

# Capítulo 1

## Representação da Informação

### Códigos Binários e de Dados



# 1. Representação da Informação

## 1.4 Códigos Binários

1.4.1 Código BCD

1.4.2 Código de Gray

1.4.3 Códigos  $k$ -de- $n$

1.4.4 Códigos de paridade

1.4.5 Códigos de Hamming

# 1.4 Códigos Binários

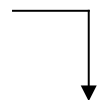
- Qualquer informação no computador é representada por códigos binários: caracteres, números, símbolos, etc.
- Existem diversas alternativas para codificar elementos dependendo das características desejadas.
- Um código pode ser otimizado para reduzir espaço de armazenamento necessário, ou para representar informações de forma unívoca, ou ainda explorar redundâncias para detecção e correção de erros.

## 1.4.1 Códigos BCD (Binary Coded Decimal)

- Associam os 10 algarismos decimais (0 a 9) a códigos de 4 bits (16 combinações possíveis)
- Diversas associações são utilizadas, com predominância da representação *BCD natural*
- Na tabela apresentada a seguir, os pesos associados a cada um dos quatro dígitos binários aparecem entre parenteses

## 1.4.1 Códigos BCD (Binary Coded Decimal)

Mais utilizado



Dígito	BCD natural			
	(8	4	2	1)
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

BCD Natural é igual ao código binário até o Valor 9. Os valores entre 10 e 15 não são válidos.

## 1.4.1 Códigos BCD (Binary Coded Decimal)

Dígito	BCD natural				Aiken			
	(8	4	2	1)	(2	4	2	1)
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1
4	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	1	1
6	0	1	1	0	1	1	0	0
7	0	1	1	1	1	1	0	1
8	1	0	0	0	1	1	1	0
9	1	0	0	1	1	1	1	1

Aiken é igual ao código binário até o Valor 4. Os valores 5 a 9 são formados Pela inversão dos bits dos Valores 4 a 0.

## 1.4.1 Códigos BCD (Binary Coded Decimal)

Excesso-de-três: simplifica a aritmética BCD

Dígito	BCD natural				Aiken				Stibitz			
	(8	4	2	1)	(2	4	2	1)	(8	4	2	1) -3
0	0	0	0	0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	0	1	0	0	1	0	1
3	0	0	1	1	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	1	1	1	0	0	0
6	0	1	1	0	1	1	0	0	1	0	0	1
7	0	1	1	1	1	1	0	1	1	0	1	0
8	1	0	0	0	1	1	1	0	1	0	1	1
9	1	0	0	1	1	1	1	1	1	1	0	0

Stibitz é igual  
ao BCD – 3.  
(Complemento 9)

# 1.4.1 Códigos BCD (Binary Coded Decimal)

Excesso-de-três: simplifica a aritmética BCD

Dígito	BCD natural				Aiken				Stibitz											
	(8	4	2	1)	(2	4	2	1)	(8	4	2	1) -3	(7	4	2	1)	(6	4	2	-1)
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	0	1	1
2	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1	0	1	1	0	0	0	1	1	0	1	0	1
4	0	1	0	0	0	1	0	0	0	1	1	1	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	1	1	1	0	0	0	0	1	0	1	0	1	1	1
6	0	1	1	0	1	1	0	0	1	0	0	1	0	1	1	0	1	0	0	0
7	0	1	1	1	1	1	0	1	1	0	1	0	0	1	1	1	1	0	1	1
8	1	0	0	0	1	1	1	0	1	0	1	1	1	0	0	1	1	0	1	0
9	1	0	0	1	1	1	1	1	1	1	0	0	1	0	1	0	1	1	0	1

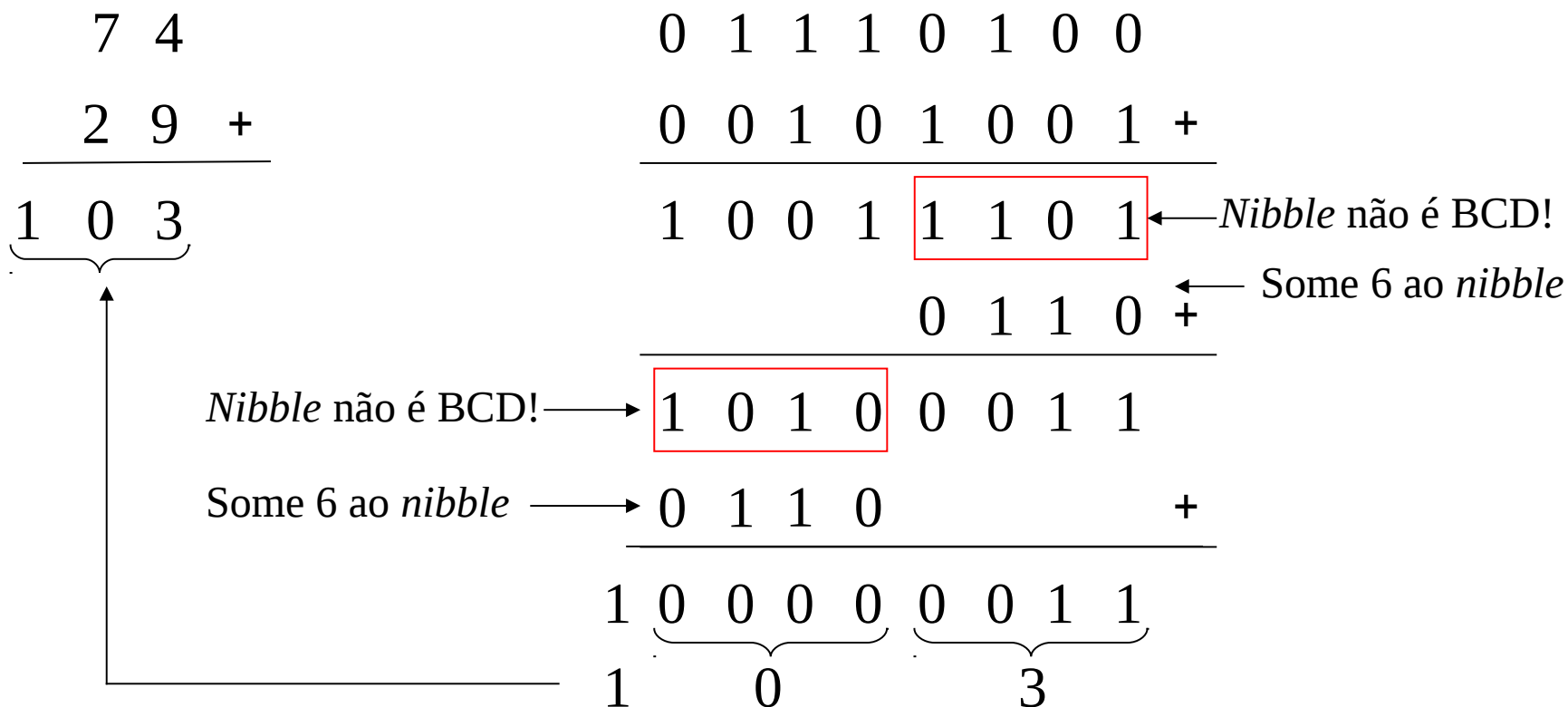


## 1.4.1 Códigos BCD (Binary Coded Decimal)

- Algoritmo para soma de números BCD
  1. Efetuar a soma binária convencional dos dois números
  2. Adicionar 6 a cada *nibble* (grupo de 4 bits) que não seja um valor BCD válido
  3. Repetir o passo 2 até que todos os *nibbles* do resultado correspondam a valores BCD válidos

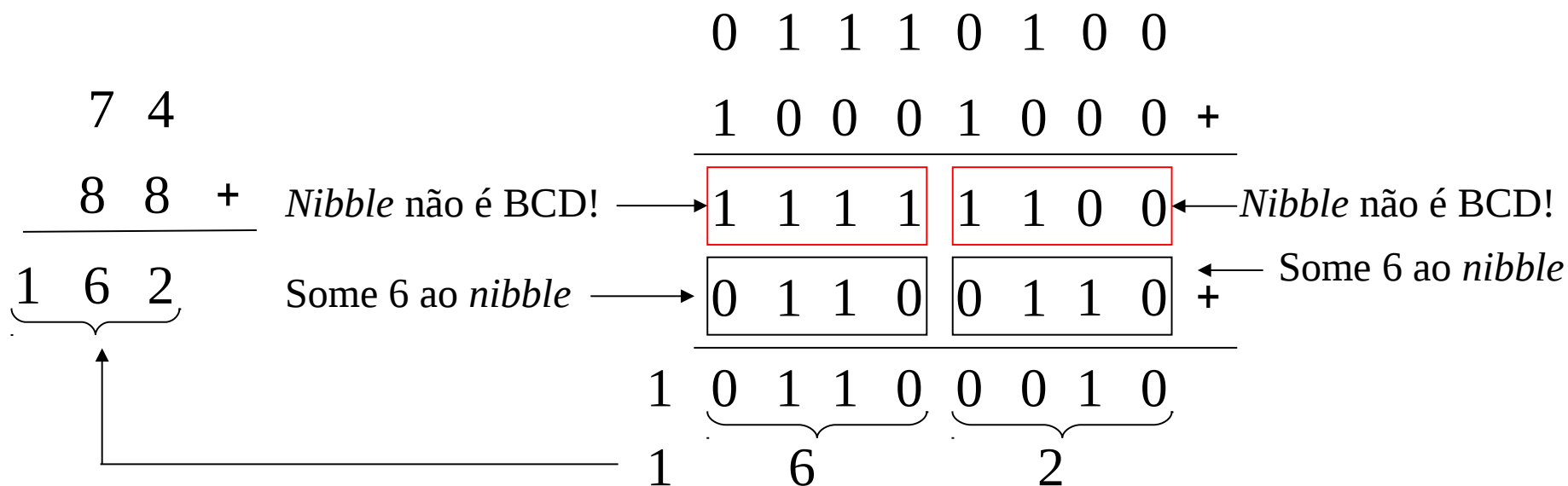
## 1.4.1 Códigos BCD (Binary Coded Decimal)

### ■ Exemplo de soma de números BCD



## 1.4.1 Códigos BCD (Binary Coded Decimal)

### ■ Exemplo de soma de números BCD



## 1.4.2 Código de Gray

- É um código numérico binário onde dois valores sucessivos diferem em somente um bit
- Também conhecido como *código binário refletido*, pois o código de Gray para  $n$  bits pode ser obtido a partir da reflexão do código de Gray para  $(n-1)$  bits em torno de um eixo situado ao término do código
  - Adiciona-se “0” como bit mais significativo (MSB - *Most Significant Bit*) acima do eixo
  - Adiciona-se “1” como MSB abaixo do eixo.

# 1.4.2 Código de Gray

1 bit

0

1

2 bits

0

0

0

1

1

0

1

Refletir

3 bits

0

0 0

0

0 1

0

1 1

0

1 0

1

1 0

1

1 1

1

0 1

1

0 0

1

Refletir

0

0 0 0

0

0 0 1

0

0 1 1

0

0 1 0

0

1 1 0

0

1 1 1

0

1 0 1

0

1 0 0

Refletir

1

1 0 0

1

1 0 1

1

1 1 1

1

1 1 0

1

0 1 0

1

0 1 1

1

0 0 1

1

0 0 0

# 1.4.2 Código de Gray

## ■ Conversão Gray – binário

□  $g_i = i$ -ésimo bit do código de Gray

■  $g_0 = \text{MSB}$

□  $b_i = i$ -ésimo bit do código binário

■  $b_0 = \text{MSB}$

$$g_0 = b_0$$

$$g_i = b_i \oplus b_{i-1}$$

Gray			Binário		
$g_0$	$g_1$	$g_2$	$b_0$	$b_1$	$b_2$
0	0	0	0	0	0
0	0	1	0	0	1
0	1	1	0	1	0
0	1	0	0	1	1
1	1	0	1	0	0
1	1	1	1	0	1
1	0	1	1	1	0
1	0	0	1	1	1

## 1.4.2 Código de Gray

### ■ Conversão Gray – binário

□  $g_i = i$ -ésimo bit do código de Gray

■  $g_0 = \text{MSB}$

□  $b_i = i$ -ésimo bit do código binário

■  $b_0 = \text{MSB}$

$$b_0 = g_0$$

$$b_i = g_i \oplus b_{i-1}$$

Binário			Gray		
$b_0$	$b_1$	$b_2$	$g_0$	$g_1$	$g_2$
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

## 1.4.2 Código de Gray

### ■ Principais utilizações

#### □ *Codificadores mecânicos*

- Pequenas mudanças de posição afetam apenas um único bit, diferentemente de certas situações que ocorrem com o código binário tradicional

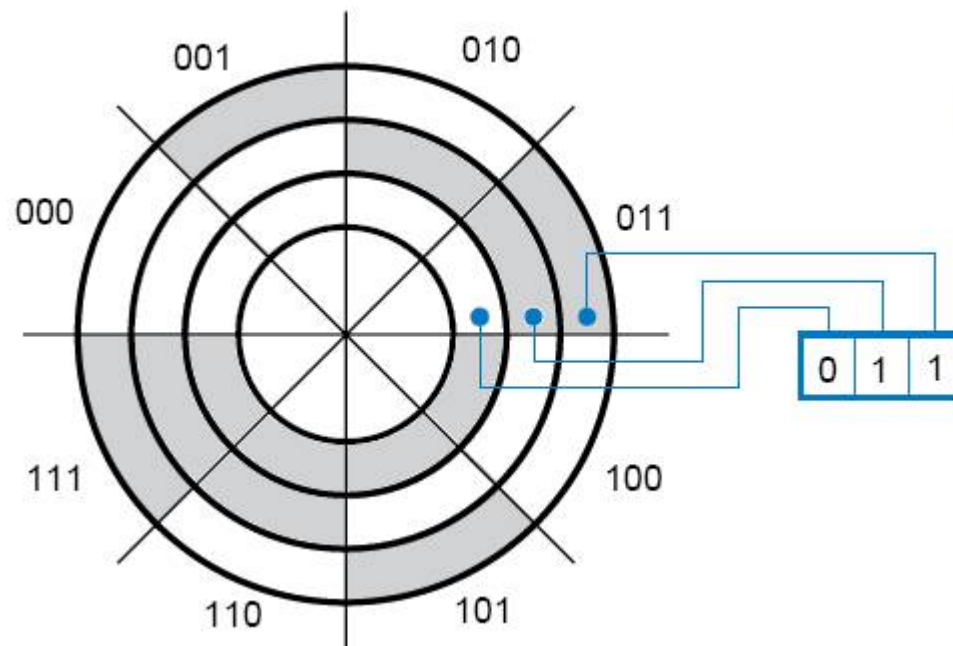
#### □ *Mapas de Karnaugh*

- O ordenamento das células é feito segundo o código de Gray, para possibilitar as simplificações booleanas
- Explorados no Capítulo 2



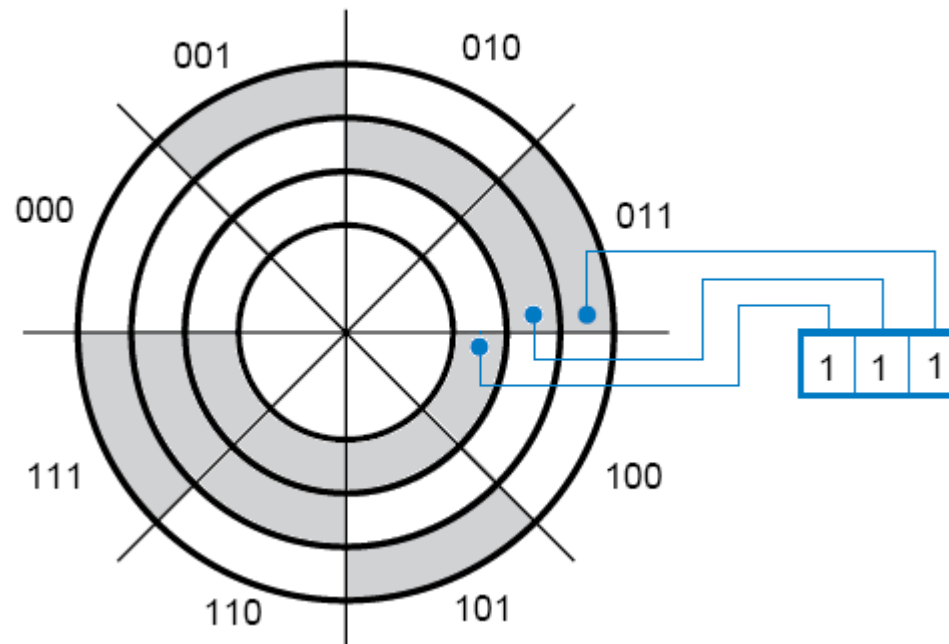
## 1.4.2 Código de Gray

- Ex: Codificador binário para um sistema mecânico rotacional
  - Codificação binária: 45°



## 1.4.2 Código de Gray

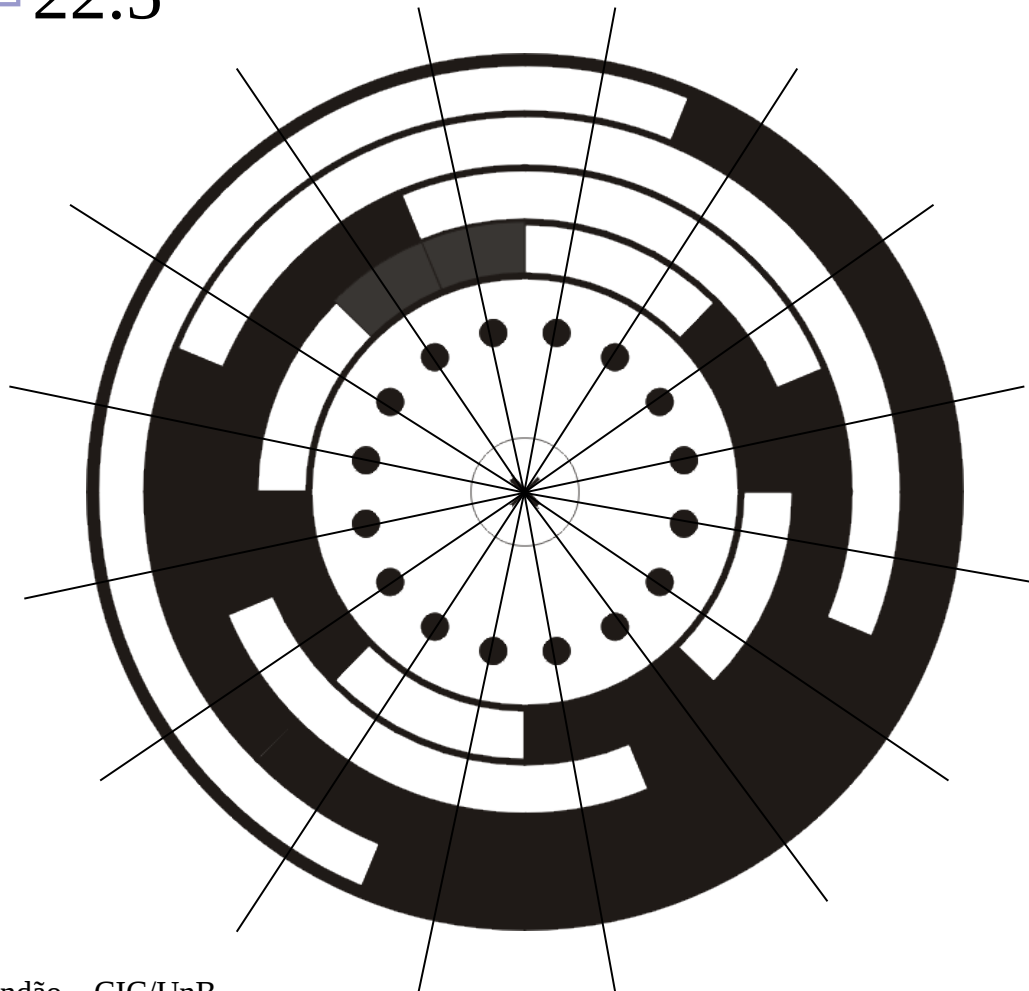
- Ex: Codificador binário para um sistema mecânico rotacional
- Caso haja desbalanceamento nas agulhas o erro produzido pode ser grande:



# 1.4.2 Código de Gray

## ■ Codificador mecânico rotacional

□ 22.5°



Posição	Código			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0
13	1	0	1	1
14	1	0	0	1
15	1	0	0	0

## 1.4.3 Códigos $k$ de $n$

- São códigos *ponderados* constituídos por  $n$  bits
  - $k$  bits são “1”
  - $n-k$  bits são “0”
- Normalmente utilizados em detecção de erros transmissões de dados
- Número de combinações possíveis

$$\frac{n!}{k!(n-k)!}$$

## 1.4.3 Códigos $k$ de $n$

■ Exemplo: código 74210, ou 2 de 5

□ A detecção de erros pode ser feita simplesmente conferindo se o número de “1”s for diferente de 2

Dígito decimal	(7 4 2 1 0)	
0	1 1 0 0 0	← Caso especial
1	0 0 0 1 1	
2	0 0 1 0 1	
3	0 0 1 1 0	
4	0 1 0 0 1	
5	0 1 0 1 0	
6	0 1 1 0 0	
7	1 0 0 0 1	
8	1 0 0 1 0	
9	1 0 1 0 0	

## 1.4.3 Códigos $k$ de $n$

■ Exemplo: código 2 de 7 bits ponderado

□ (50 43210), ou biquinário

Dígito decimal	(5	0	4	3	2	1	0)
0	0	1	0	0	0	0	1
1	0	1	0	0	0	1	0
2	0	1	0	0	1	0	0
3	0	1	0	1	0	0	0
4	0	1	1	0	0	0	0
5	1	0	0	0	0	0	1
6	1	0	0	0	0	1	0
7	1	0	0	0	1	0	0
8	1	0	0	1	0	0	0
9	1	0	1	0	0	0	0

## 1.4.3 Códigos de Paridade

- Em códigos de *paridade simples* acrescenta-se um bit à palavra de tal forma que a paridade seja *par* ou *ímpar*
- O objetivo é a detecção de *erros simples* na transmissão de dados

## 1.4.3 Códigos de Paridade

Paridade			Par	Nº de “1”s
Código				
0	0	0	0	0
0	0	1	1	2
0	1	0	1	2
0	1	1	0	2
1	0	0	1	2
1	0	1	0	2
1	1	0	0	2
1	1	1	1	4

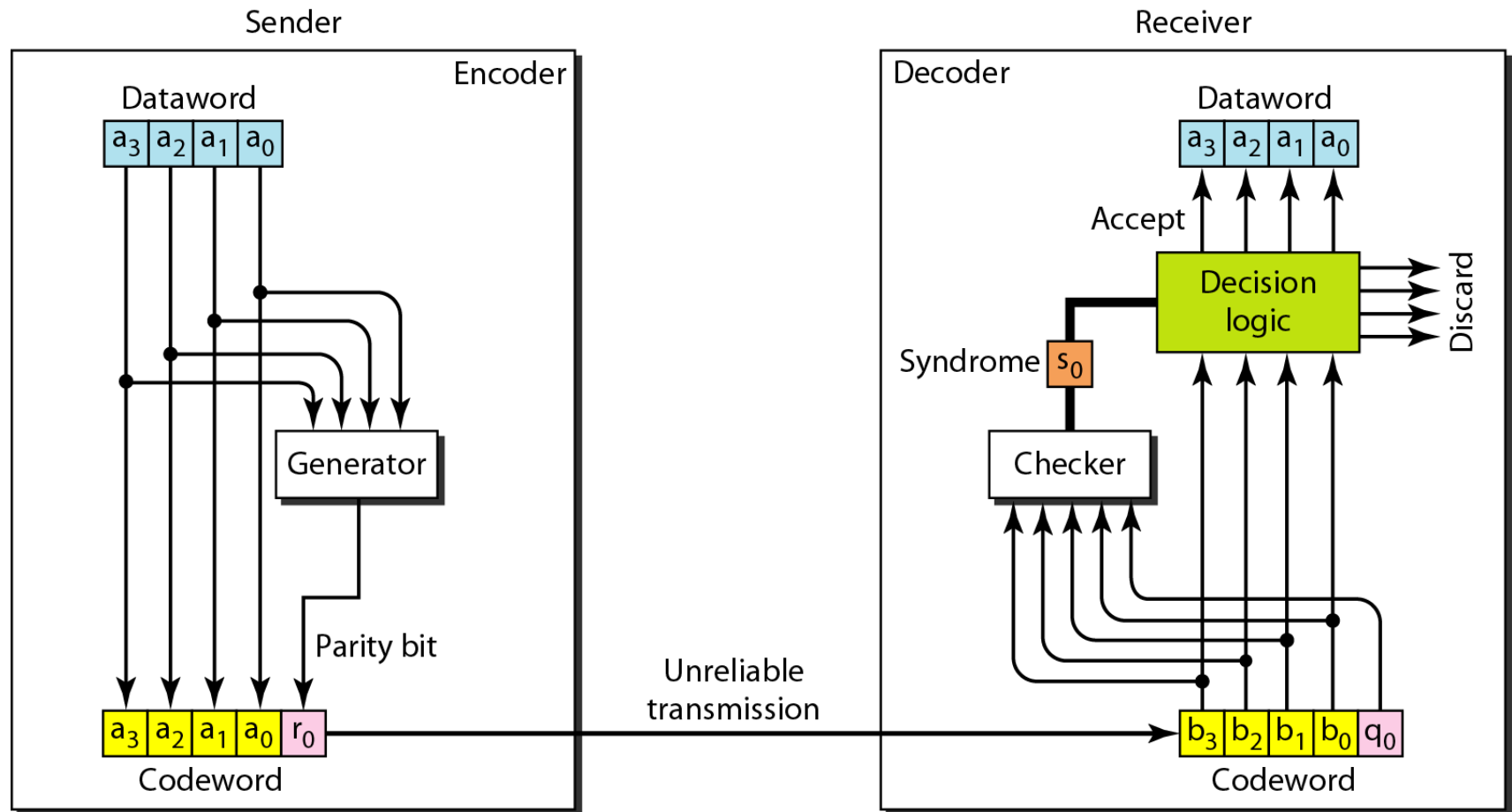


## 1.4.3 Códigos de Paridade

Paridade			Nº de “1”s
Código	Ímpar		
0 0 0	1		1
0 0 1	0		1
0 1 0	0		1
0 1 1	1		3
1 0 0	0		1
1 0 1	1		3
1 1 0	1		3
1 1 1	0		3

# 1.4.3 Códigos de Paridade

## ■ Paridade em transmissão de dados



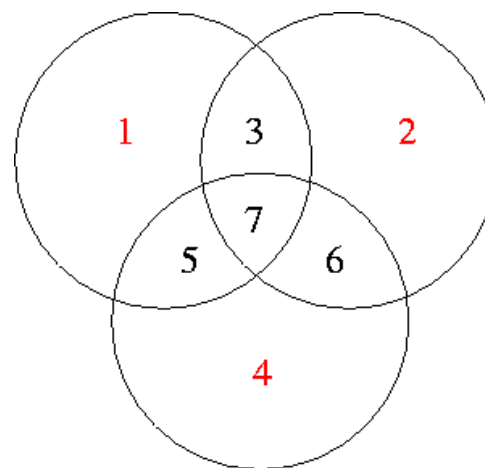
## 1.4.3 Código de Hamming

- Códigos de Hamming utilizam vários bits de paridade para
  - a *detecção* e *correção* de erros simples (em apenas um bit da palavra)
  - a *detecção* (mas não a *correção*) de erros duplos
- É possível demonstrar matematicamente que, para cada inteiro  $m > 2$ , existe um código de Hamming com  $m$  bits de paridade e com  $2^m - m - 1$  bits de dados

## 1.4.3 Código de Hamming

- Ilustramos aqui o código de Hamming (7,4) para palavras de 7 bits com 4 bits de dados
  - Existem 7 erros simples possíveis
  - Os **3 bits de paridade** utilizados são dispostos de tal forma a permitir a identificação (e correção) de erro simples

7	6	5	4	3	2	1	Paridade
D	-	D	-	D	-	P	Par
D	D	-	-	D	P	-	Par
D	D	D	P	-	-	-	Par



# 1.4.3 Código de Hamming

- Note que uma *palavra válida* difere de outra em, no mínimo, 3 bits (*distância mínima*).
- Pode-se demonstrar que, em uma família de palavras com *distância mínima*=3, qualquer *erro simples* pode ser detectado e corrigido.

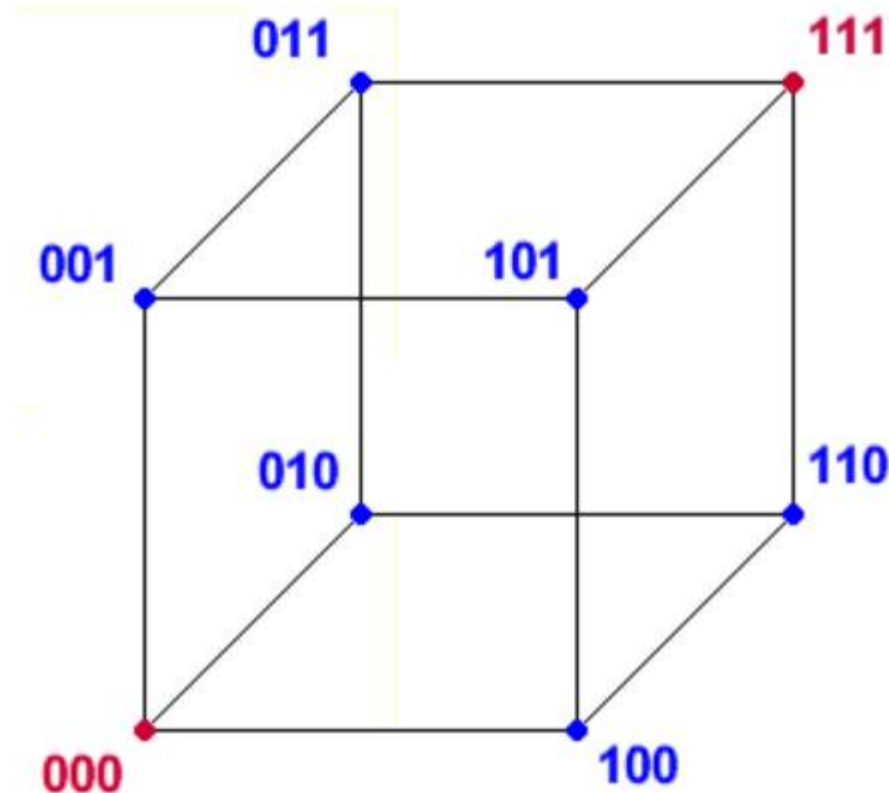
Dado	Bits de Dados						
	7	6	5	4	3	2	1
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1
2	0	0	1	1	0	0	1
3	0	0	1	1	1	1	0
4	0	1	0	1	0	1	0
5	0	1	0	1	1	0	1
6	0	1	1	0	0	1	1
7	0	1	1	0	1	0	0
8	1	0	0	1	0	1	1
9	1	0	0	1	1	0	0
A	1	0	1	0	0	1	0
B	1	0	1	0	1	0	1
C	1	1	0	0	0	0	1
D	1	1	0	0	1	1	0
E	1	1	1	1	0	0	0
F	1	1	1	1	1	1	1

## 1.4.3 Código de Hamming

■ Exemplo de distância mínima = 3

□ valor “000” válido

□ valor “111” válido



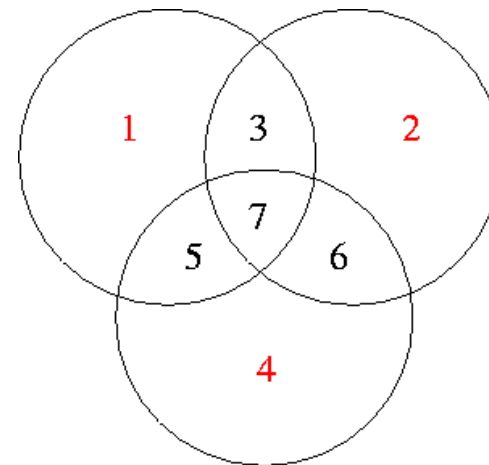
## 1.4.3 Código de Hamming

- Na recepção da palavra, os três bits de paridade são recalculados e comparados com os valores recebidos. Se ocorrer um erro simples (divergência em apenas um bit da palavra) a posição do bit errado pode ser determinada como abaixo:
  - A função *erro\_de\_paridade*, cujo valor é "0" se não há erro de paridade e "1" se há erro de paridade, é calculada para cada um dos três bits de paridade;
  - A palavra *Erro\_de\_Paridade*, formada pelos bits *erro\_de\_paridade(Bit 4)*, *erro\_de\_paridade(Bit 2)* e *erro\_de\_paridade(Bit 1)*, aponta para o bit errado da palavra.

# 1.4.3 Código de Hamming

- *Erro\_de\_Paridade* aponta para o bit com erro, que pode então ser corrigido:

Bit	7	6	5	4	3	2	1
Palavra	1	0	0	0	0	0	0
Erro de Paridade				1		1	1



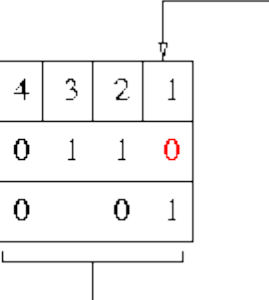
Bit	7	6	5	4	3	2	1
Palavra	0	1	0	0	0	0	0
Erro de Paridade				1		1	0

Bit	7	6	5	4	3	2	1
Palavra	1	1	1	1	1	0	0
Erro de Paridade				0		1	1

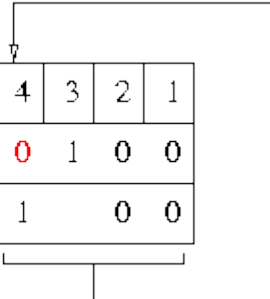


## 1.4.3 Código de Hamming

- Um dos *Bits de paridade* pode também estar com erro, e ser devidamente corrigido.



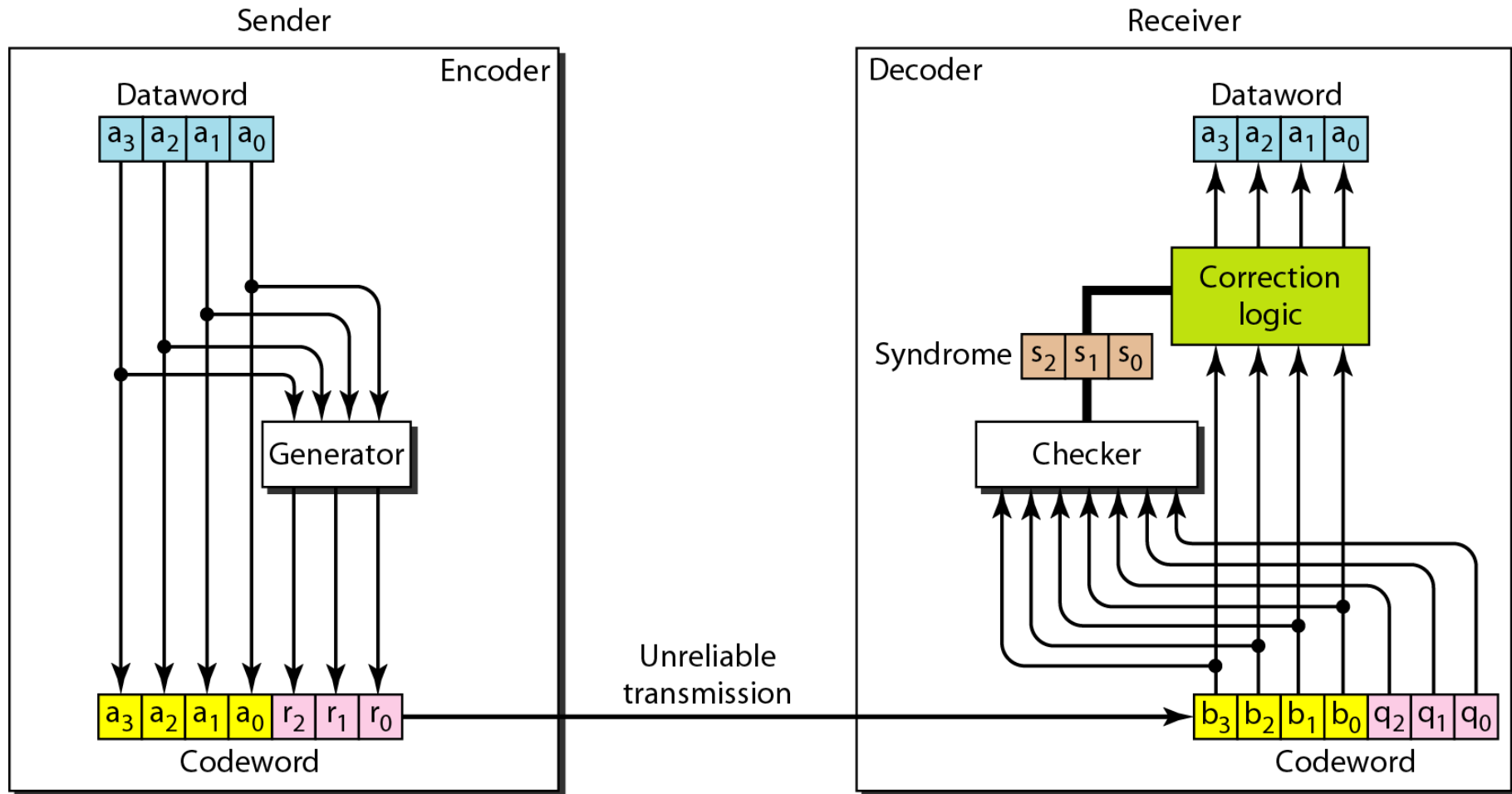
Bit	7	6	5	4	3	2	1
Palavra	0	0	0	0	1	1	0
Erro de Paridade				0		0	1



Bit	7	6	5	4	3	2	1
Palavra	1	0	0	0	1	0	0
Erro de Paridade				1		0	0

# 1.4.3 Código de Hamming

## ■ Codificador/decodificador para o Código de Hamming



# Referências

- Código de Hamming

- [http://www.ics.uci.edu/~magda/Courses/netsys270/ch10\\_2\\_v1.ppt](http://www.ics.uci.edu/~magda/Courses/netsys270/ch10_2_v1.ppt)

- Pedroni, Volnei A. (2010), *Eletrônica Digital Moderna e VHDL*, Elsevier