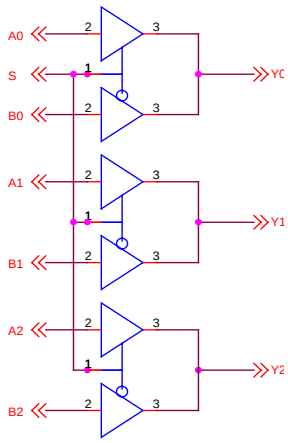


Fig. 6.22

La Fig. 6.22 è rappresentato un esempio di multiplexer che seleziona uno dei due dati: A o B. Ciascun ingresso consiste di 3 bit A_2, A_1, A_0 e B_2, B_1, B_0. Una selezione ad un bit determina

quale degli ingressi, A o B devono essere collegati in uscita. Questo consiste, come è evidente, da tante unità di selezione identiche (una per ogni bit) controllate dalle stesse linee di selezione. Le funzioni Booleane delle uscite sono:

$Y_0 = B_0S + A_0S'$; $Y_1 = B_1S + A_1S'$; $Y_2 = B_2S + A_2S'$;

Per la realizzazione alternativa di questo circuito, con porte AND ed OR, basta realizzare le funzioni Booleane sopra scritte.

In generale un multiplexer di *m* dati di ingresso, *I(m,k)*, ognuno costituito da *k* bit, ha *k* uscite e richiede *n* di linee di controllo (con $m = 2^n$) per decodificare gli indirizzi degli *m* dati in ingresso. Esso può essere realizzato con un circuito decoder ad *n* ingressi e $m = 2^n$ uscite che pilotano *k* sezioni di selezione, ognuna costituita da *m* buffer tri-state (uno per ogni linea dati del k.mo bit), le cui uscite sono collegate in wiring OR. La Fig. 6.23 mostra uno schema a blocchi di tale circuito in cui m=32 e k=32.

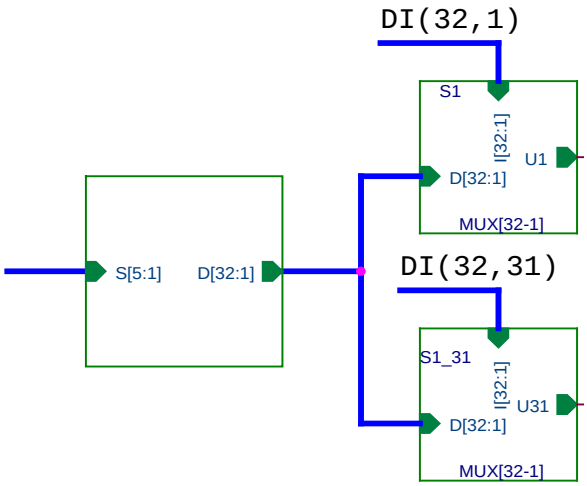


Fig 6.23

Una interessante applicazione di un circuito multiplexer è il suo uso come un elemento logico universale; cioè un circuito che realizza qualsiasi funzione Booleana di *n* variabili. Un circuito multiplexer con *k* linee di selezione e 2^k linee di ingresso è un elemento logico universale per funzioni Booleane di $k + 1$ variabili. Per es. il circuito di Fig. 6.14 con due linee di controllo S_1 e S_0 e 4 linee di ingresso I_3, I_2, I_1 e I_0, può essere usato per la generazione di una qualsiasi funzione Booleana di 3 variabili $Y(I, S_1, S_0)$ dove S_1, S_0 ed I sono le tre variabili di ingresso. Espandendo la funzione nella somma di prodotti di minterm, è possibile determinare i valori per gli ingressi I_3, I_2, I_1 e I_0. Questi valori dovranno essere uno dei seguenti: 0, 1, I ed I'. Gli ingressi 0 ed 1 sono segnali logici fissi, gli ingressi I ed I' sono i valori, rispettivamente normale e complementati, della terza variabile. Per es. consideriamo la seguente funzione Booleana di tre variabili espressa nella somma di minterm:

$$Y(I, S_1, S_0) = m_0 + m_3 + m_4 + m_6$$

Per realizzare questa funzione con il circuito multiplexer di Fig. 6.14 o 6.15 bisogna determinare i valori di I_3, I_2, I_1 e I_0. Si esprime prima la funzione nella somma di minterm delle 3 variabili:

$$Y(I, S_1, S_0) = I'S'_1S'_0 + I'S_1S_0 + IS'_1S'_0 + IS_1S'_0$$

Poichè bisogna che ci sia un solo termine con lo stesso indirizzo, combiniamo due minterm se e solo se è possibile l'eliminazione della variabile I o I'. Per il nostro esempio posiamo combinare il primo e terzo minterm per cui l'espressione per Y diventa:

$$Y = S'_1S'_0 + I(S_1S'_0) + I'(S_1S_0)$$

Paragonando questa espressione con la funzione Booleana del multiplexer data sopra fissiamo i valori per le variabili di ingresso:

$I_0 = 1$; $I_1 = 0$; $I_2 = I$; $I_3 = I'$;

6.9 Demultiplexer

Il *Demultiplexing* è l'operazione inversa di quella del *multiplexing* e si riferisce alla operazione di ricezione di informazioni o dati da un piccolo numero di linee e la loro selettiva distribuzione ad un numero di linee di destinazione maggiore.

La Fig. 6.24 presenta, nella sezione (a), la funzione equivalente nel caso in cui il numero delle uscite è 5 ed un sol ingresso, attraverso un commutatore di tipo meccanico-manuale. A seconda della posizione del commutatore una sola delle uscite fornisce il segnale di ingresso.

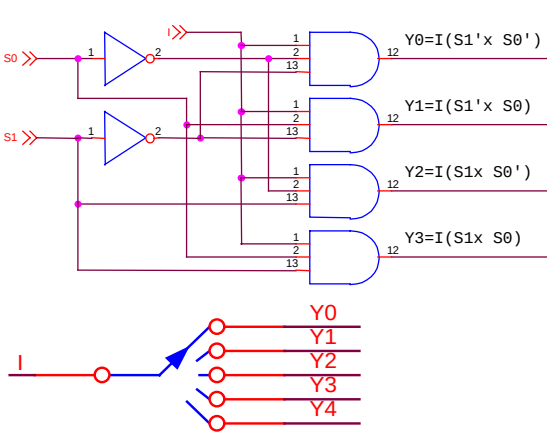


Fig. 6.24

Un esempio di demultiplexer digitale è presentato in Fig. 6.24(b). Una sola linea di ingresso è commutata su quattro identiche linee di uscita selezionate (indirizzate) da 2 linee di controllo. Esso consiste di quattro identiche porte AND a 3 ingressi, 2 dei 3 ingressi sono una delle combinazione delle 2 variabili di controllo e il terzo ingresso è la variabile da commutare. La singola variabile di ingresso può essere collegata a tutte e quattro le uscite, ma soltanto una uscita alla volta è attiva. Se il singolo ingresso è connesso permanentemente ad un segnale logico 1, un demultiplexer può funzionare come un circuito di decodifica .

Un demultiplexer può a sua volta essere realizzato con un decoder e dei buffer tri-state.

La Fig. 6.25 mostra tale realizzazione. Questo costituisce anche un esempio di quando è necessario dotare le uscite dei buffer tri-state di resistenze di pull up o pull down. Infatti uno ed uno solo dei buffer è attivo in ogni istante e gli altri sono in tri-state. In questo caso, se non colleghiamo le uscite a delle resistenze collegate o all'1 o allo 0 logico queste sarebbero in uno stato logico indeterminato. La connessione indicata con D, in Fig. 6.25, ha quindi lo scopo di fissare l'uscita logica a 0 o ad 1.

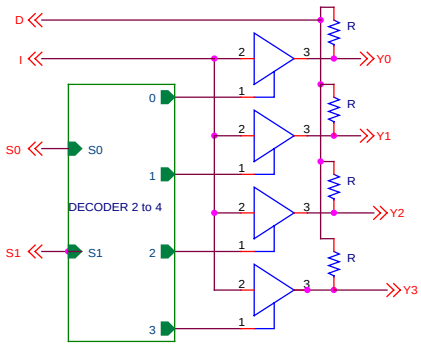


Fig. 6.25

Quando D = 0 i circuiti di Fig. 6.24 e Fig. 6.25 risultano del tutto equivalenti.

Se D = 1 otteniamo una logica di decodifica invertita cioè la linea selezionata è la sola ad essere nello stato logico 0.

I circuiti multiplexer e demultiplexer sono spesso usati in coppia quando si vuole realizzare un sistema in cui si desidera trasmettere molte linee di dati su di una sola linea (serializzazione dei dati) e quindi ritornare, dopo averli ricevuti, ai dati originali (parallelizzazione dei dati).

PROBLEMI

P6.1) Progettare un circuito combinatorio che, a partire da un numero BCD, piloti un display a sette segmenti per fornirne una rappresentazione numerica. Un display a sette segmenti è caratterizzato come rappresentato in figura.Un segmento è acceso se la linea che lo controlla è al valore logico 1 e spento

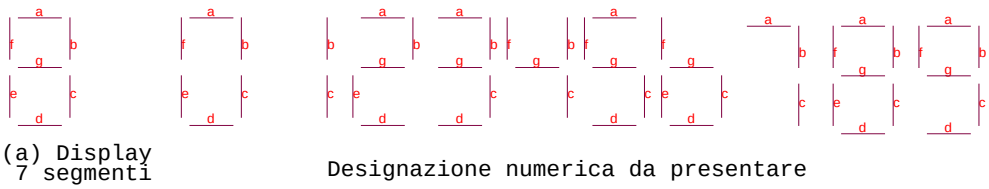


Fig. P6.1

viceversa.

a) Progettare il decoder a 7 segmenti usando tutte le condizioni di don't care
b) Fare vedere quale sarà l'uscita sul display quando in ingresso è fornito una configurazione non BCD
c) Riprogettare il decoder in modo tale che il display fornisce una rappresentazione non intelligibile per una combinazione di ingresso non BCD.
d) Disegnare lo schema del circuito trovato nel punto c).

P6.2) Progettare un circuito combinatorio con il minino numero di porte NOR che ha quattro variabili di ingresso che rappresentano una cifra BCD ed una sola funzione di uscita che è =1 se il valore della cifra di ingresso è minore di 3 e maggiore di 7.

P6.3) Progettare un circuito combinatorio che ha tre variabili di ingresso ed una sola funzione di uscita. Questa assume il valore 1 se la maggioranza degli ingressi sono ad 1, altrimenti prende il valore 0.

P6.4) Progettare un circuito combinatorio che faccia la conversione di un numero in codice BCD in uno in eccesso di 3. Si imponga la condizione che ad un codice non permesso BCD corrisponda un codice non permesso in eccesso di 3.

P6.5) Progettare un circuito combinatorio che faccia la conversione di un numero codificato in eccesso di 3 in uno in codice BCD. Si imponga la condizione che ad un codice non permesso in eccesso di 3 corrisponda un codice non permesso in BCD.

P6.6) Progettare un circuito combinatorio che faccia la conversione di un numero binario puro a 4 bit (8421) in un numero codificato GRAY.

P6.7) Progettare un circuito combinatorio che faccia la conversione di un numero a 4 bit, codificato in GRAY, in un numero binario puro a 4 bit (8421).

P6.8) Facendo tesoro di quanto ottenuto con la soluzione del problema P6.6) trovare una formula di conversione che vale per un numero n di bit.

P6.9) Facendo tesoro di quanto ottenuto con la soluzione del problema P6.7) trovare una formula di conversione che vale per un numero n di bit.

P6.10 Progettare un circuito Combinatorio che converte una cifra decimale dal codice 8,4,2,1 al codice 8,4,-2,-1 (e disegnarne lo schema logico).

P6.11 Progettare un circuito Combinatorio che converte una cifra decimale dal codice 2,4,2,1 al codice 8,4,-2,-1 (e disegnarne lo schema logico).

P6.12 Progettare un circuito Combinatorio che converte un numero binario a 4 bit, in codice 8421, ad un numero decimale nel codice BCD (e disegnarne lo schema logico). Notare che per rappresentare in BCD i numeri da 0 a 15 sono necessarie due cifre BCD.

P6.13 Realizzare la seguente funzione Booleana con un multiplexer digitale a 8 ingressi:

$Y(I,S_2,S_1,S_0) = m_2+m_4+m_7+m_{10}+m_{12}+m_{14}$

P6.14 Realizzare un circuito Full hadder con due multiplexer digitali a 4 ingressi.