

Esercizio d'esame (appello straordinario Marzo 2021)

Segnale di durata complessiva 32 μs
Il segnale ha l'andamento seguente:

- a) $x(t) = d \quad 0 \leq t \leq 8 \mu s$
- b) $x(t) = -d \quad 8 \leq t < 16 \mu s$
- c) $x(t) = d \quad 16 \leq t < 24 \mu s$
- d) $x(t) = -d \quad 24 \leq t < 32 \mu s$

1) Sapendo che la codifica di linea associa il bit 1 ad un livello alto e il bit 0 a un livello basso, è possibile decodificare il segnale dato, ricevendo la corrispondente sequenza di bit?

NO perché manca l'intervallo di simbolo

2) Ritmo binario del trasmettitore ha 250 kbit/s. Scrivete la sequenza binaria associata al segnale.

$$\text{Intervallo di bit} = \frac{1 \text{ bit}}{250 \times 10^3 \text{ bit/s}} = 4 \mu s \xrightarrow{\text{un bit ogni } 4 \mu s}$$

- a) $x(t) = d \quad 0 \leq t \leq 8 \mu s \rightarrow 11$
- b) $x(t) = -d \quad 8 \leq t < 16 \mu s \rightarrow 00$
- c) $x(t) = d \quad 16 \leq t < 24 \mu s \rightarrow 11$
- d) $x(t) = -d \quad 24 \leq t < 32 \mu s \rightarrow 00$

3) Ripetere con 500 kbit/s

$$\text{Intervallo di bit} = \frac{1 \text{ bit}}{500 \times 10^3 \text{ bit/s}} = 2 \mu s$$

11110000 11110000

4)

How good is the single parity check code?

- Redundancy: Single parity check code adds 1 redundant bit per k information bits: overhead = $1/(k+1)$
- Coverage: all error patterns with odd # of errors can be detected
 - An error pattern is a binary $(k+1)$ -tuple with 1s where errors occur and 0s elsewhere
 - Of 2^{k+1} binary $(k+1)$ -tuples, $1/2$ have odd weight, so 50% of error patterns can be detected
- Is it possible to detect more errors if we add more check bits? Yes, with the right codes

Esempio:

$$k = 7 \quad n = k + 1 = 8 \quad y \in C \xleftarrow{\delta^1} \text{No}$$

* errori | Esito ricezione | ✗ Probabilità

0	✓	$(1-p)^8$
1	✓ (x)	$\binom{8}{1} p(1-p)^7$
2	✗	$\binom{8}{2} p^2(1-p)^6$
3	✓ (x)	$\binom{8}{3} p^3(1-p)^5$
4	✗	$\binom{8}{4} p^4(1-p)^4$
5	✓ (x)	$\binom{8}{5} p^5(1-p)^3$
6	✗	$\binom{8}{6} p^6(1-p)^2$
7	✓ (x)	$\binom{8}{7} p^7(1-p)^1$
8	✗	$\binom{8}{8} p^8(1-p)^0$

probabilità di ricevere 8 bit senza errori

Con il single parity check si accorge solo degli errori dispari.

$$p \leq \frac{1}{2}$$

Non stiamo massimizzando la prestazione del codice.

probabilisticamente gli errori hanno diversa rilevanza

Un bit può essere effetto d'errore con probabilità p ($= \text{BER} = \frac{n_{bit errati}}{n_{Rx}}$)] Aggiungiamo la prob. che un errore è presente ✗

Errori binari statisticamente indipendenti

Una volta completata la tabella sappiamo calcolare la probabilità che il codice fallisca.

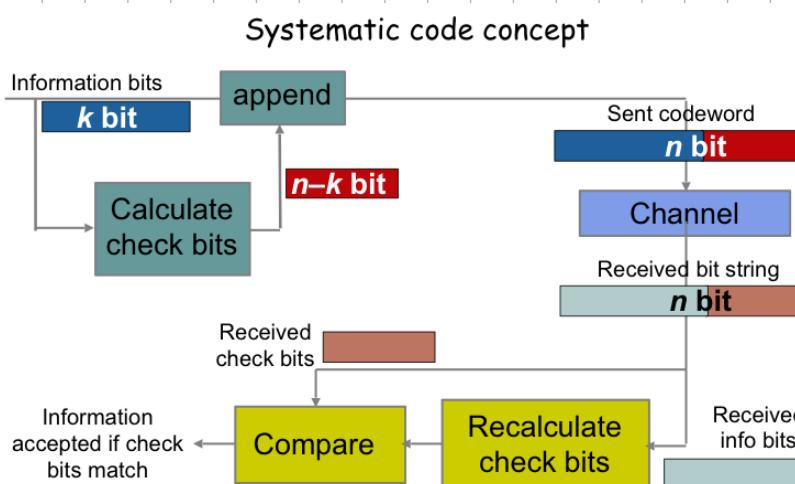
$$P(\text{codice fallisca}) = \binom{8}{2} p^2(1-p)^2 + \binom{8}{4} p^4(1-p)^4 + \binom{8}{6} p^6(1-p)^2 + p^8$$

$$\text{sapendo che } p \ll 1 \approx \binom{8}{2} p^2(1-p)^6 \approx 28 p^2$$

Se uso il codice

$$\begin{aligned} P(\text{Rx ok}) &= (1-p)^8 \\ P(\text{errore}) &= 1 - (1-p)^8 \approx 7 \cdot 10^{-3} \end{aligned} \quad \rightarrow \text{senza codice}$$

Muovendoci su codici più complessi, analizzeremo i codici sistematici. Il parity check rientra in questi.



La sorgente ha k bit di dati che vuole trasmettere. In base al codice c'è un algoritmo che dati questi k bit calcola $n-k$ bit, nel semplice codice di parità questo algoritmo è la somma. Questi $n-k$ bit vengono aggiunti ai k formando una parola di codice lunga n bit. Questa parola è fatta: i primi k bit sono copiati dai bit dei dati, gli ultimi sono quelli calcolati. I codici che fanno così vengono chiamati sistematici.

La parola di codice viene trasmessa nel canale binario, il ricevitore prende i primi k bit che ha ricevuto, ricalcola la ridondanza come fa il trasmettitore (con lo stesso algoritmo) e ottiene gli $n-k$ bit. Poi prende gli $n-k$ bit che ricevuto prima e confronta e vede se coincidono sulla base dei dati ricevuti. Se si, tutto ok, se no, ci sono errori.

Questa codifica si può rendere più efficiente, lavorando sulla struttura particolare del codice.

Polynomial Codes

Per descrivere questi codici facciamo una premessa, sono progettati utilizzando l'algebra dei polinomi.

Introduciamo i polinomi di una stringa binaria.

Data una stringa binaria posso associare sempre un polinomio utilizzando i bit come i coefficienti del polinomio.

- Binary vectors map to polynomials

$$(b_{k-1}, b_{k-2}, \dots, b_2, b_1, b_0) \rightarrow b_{k-1}x^{k-1} + b_{k-2}x^{k-2} + \dots + b_2x^2 + b_1x + b_0$$

Successivamente si opera sui coefficienti con prodotti e somme:

\oplus	EXOR	0	1	AND	0	1
SOMMA	0	0	1	0	0	0
	1	1	0	1	0	1

prodotto

Esempio:

$$1) (x+1)(x^2+1) = x^3 + x + x^2 + 1 = x^3 + x^2 + x + 1$$

$$2) (x-1)(x+1) = x^2 + \underbrace{x + x}_{1 \oplus 1 = 0} + 1 = x^2 + 1$$

$$3) \begin{array}{r} x^3 + x + 1 \\ \times \quad x + 1 \\ \hline x^3 \quad 0 \quad 0 \end{array}$$

gli unici coefficienti possibili sono 1 e 0

$$\begin{array}{r} x^6 + x^5 \\ \times \quad x^3 \\ \hline 0 \quad x^5 \quad x^4 \quad x^3 \\ \quad x^5 \quad x^3 \quad x^2 \\ \hline 0 \quad x^6 \quad 0 \quad x^2 \\ \quad x^4 \quad x^2 \quad x \\ \hline 0 \quad 0 \quad x \end{array}$$

$$\begin{array}{r} x^3 + x + 1 \\ \times \quad x^3 \\ \hline x^6 + x^5 + x^4 \end{array}$$

$$x^6 + x^5 = q(x) \cdot (x^3 + x + 1) \cdot r(x)$$

$$q(x) = x^3 + x^2 + 1$$

$$r(x) = x$$

Si parte da k bit di dati che vanno immaginati come un polinomio $b(x)$.

Assegnare un polinomio ad un codice binario, viene chiamato un polinomio generatore $g(x)$ di grado $n-k$.

Poi si fa questa divisione:

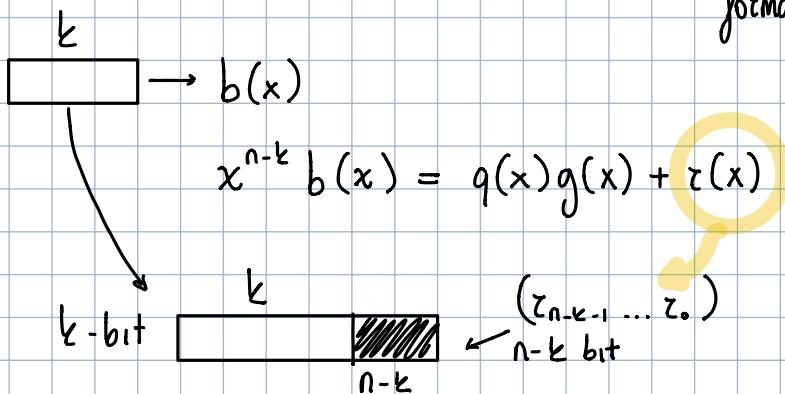
$$x^{n-k} b(x) = q(x) g(x) + r(x)$$

ha grado al massimo $n-k-1$

$r(x)$ grado $\leq n-k-1$

$r(x) = r_{n-k-1} x^{n-k-1} + \dots + r_1 x + r_0$ * quanti bit ci sono in questo polinomio?

* bit associati a $r(x) \rightarrow r(x) = n-k$ questi sono esattamente la ridondanza che aggiungeremo ai bit di dati per formare la parola di codice.



Esempio

$$x^6 + x^5 : x^3 + x + 1 \quad r(x) = x$$

$$g(x) = x^3 + x + 1 \xrightarrow{\text{polin. generatore}} n-k = 3$$

Vogliamo trasmettere:

$$b = [1 \ 1 \ 0 \ 0] \quad k = 4$$

$$b(x) = x^3 + x^2 + 0x^1 + 0 \cdot x^0 = x^3 + x^2$$

$$x^{n-k} b(x) = x^3(x^3 + x^2) = x^6 + x^5 = q(x)g(x) + r(x)$$

$$r(x) = x \quad (\text{check: deve essere polinomio di grado } < n-k = 3)$$

$$* r_2 x^2 + r_1 x + r_0$$

$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} r_2 & r_1 & r_0 \end{bmatrix} \quad \rightarrow \underline{s} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

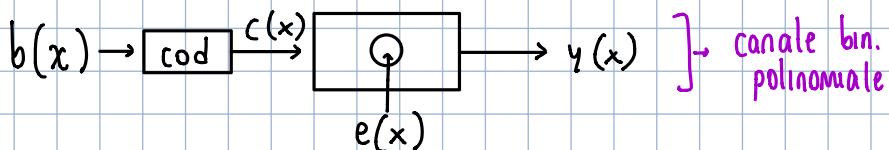
\uparrow
ridondanza

$k=4 \quad n-k=3$

$$c = [b_{k-1} \dots b_0 \quad z_{n-k-1} \dots z_0]$$

b bit (dati) z bit (ridondanza)

$$c(x) = x^{n-k} b(x) + z(x)$$



Adesso bisogna trovare un modo per capire se $y(x)$ è una parola di codice o no.

La parola di codice si ottiene:

$$c(x) = x^{n-k} b(x) + z(x)$$

$$y(x) = c(x) + e(x)$$

$y(x)$ è una parola di codice?

La proprietà che caratterizza le parole di codice, visto che sono un particolare insieme di polinomi, sostituiamo:

$$x^{n-k} b(x) = q(x)g(x) + z(x)$$

(1+1=0)

$$c(x) = x^{n-k} b(x) + z(x) = q(x)g(x) + z(x) + z(x) = q(x)g(x)$$

Quindi le parole di codice sono quelle per cui il resto è 0. Sono quei polinomi particolari che sono multipli di $g(x)$.

Quindi $y(x)$ è una parola di codice se dividendolo per $g(x)$ il resto viene 0.

accetta i dati

$$\text{Parola di codice} \iff \text{Resto } \frac{y(x)}{g(x)} = 0 \quad (\text{se } e \neq 0 \Rightarrow \text{erri})$$

Tuttavia se $R(x) \neq 0$, posso comunque sbagliare?

$$y(x) = c(x) + e(x)$$

$$\frac{y(x)}{g(x)} = \frac{q(x)g(x)}{g(x)} + \frac{e(x)}{g(x)}$$

$$\text{Resto} \left(\frac{y(x)}{g(x)} \right) = \text{Resto} \left(\frac{e(x)}{g(x)} \right) = \begin{cases} 0 & e(x) = 0 \\ \neq 0 & e(x) \neq 0 \\ = 0 & e(x) \neq 0 \end{cases}$$

ma $e(x) = \tilde{q}(x)g(x)$ ossia gli errori sono multipli di $g(x)$

Allora in ricezione operiamo coh., c'è fallimento $\Leftrightarrow e(x) = \tilde{q}(x)g(x)$

$g(x)$ si sceglie in m.t.c la prob. che $e(x)$ ha uno multiplo è molto bassa.

Standard Generator Polynomials

CRC = cyclic redundancy check

- $CRC-8: = x^8 + x^2 + x + 1$ ATM
- $CRC-16: = x^{16} + x^{15} + x^2 + 1$ Bisync
 $= (x + 1)(x^{15} + x + 1)$
- $CCITT-16: = x^{16} + x^{12} + x^5 + 1$ HDLC, XMODEM, V.41
- $CCITT-32: = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11}$
 $+ x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ IEEE 802, DoD, V.42