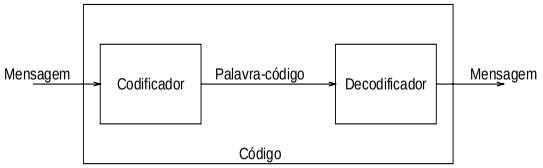
Capítulo 8

Códigos Binários

Os sistemas digitais só operam com dois dígitos: 0 e 1. Por causa da diversidade das aplicações, muitos códigos foram propostos para representar diferentes tipos de informação. Esta informação pode ser números (em diferentes sistemas de numeração), letras (p. ex., a, A, w, E), símbolos (p. ex., *, #, $) e sinais de controle (p. ex., CTRL-C, "mudança de linha").

De uma maneira geral, podemos dizer que um **código binário** $\mathcal{C}$ é um conjunto de sequências de comprimento n (**palavras-código**) sobre $GF(2)$, associado a esquemas de **codificação** e **decodificação**:

- num esquema de codificação cada unidade de informação é mapeada univocamente a uma **palavra-código**.
- num esquema de decodificação cada palavra-código é convertida na unidade de informação original.

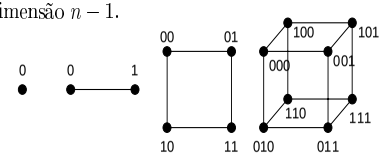


No capítulo 6 já tivemos oportunidade de ver alguns circuitos simples de codificação e decodificação.

Neste capítulo introduziremos algumas noções básicas, mas mais formais, sobre códigos.

8.1 Conceitos Básicos

Podemos representar graficamente uma palavra binária de n dígitos como vértices de um cubo unitário de dimensão n no espaço R^n, de modo que um cubo de dimensão 0 é um ponto e a projeção de um cubo de dimensão n é um cubo de dimensão $n-1$.



A distância, em termos de arestas, entre dois vértices é denominada **distância de Hamming** H. Como por construção, dois vértices adjacentes (por arestas) diferem só de um bit, é comum definir a distância de Hamming entre duas palavras binárias como a quantidade de bits que elas diferem.

Exemplo 8.1 *Dadas as palavras: a=00110, b=11001 e c=00101. A distância de Hamming H(a,b)=5, H(d,c)=4 e H(a,c)=2.*

Considere um código $\mathcal{C}$. A **distância de Hamming** d do código é a mínima distância e Hamming H entre todos os pares de palavras pertencentes ao código.

Exemplo 8.2 *Dados dois códigos* $\mathcal{C}_1 = \{00, 01, 10, 11\}$ *e* $\mathcal{C}_2 = \{000, 111\}$. *A distância de Hamming destes códigos é, respectivamente igual a 1 e 3.*

	00	*01*	*10*	*11*
00	-	*1*	*1*	*2*
01	*1*	-	*2*	*1*
10	*1*	*2*	-	*1*
11	*2*	*1*	*1*	-

d=1

	000	*111*
000	-	*3*
111	*3*	-

d=3

Observe que $\mathcal{C}_2$ *é conhecido como* **código de repetição**.

O **peso** de uma palavra binária a, $W(a)$, é a quantidade de bits 1 que aparecem na palavra.

Exemplo 8.3 *Sejam as palavras a=0000, b=0110 e c=1011, então W(a)=0, W(b)=2 e W(c)=3.*

Pode-se obter a distância de Hamming entre duas palavras binárias, a e b, através da expressão

$$H(a,b) = W(a \oplus b),$$

onde $\oplus$ denota a operação XOR, bit a bit, entre as duas palavras.

Exemplo 8.4 *Sejam duas palavras a=1001 e b=0011.* $H(a,b) = W(a \oplus b) = W(10100) = 2$.

Seja d a distância mínima de um código. Se $d = 1$, então o código não pode detectar todos os erros. Se $d = 2$, então o código **pode detectar, mas não corrigir, erros simples** (quando só há um bit trocado na palavra). E se $d \geq 3$, então o código pode **detectar e corrigir todos os erros simples**.

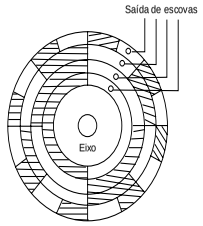
Um princípio bastante intuitivo para deteção e correção de erros introduzidos num código original é o princípio de **máxima verossimilhança** (*maximum likelihood*), o qual considera como **a mais provável palavra válida** aquela que tiver a menor distância de Hamming em relção à palavra em análise.

Exemplo 8.5 *Dado o código* $\mathcal{C} = \{00000, 01110, 10111, 11001\}$ *e uma palavra* $r = 11111$. *Por comparações exaustivas, pode-se concluir que r não pertence a* $\mathcal{C}$*, portanto ela não é válida e, pela similaridade, a mais provável palavra é 10111, pois H(r,00000)=5, H(r,01110)=2, H(r,10111)=1 e H(r,11001)=2.*

Exemplo 8.6 *Dado o código* $\mathcal{C} = \{000, 011, 101, 110\}$ *e uma palavra* $r = 100$. *Por comparações exaustivas, pode-se concluir que r não pertence a* $\mathcal{C}$*, portanto ela não é válida e, pela similaridade, não se pode inferir a provável palavra original, pois H(r,000)=1, H(r,011)=2, H(r,101)=1 e H(r,110)=1.*

8.2 Códigos de Gray

O primeiro codificador é um codificador angular que converte uma posição do eixo mecânico num número digital codificado em binário. Este codificador consiste em um disco de segmentos de material condutor (áreas escurecidas) e de material isolante (áreas claras) acoplado a um eixo. Um conjunto de escovas é então ligado de modo que uma só escova é posicionada no centro de cada faixa concêntrica do disco. Se a escova estiver sobre a área escurecida resultará num sinal 1 e se estiver sobre a área clara, num sinal 0.



Há um problema crítico com este esquema de codificação: se o disco estiver numa posição em que o número de saída varie, por exemplo, de 0011 a 0100, ou em qualquer posição onde vários bits mudam de valor simultaneamente, o sinal de saída pode ser ambíguo devido às imprecisões mecânicas (a probabilidade das escovas cobrirem simultaneamente os novos segmentos é muito pequena). Para evitar esta ambigüidade foi proposto o uso de **códigos de Gray**, no qual dois quaisquer números adjacentes diferem somente de 1 bit.

Decimal	Código de Gray (3 bits)	Código de Gray (4 bits)
0	000	0000
1	001	0001
2	011	0011
3	010	0010
4	110	0110
5	111	0111
6	101	0101
7	100	0100
8		1100
⋮	⋮	⋮

O código de Gray pertence à classe de **códigos refletidos**, pois observe na tabela acima que as palavras-código de n bits podem ser geradas através da reflexão das palavras-código de $n-1$ bits. Como já vimos no capítulo 6, os codificadores e decodificadores destes códigos são circuitos combinacionais, nos quais cada bit de saída é uma função lógica de todos os bits de entrada (números em representação binária).

8.3 Códigos BCD

Para facilitar a interpretação dos números representados no sistema binário, foram porpostos os códigos BCD (*binary-coded decimal*). Este códigos são também conhecidos como **códigos ponderados**, pois cada dígito decimal é representado por um grupo de 4 bits e cada bit tem um peso associado.

A classe de códigos BCD mais conhecida é a classe dos códigos **8421**, no qual o dígito mais significativo tem o peso 8, o segundo dígito 4, o terceiro 2, e o dígito menos significativo o peso 1.

Decimal	Código 8421 BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Exemplo 8.7 *Os seguintes números* (245), $(16,72)$ *e* $(0,398)$ *são representados, respectivamente, pelas palavras-código* (001001000101), $(00010110, 01110010)$ *e* $(0000, 001110011000)$ *no código 8421 BCD.*

Na tabela que se segue são apresentados 4 códigos BCD com outras ponderações:

Binário	4221 BCD	5421 BCD	2421 BCD	84-2-1 BCD
0000	0000	0000	0000	0000
0001	0001	0001	0001	0111
0010	0010	0010	0010	0110
0011	0011	0011	0011	0101
0100	1000	0100	0100	0100
0101	0111	1000	1011	1011
0110	1100	1001	1100	1010
0111	1101	1010	1101	1001
1000	1110	1011	1110	1000
1001	1111	1100	1111	1111

Observe que a partir desta tabela é fácil sintetizar os codificadores e decodificadores destes códigos em circuitos combinacionais.

8.4 Códigos XS3

Como nos códigos BCD, nos códigos XS3 ou **códigos 3 em excesso** cada dígito decimal é também representado por um conjunto de 4 bits. Entretanto, diferentemente dos códigos BCD, estes códigos não são considerados códigos ponderados, pois a cada bit não é associado um peso específico, como mostra a seguinte tabela

Decimal	Código 3 em excesso
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Observe que ele é obtido a partir do código BCD 8421, adicionando 011 a cada palavra-código correspondente.

O código XS3 facilita a implementação de operações aritméticas com uso de complementos. Este código, como código BCD 4221, é um *self-complementing code*. Um *self-complementing code* é aquele para o qual o complemento de 9 do número correspondente pode ser facilmente obtido complementando bit a bit (invertendo cada bit).

Exemplo 8.8 *Seja um número em decimal* $(8315)_{10}$. *A sua presentação em XS3 é* $(1011011001001000)_{XS3}$. *O compelemnto, bit a bit, do número é* $(0100100110110111)_{XS3}$ *que em XS3 corresponde a* $(1684)_{10}$. *Este é exatamente o complemento de 9 de* $(8315)_{10}$.

8.5 Códigos Alfanuméricos

Códigos alfanuméricos são códigos capazes de representar tanto as letras quanto os números e os caracteres de controle. Foram feitas várias tentativas no sentido de padronizar um código alfanumérico que satisfizesse os interesses dos fabricantes e dos usuários.

O código mais utilizado nos sistemas computacionais é o código ASCII (*American Standard Code for Information Interchange*) proposto pela ANSI (*American National Standards Institute*). Ele é considerado o formato-padrão para a entrada/saída nos microcomputadores. Neste código as palavras-código são formadas por 7 bits. Os 4 últimos bits das plavras-código que representam os dígitos decimais são exatamente iguais às palavras-código correspondentes no código 8421 BCD.

Bits	Bits 654							
3210	000	001	010	011	100	101	110	111
0000	NUL	DLE	SPACE	0	@	P	’	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	‘‘	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L		l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N		n	∼
1111	SI	US	/	?	O	-	o	DEL

Outros dois códigos conhecidos, utilizados pela IBM, são BCDIC (*binary-coded decimal interchange code*) e EBCDIC (*extended binary-coded decimal interchange code*). No primeiro, os caracteres alfanuméricos são representados pelas palavras de 7 bits e no segundo, 8 bits.

8.6 Códigos Detectores e Corretores de Erro

O processo da transferência de informação entre os dispositivos digitais é bastante susceptível a erros. Para aumentar a confiabilidade de um sistema, é desejável detectar e, se possível, corrigi-los. Para isso, é necessário introduzir r bits de redundâncias na informação (codificada) real de k bits. Tais bits de redundância são denominados **bits de paridade** e usamos a notação (n,k), onde $n = k + r$ é a quantidade total dos bits nas palavras-código.

O código detector de erro mais conhecido é o **código de paridade**. Neste código adiciona um bit de paridade aos k bits da informação original, de tal forma que $b_k \oplus b_{k-1} \oplus \cdots \oplus b_2 \oplus b_1 \oplus b_0 \oplus p = 0$ (paridade par) ou $b_k \oplus b_{k-1} \oplus \cdots \oplus b_2 \oplus b_1 \oplus b_0 \oplus p = 1$ (paridade ímpar). Chamamo o resultado destas expressões de **síndrome** da palavra-código.

Exemplo 8.9 *Seja um código de Gray de 3 bits ($b_2b_1b_0$). O código de paridade correspondente é listado na seguinte tabela.*

Gray	*Paridade Par*	*Paridade Ímpar*
000	*000 0*	*000 1*
001	*001 1*	*001 0*
011	*011 0*	*011 1*
010	*010 1*	*010 0*
110	*110 0*	*110 1*
111	*111 1*	*111 0*
101	*101 0*	*101 1*
100	*100 1*	*100 0*

Note que a distância de Hamming do código de paridade é 2. No caso da paridade par, a distância é também 2.

	0000	*0011*	*0110*	*0101*	*1100*
0000	-	*2*	*2*	*2*	*2*
0011	*2*	-	*2*	*2*	*4*
0110	*2*	*2*	-	*2*	*2*
0101	*2*	*2*	*2*	-	*2*
1100	*2*	*4*	*2*	*2*	-

d=2

Portanto, **códigos de paridade é um código detector de erros simples**. *Supomos que o segundo bit de uma palavra for invertido por algum motivo, então no lugar de 0011 leremos 0111 cuja síndrome é $0 \oplus 1 \oplus 1 \oplus 1 = 1 \neq 0$. Por essa síndrome, podemos dizer que ocorreu um erro na palavra.*

Não podemos, entretanto, corrigí-lo, pois $d(0111,0011) = d(0111,0101) = 1$. *Qual das duas é a original?*

Um **código corretor de erro simples** muito utilizado nos sistemas computacionais é o código de Hamming (7,4)[1]. Neste código são adicionados três bits de paridade (p_0, p_1, p_2) aos 4 bits originais de informação (i_0, i_1, i_2, i_3) para corrigir **erros simples** que possam ocorrer numa palavra $(p_0, p_1, i_0, p_2, i_1, i_2, i_3)$. Os valores dos três bits de paridade são tais que devem satisfazer as seguintes equações:

$$\begin{aligned} p_2 \oplus i_1 \oplus i_2 \oplus i_3 &= 0 \\ p_1 \oplus i_0 \oplus i_2 \oplus i_3 &= 0 \\ p_0 \oplus i_0 \oplus i_1 \oplus i_3 &= 0 \end{aligned} \tag{8.1}$$

Ou seja,

$$\begin{aligned} p_2 &= i_1 \oplus i_2 \oplus i_3 \\ p_1 &= i_0 \oplus i_2 \oplus i_3 \\ p_0 &= i_0 \oplus i_1 \oplus i_3 \end{aligned} \tag{8.2}$$

Assim cada mensagem $\vec{i} = (i_0, i_1, i_2, i_3)$ é convertida numa palavra-código $(i_0 \oplus i_1 \oplus i_3, i_0 \oplus i_2 \oplus i_3, i_0, i_1 \oplus i_2 \oplus i_3, i_1, i_2, i_3)$.

Utilizando a notação matricial temos

$$c = \begin{bmatrix} i_0 & i_1 & i_2 & i_3 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} = \vec{i}G$$

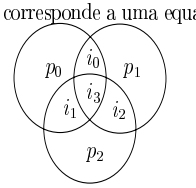
A matriz G é denominada **matriz geradora** deste código.

Os valores que ficam no lado direito da igualdade das eq.(8.1) formam um vetor que denominamos **síndrome** (de erro) de uma palavra. Pois, para qualquer palavra de 7 bits que satisfaz tais iguldades (vetor nulo), consideramos que ela pertence ao código. Caso contrário, dizemos que ela é válida para o tal código (vetor não nulo). Se supomos que só um e somente um

[1] Códigos de Hamming (n,k) apresentam as seguintes propriedades: distância de Hamming igual a 3, $n = 2^m - 1$, do qual $k = 2^m - m - 1$ são bits de informação e $n - k$ são bits de paridade

bit da palavra-código pode ser invertido (erro simples), podemos detectar e corrigir este erro, uma vez que a distância de Hamming dese código é 3.

O seguinte diagrama de Venn ajuda visualizar a relação entre os bits de cada palavra. Cada círculo corresponde a uma equação do sistema (8.1).



Observe que quando um dos bits de informação i_0, i_1, i_2 for invertido, duas das expressões do (8.1) terão resultados diferentes de 0. Se o bit de informação i_3 for invertido, as três equações terão resultados diferentes de 0. E, finalmente, se um bit de paridade for invertido, somente uma equação tem resultado diferente de zero. A partir dessa observação podemos estabelecer a seguinte relação entre cada síndrome $(p_2 p_1 p_0)$ e os padrões de erro

síndrome	padrão de erro
000	0000000
100	0001000
010	0100000
001	1000000
110	0000010
101	0000100
011	0010000
111	0000001

Exemplo 8.10 *Seja um código de Hamming (7,4) cujas palavras-código tem o formato* $\vec{c} = (p_0, p_1, i_0, p_2, i_1, i_2, i_3)$. *Considere que seja recebida a palavra* 0001001 *e que no sistema só ocorre erros simples. A palavra pertence ao código?*

A síndrome desta palavra é 011, pois

$$\begin{aligned} 1 \oplus 0 \oplus 0 \oplus 1 &= 0 \\ 0 \oplus 0 \oplus 0 \oplus 1 &= 1 \\ 0 \oplus 0 \oplus 0 \oplus 1 &= 1 \end{aligned}$$

Portanto, ela não é válida. Consultando a tabela de relação entre síndromes e padrões de erro, conclui-se que o padrão de erro correspondente é 0010000.

Assim, a palavra original deveria ter sido $0001001 \oplus 0010000 = 0011001$ *e não* 0001001.

É comum utilizar a notação matricial para representar a síndrome em função de palavras-código

$$s = \begin{bmatrix} p_0 & p_1 & i_0 & p_2 & i_1 & i_2 & i_3 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \vec{c}H^T$$

A matriz H é denominada **matriz de paridade**.

Como a síndrome de uma palavra-código é nula, pode-se concluir que

$$GH^T = 0.$$

Para qualquer palavra-código $\vec{c}$ com erro simple $\vec{e}$, temos $\vec{c_1} = \vec{c} + \vec{e}$. A sua síndrome é dada por $(\vec{c} + \vec{e})H = \vec{c}H + \vec{e}H = \vec{e}H$. Segue-se que a **síndrome de uma palavra-código depende somente do padrão de erro**. Ainda mais, o produto da matriz H com um padrão de erro $\vec{e}$ de peso igual a 1 (erro simples) na posição i da palavra-código é igual à coluna i de H.

Exemplo 8.11 *Para os padrões de erros simples do código de Hamming (7,4), cujas palavras-código tem o formato* $\vec{c} = (p_0, p_1, i_0, p_2, i_1, i_2, i_3)$*, temos as seguintes síndromes:*

$$s = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Vale chamar atenção na organização dos bits deste código: a síndrome corresponde exatamente a posição do erro simples! Com isso pode se determinar facilmente o bit invertido com uso de um decodificador simples 3-to-8 (ver seção 6.1.1).

8.7 Códigos Cíclicos

Veremos nesta seção a construção de um código cíclico, cujo codificador e decoificador podem ser implmentados com uso de máquinas lineares (seção 6.8).

Um código $\mathcal{C}$ é chamado **cíclico** se para qualquer palavra-código $\vec{c} = (c_0, c_1, \cdots, c_{n-2}, c_{n-1}) \in \mathcal{C}$ existe também uma palavra-código $\vec{c'} = (c_{n-1}, c_0, \cdots, c_{n-3}, c_{n-2}) \in \mathcal{C}$. Para análise e processamento, é conviente associar a uma palavra-código $\vec{c} = (c_0, c_1, \cdots, c_{n-2}, c_{n-1}) \in \mathcal{C}$ no corpo $GF[q]$ um **polinômio-código** $c(x) = c_0 + c_1 D + \cdots + c_{n-2} D^{n-2} + c_{n-1} D^{n-1}$. As propriedades básicas de um código cíclico $\mathcal{C}(n,k)$ em $GF(q)$ são:

1. existe um único polinômio (mônico) de menor grau $r < n$, $g(D) = g_0 + g_1 x + \cdots + g_r D^r$, em $\mathcal{C}$. Este polinômio é conhecido o **polinômio gerador** de $\mathcal{C}$.

2. toda palavra-código $c(D) \in \mathcal{C}$ pode ser expressa de forma única como $c(D) = m(D)g(D)$, onde $g(D)$ é o polinômio gerador e $m(D)$ é um polinômio de grau menor que $(n-r)$ em $GF(q)[D]$.

3. o polinômio gerador é o divisor de $1 + D^n$ no $GF(q)[D]$.

O princípio básico destes códigos é construir uma palavra-código a partir de uma informação $m(D) = (m_0, m_1, \cdots, m_{k-r-1})$ através da multiplicação

$$c(D) = m(D)g(D) \tag{8.3}$$

Para decodificar esta palavra efetua-se simplesmente uma divisão

$$m(D) = \frac{c(D)}{g(D)}. \tag{8.4}$$

Caso a palavra $c(D)$ for alterada por algum ruído $e(D)$, termos $c_1(D) = c(D) + e(D)$ no lugra de $c(D)$. A divisão resultará em:

$$\frac{c_1(D)}{g(D)} = \frac{c(D)}{g(D)} + \frac{e(D)}{g(D)}. \tag{8.5}$$

Portanto, se a divisão for exata ($e(D) = 0$), $c_1(D)$ é uma palavra-código e $\frac{c_1(D)}{g(D)}$ corresponde a alguma mensagem; caso contrário, o resto $e(D)$ corresponde a um padrão de erro adicionado à palavra-código $c(D)$. Portanto,

$e(D)$ é a **síndrome** do código cíclico. Dependendo da quantidade de padões distintos e dos tipos de erros que podem ocorrer no sistema, podemos associar a cada síndrome um padrão de erro biunivocamente.

Uma forma alternativa para representar a Eq.(8.3) é com uso da notação matricial

$$\begin{aligned} c(D) = m(D)g(D) &= (m_0 + m_1 D + \cdots + m_{k-r-1}D^{k-r-1})g(D) \\ &= m_0 g(D) + m_1 D g(D) + \cdots + m_{k-r-1}D^{k-r-1}g(D) \\ &= \begin{bmatrix} m_0 & m_1 & \cdots & m_{k-r-1} \end{bmatrix} \begin{bmatrix} g(D) \\ Dg(D) \\ D^2 g(D) \\ \vdots \\ D^{k-r-1}g(D) \end{bmatrix} \end{aligned} \quad (8.6)$$

A matriz

$$G(D) = \begin{bmatrix} g(D) \\ Dg(D) \\ D^2 g(D) \\ \vdots \\ D^{k-r-1}g(D) \end{bmatrix} = \begin{bmatrix} g_0 & g_1 & \cdots & g_r & & & 0 \\ & g_0 & g_1 & \cdots & g_r & & \\ & & \ddots & \ddots & \ddots & & \\ & & & g_0 & g_1 & \cdots & g_r \end{bmatrix} \quad (8.7)$$

é conhecida como a **matriz geradora** dos códigos cíclicos.

O **polinômio-paridade** $h(D) = h_0 + h_1 D + \cdots + h_k D^k$ é aquele que satisfaz a expressão

$$c(D)h(D) = 0 \bmod (1 + D^n).$$

Particularmente, $h(D)$ deve ser a solução de $g(D)h(D) = (1+D^n) = 0 \bmod (1+D^n)$.

Pode-se mostrar que uma palavra é válida, se

$$s(D) = \begin{bmatrix} c_0 & c_1 & \cdots & c_{n-1} \end{bmatrix} \begin{bmatrix} h_k & \cdots & h_1 & h_0 & & & 0 \\ & h_k & \cdots & h_1 & h_0 & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & h_k & \cdots & h_1 & h_0 \\ 0 & & & & h_k & \cdots & h_1 & h_0 \end{bmatrix}^T = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

A matriz

$$H(x) = \begin{bmatrix} h_k & \cdots & h_1 & h_0 & & & 0 \\ & h_k & \cdots & h_1 & h_0 & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & h_k & \cdots & h_1 & h_0 \\ 0 & & & & h_k & \cdots & h_1 & h_0 \end{bmatrix} \quad (8.8)$$

é a **matriz de paridade**.

Exemplo 8.12 *Seja um polinômio primitivo $p(D) = 1+D^2+D^3$ em $GF(2)[D]$. Lembrando que um polinômio primitivo em $GF(q)[D]$ de grau m é divisor de $1 + D^{q^m-1}$, ele gera códigos de comprimento igual a $2^3 - 1 = 7$.*

A matriz geradora G do código é

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

As palavras válidas deste código são:

$m_0m_1m_2m_3$	$c_0c_1c_2c_3c_4c_5c_6$
0000	*0000000*
0001	*0001011*
0010	*0010110*
0011	*0011101*
0100	*0101100*
0101	*0100110*
0110	*0111010*
0111	*0110001*
1000	*1011000*
1001	*1010011*
1010	*1001110*
1011	*1000101*
1100	*1110100*
1101	*1111111*
1110	*1100010*
1111	*1101001*

A distância de Hamming do código é 3, portanto este código é capaz de detectar e corrigir erros simples.

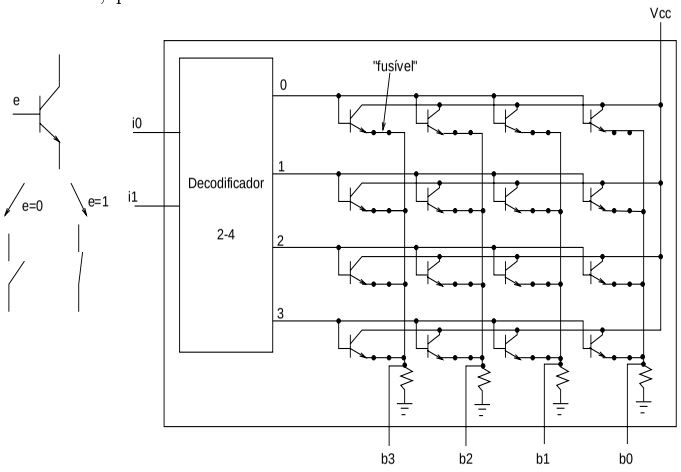
Sendo o polinômio $h(D) = \frac{1+D^7}{1+D^2+D^3} = 1 + D^2 + D^3 + D^4$*, a matriz de paridade* H *do código é*

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Assim, as síndromes para erros simples são: 100 (primeiro bit), 110 (segundo), 111 (terceiro), 011 (quarto), 101 (quinto), 010 (sexto) e 001 (sétimo bit).

Vimos na seção 6.8 que, com uso de *flip-flops* D e portas XOR, podemos implementar facilmente os codificadores e decodificadores deste código. Para isso, basta considerarmos $g(D)$ como a função de transferência do codificador e $\frac{1}{g(D)}$, a função de transferência do decodificador. E uma forma para corrigir os erros seria, **quando possível**, construir uma **tabela de decodificação**, cuja entrada é a síndrome e cuja saída é o padrão de erro. Este padrão de erro deve ser subtraído da palavra recebida para obtermos uma palavra válida.

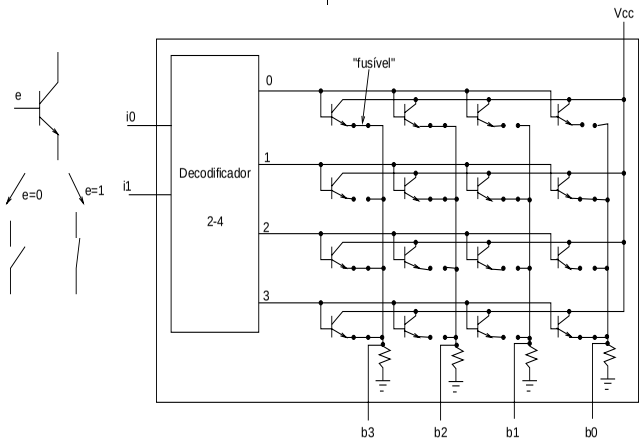
Podemos implementar esta tabela com uso de PROMS (*programmable read-only memory*) cuja estrutura interna consiste de um arranjo de "chaves eletrônicas", que fecham ao serem "excitadas".



Note-se que se quisermos armazenar, por exemplo, 1001 na linha 1, basta abrirmos os fusíveis nas colunas b_2 e b_1 da linha 1. Assim quando esta linha estiver no nível 1, leremos 1001 na saída $b_3b_2b_1b0$, mesmo que as outras linhas não estiverem no nível 1 ou os outros fusíveis das colunas b_2 e b_1 estiverem abertos. Daí a denominação de *wired-OR* para este tipo de arranjo.

Exemplo 8.13 *Programe um PROM* 2×4*, segundo a seguinte relação:*

Entrada	*Saída*
00	*1100*
01	*0111*
10	*1000*
11	*1001*

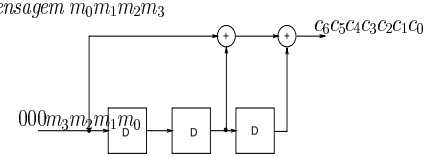


Note-se que as funções lógicas para cada saída é:

$$\begin{aligned} b_0 &= i_1\prime i_0 + i_1 i_0 \\ b_1 &= i_1\prime i_0 \\ b_2 &= i_1\prime i_0\prime + i_1 \prime i_0 \\ b_3 &= i_1\prime i_0\prime + i_1 i_0\prime + i_1 i_0 \end{aligned} \tag{8.9}$$

Exemplo 8.14 *Projete um código cíclico de comprimento 7 (7 bits) a partir de mensagens de 4 bits. O gerador utilizado é o polinômio primitivo* $p(D) = 1 + D^2 + D^3$ *em* $GF(2)[D]$*.*

Como as palavras-código devem satisfazer $c(D) = m(D)p(D)$, o seguinte multiplicador de polinômios é suficiente para gerar uma palavra-código a partir de uma mensagem $m_0m_1m_2m_3$



As palavras-código estão listadas no exemplo 8.12 e o código é um código corretor de erros simples.

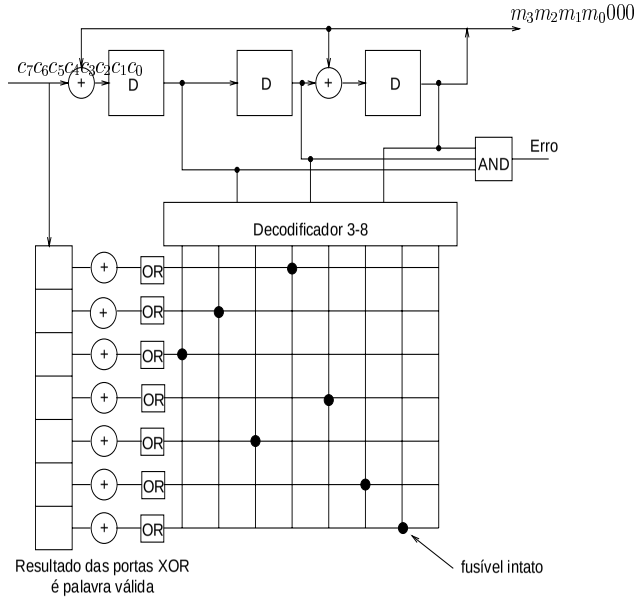
Para recuperar $m(D)$ a partir de uma palavra $c_1(D)$, um divisor de polinômios em $GF(2)[D]$ é suficiente. Sendo um código corretor de erros simples, podemos ainda tentar corrigir erros na recuperação da mensagem.

Lembrando que o resto de cada divisão fica armazenada nos flip-flops, podemos conectar a saída destes na entrada de uma tabela de decodificação que nos fornece o padrão de erro simples.

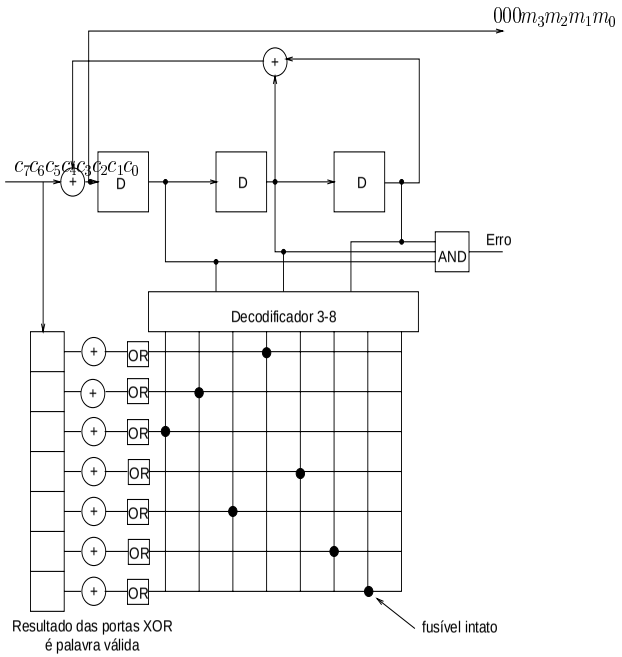
De acordo com o exemplo 8.12, o conteúdo desta tabela deve ser

Síndrome (entrada)	*Padrão de erro (saída)*
000	*0000000*
001	*0000001*
010	*0000010*
011	*0001000*
100	*1000000*
101	*0000100*
110	*0100000*
111	*0010000*

O esquema de um decodificador capaz de corrigir erros simples para o nosso código cíclico pode ser



Um outro esquema alternativo (seção 6.8) é



Observe ainda que, se entrarmos com a palavra 1000000 *no segundo circuito, na sequência de 1 a 0, obteremos uma sequência de 7 estados distintos que correspondem a síndromes dos padões de erro simples. O estado i corresponde exatamente a síndrome da inversão no bit i, considerando que o bit mais significativo seja o bit* 1 *e o bit menos significativo o bit* 7.