Appunti di Elettronica Digitale
Circuiti per il codice Hamming

Circuito per la decodifica Hamming

Vogliamo realizzare un **circuito combinatorio** che rilevi e corregga l'errore singolo su parole di **codice Hamming**. In particolare, le specifiche sono le seguenti:

- il circuito deve ricevere le parole binarie in ingresso in parallelo, ossia con tutti i bit contemporaneamente: ciò significa che il circuito sarà di tipo combinatorio (non c'è bisogno di elementi di memoria) e dovrà avere tanti ingressi quanti sono i bit che compongono le parole del codice;
- consideriamo un codice binario con parole di lunghezza **n=8**: allora, se vogliamo passare da tale codice binario generico ad un codice di Hamming che consenta la rilevazione e la correzione dell'errore singolo, dovremo aggiungere altri k bit di parità, dove k deve soddisfare la relazione $2^k \geq n+k+1$; ponendo n=8 e risolvendo per tentativi, si trova k=4, per cui il codice che consideriamo è un codice **8B-12B**, cioè un codice con parole di lunghezza 12, dove 8 sono i bit di dati a 4 i **bit di parità**; tali bit di parità devono essere sistemati, nella generica parola da 12 bit, in posizioni corrispondenti a potenze di due: se numeriamo la posizioni da 1 a 12 partendo da sinistra, la generica parola che andrà in ingresso al circuito sarà allora nella forma

p_1	p_2	b_1	p_3	b_2	b_3	b_4	p_4	b_5	b_6	b_7	b_8
1	2	3	4	5	6	7	8	9	10	11	12

dove $b_1b_2b_3b_4b_5b_6b_7b_8$ è la parola di dati e $p_4p_3p_2p_1$ la parola binaria che è venuta fuori, in trasmissione, a seguito dell'esecuzione delle 4 prove di parità;

- infine, il criterio con cui si assume che siano stati impostati i bit di parità è quello di ottenere, nelle prove di parità, una **parità pari**, ossia un numero pari di bit ad 1 in ciascun gruppo di bit considerato.

Quindi, il circuito sarà una “scatoletta” con 12 ingressi (corrispondenti alla parola binaria su cui indagare) e 8 uscite (corrispondenti alla parola binaria corretta).

La prima cosa che il circuito deve compiere sono le **4 prove di parità**:

1) il risultato è 1 se i bit di posizione 1,3,5,7,9,11 contengono un numero dispari di 1 (cioè se è stata violata la parità pari), mentre il risultato è 0 in caso contrario;
2) il risultato è 1 se i bit di posizione 2,3,6,7,10,11 contengono un numero dispari di 1, è 0 in caso contrario;
3) il risultato è 1 se i bit di posizione 4,5,6,7,12 contengono un numero dispari di 1, mentre è 0 in caso contrario;
4) il risultato è 1 se i bit di posizione 8,9,10,11,12 contengono un numero dispari di 1, mentre è 0 in caso contrario.

I risultati delle 4 prove, posti nell’ordine $r_4r_3r_2r_1$, danno, in binario, la posizione dell’eventuale bit sbagliato: se le prove hanno dato tutte esito negativo, ossia se $r_4r_3r_2r_1 = 0000$, significa che non ci sono stati errori. Se, invece, ad esempio, è risultato $r_4r_3r_2r_1 = 0110$, allora si è verificato un errore sul bit in posizione 6, che quindi può essere corretto.

A livello implementativo, si può procedere nel modo seguente:

- la generica prova di parità corrisponde a prendere un certo numero di bit ed a contare il numero di 1 presenti: se tale numero è dispari, la prova ha risultato r=1, altrimenti è r=0. Allora, si può realizzare questo mediante la funzione **EXOR**: infatti, l’EXOR di un numero k di ingressi fornisce 1 in uscita se ci sono un numero dispari di 1 in ingresso, altrimenti fornisce 0. Quindi, le 4 prove di parità si possono effettuare o usando porte logiche EXOR a più ingressi oppure varie porte EXOR a 2 ingressi in cascata;

- una volta effettuate le 4 prove di parità, i risultati costituiscono la parola binaria $r_4r_3r_2r_1$ che individua l'eventuale bit sbagliato: allora, dobbiamo usare un circuito che, ricevendo in ingresso la parola $r_4r_3r_2r_1$, attivi, ponendo ad 1, una precisa linea di uscita: questo è quello che fa un **DECODER**. Nel nostro caso, avendo 4 ingressi, il decoder dovrà essere 4x16: delle 16 linee di uscita, però, non ci interessano tutte, ma solo quelle che corrispondono alle posizioni, nella parola di ingresso, cui corrispondono bit di informazione;
- dobbiamo dunque capire come si possa effettuare la correzione, tramite il decoder, qualora sia stato individuato un errore: supponiamo, per esempio, che i risultati delle prove di parità siano stati $r_4r_3r_2r_1 = 0110$, indicando cioè un errore nella 6° posizione (corrispondente a b_3); il decoder riceve in ingresso la parola $r_4r_3r_2r_1 = 0110$ e attiva, ponendo ad 1, la linea I_6, lasciando le altre linee a 0; il valore I_6=1 deve quindi indicare che il bit in posizione 6, cioè b_3, va modificato: ciò significa che dobbiamo realizzare una funzione che, leggendo il valore di I_6, complementi il valore di b_3; la tabella della verità di questa funzione è quindi la seguente:

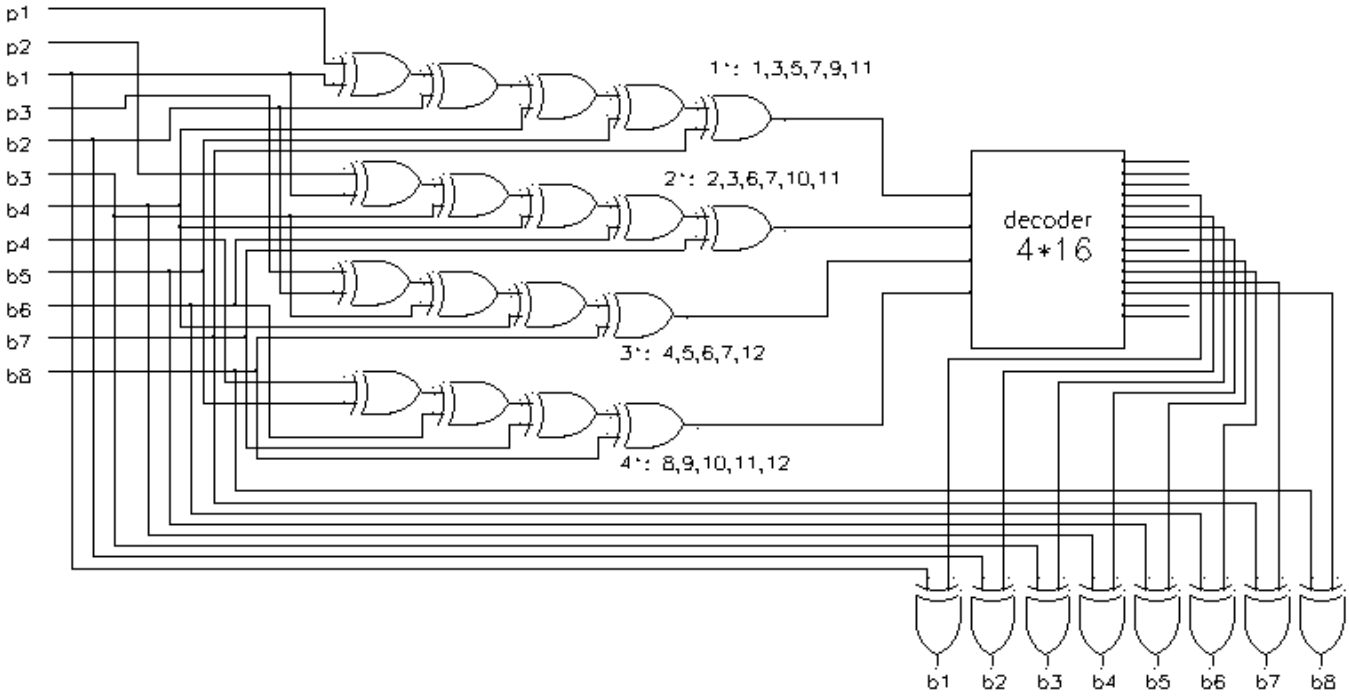
b_3	I_6	uscita
0	0	0
0	1	1
1	0	1
1	1	0

$\longrightarrow \text{uscita} = b_3 \oplus I_6$

Come si nota, la funzione da realizzare è semplicemente un altro EXOR, tra il bit da modificare e la corrispondente linea di uscita del decoder: se la linea di uscita vale 0, il bit rimane invariato, altrimenti viene complementato.

Identico discorso va fatto ovviamente per gli altri 7 bit: ad esempio, per il bit in posizione 3, cioè b_1, dovremo fare l'EXOR tra b_1 e la linea I_3 del decoder, che è quella indirizzata quando è rilevato un errore in posizione 3.

Seguendo questi criteri, il circuito che realizza i compiti voluti è fatto nel modo seguente:



Come si nota, non tutte le linee del decoder sono utilizzate, ma solo quelle che corrispondono a posizioni, nella parola di ingresso, in cui si trovano bit di dati: si tratta cioè delle linee 3,5,6,7,9,10,11 e 12. La linea 0 non è usata per il semplice fatto che, se dopo le prove di parità dovesse risultare $r_4 r_3 r_2 r_1 = 0000$, non ci sarebbe nessun bit da modificare.

Osserviamo che, in questo circuito, dati i **ritardi di propagazione** dei segnali nelle singole parti, ci vuole un certo tempo affinché l'uscita sia quella corretta: in altre parole, se t=0 è l'istante in cui viene applicato l'ingresso al circuito, ci vorrà un certo tempo T affinché l'uscita sia la parola binaria eventualmente corretta.

Circuito per la codifica Hamming

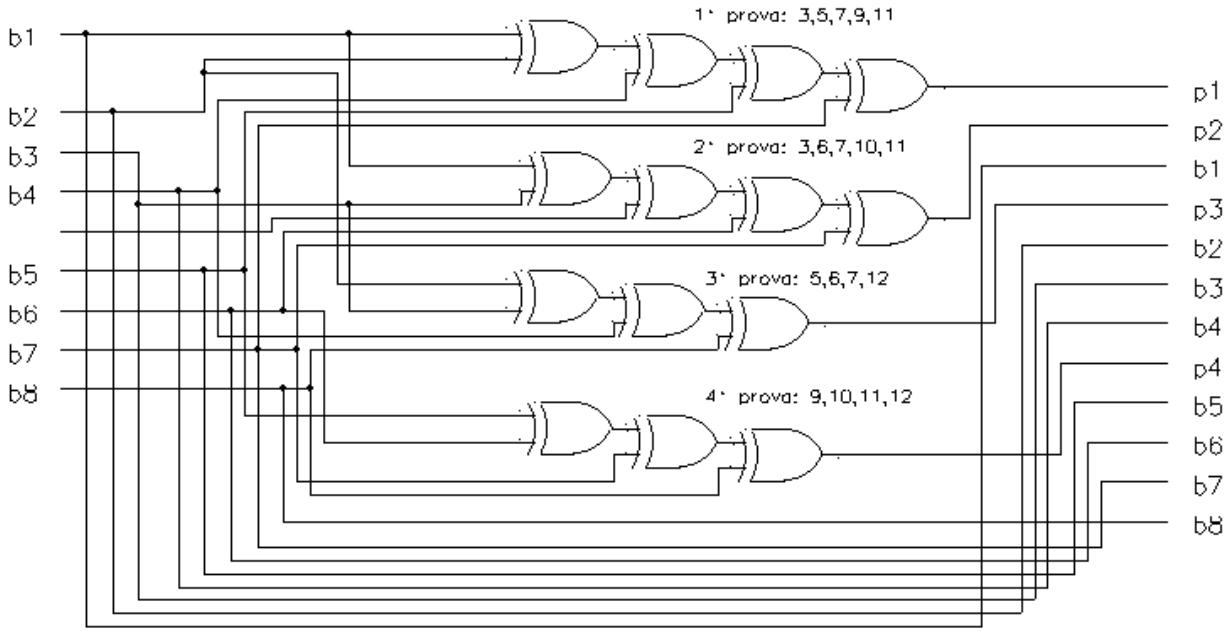
Adesso supponiamo di voler realizzare il circuito che effettui l'operazione inversa rispetto a quella descritta: esso cioè deve ricevere gli 8 bit del codice numerico in esame e deve generare in uscita la parola di 12 bit ottenuta aggiungendo agli 8 bit di dati i 4 bit di parità. E' evidente che si tratta di un circuito molto semplice, in quanto tutto sta ad effettuare le prove di parità e ad impostare le uscite in base ai risultati di tali prove. In questo caso, le prove di parità devono ovviamente riguardare solo i bit di dati Consideriamo allora nuovamente il modo con cui viene costruita la parola di uscita:

p_1	p_2	b_1	p_3	b_2	b_3	b_4	p_4	b_5	b_6	b_7	b_8
1	2	3	4	5	6	7	8	9	10	11	12

Le prove di parità sono le seguenti:

1) bisogna porre $p_1=1$ se i bit in posizione 3,5,7,9,11 contengono un numero dispari di 1 (in modo da garantire la parità pari), altrimenti bisogna porre $p_1=0$;
2) bisogna porre $p_2=1$ se i bit in posizione 3,6,7,10,11 contengono un numero dispari di 1 (in modo da garantire la parità pari), altrimenti $p_2=0$;
3) bisogna porre $p_3=1$ se i bit in posizione 5,6,7,12 contengono un numero dispari di 1 (in modo da garantire la parità pari), altrimenti $p_3=0$;
4) bisogna porre $p_4=1$ se i bit in posizione 9,10,11,12 contengono un numero dispari di 1 (in modo da garantire la parità pari), altrimenti $p_4=0$.

Ancora una volta, basta usare delle porte logiche EXOR in cascata, che questa volta serviranno ad impostare direttamente i bit di parità e non più a pilotare un decoder:



Autore: **Sandro Petrizzelli**

e-mail: sandry@iol.it

sito personale: http://users.iol.it/sandry