PCS 3115 (PCS2215)

Sistemas Digitais I

Módulo 04 – Códigos

Prof. Dr. Marcos A. Simplicio Jr.

versão: 3.0 (Jan/2016)

Conteúdo

- Códigos Binários para Decimais
  - BCD ou 8421
  - 2421
  - Excesso 3
  - 2 entre 5
  - 1 entre n
  - 7 segmentos
  - Gray
- Detecção e correção de erros
  - Hamming
- Caracteres
  - ASCII
  - Unicode
- Transmissão Serial
  - Alguns exemplos informativos

CÓDIGOS NUMÉRICOS

Códigos Numéricos

- Conjunto de cadeias com *n* bits: cadeias diferentes representam coisas diferentes;
  - Para *n* bits existem 2^n códigos válidos de comprimento fixo
- Exemplo: códigos numéricos
  - Computadores: operam com bits
  - Humanos: preferem operar com decimais
  - ➔ Códigos permitem codificar 10 símbolos decimais (0-9) em termos de bits

Códigos Numéricos

- **Pergunta 1**: quantos bits são necessários para representar 10 dígitos?
  - Resposta: 4 bits ➔ 16 possibilidades (6 códigos não são usados)
- **Pergunta 2**: Quantos diferentes códigos numéricos (i.e., para 10 dígitos) são possíveis de serem criados com esse número de bits?
  - Resposta: 16!/(16-10)! = 16*15*...*7 ~ 29 milhões
    - Corresponde a um arranjo simples (ordem dos elementos é considerada)
- **Pergunta 3**: Qual o código numérico mais imediato e fácil de ser compreendido ? Como você o chamaria ?
  - Resposta: código em binário que representa os números decimais, conforme visto na aula anterior, i.e., número representa valor do bit. Nome dado: BCD (*binary-coded decimal*) ou "8421".

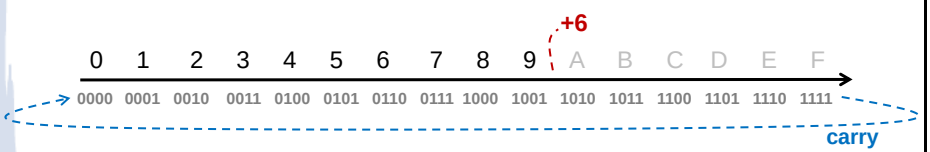
BCD – Binary Coded Decimal

- Cadeias de 4 bits;
  - Código 8421: indica peso de cada um dos bits na cadeia
- Variantes
  - *Packed BCD*: 2 dígitos BCD em 1 byte ➔ 00 a 99;
  - BCD com sinal: um dígito extra representando "+" ou "–"

Dígito decimal	BCD (8421)	Dígito decimal	BCD (8421)
0	0000	8	1000
1	0001	9	1001
2	0010	não usado	1010
3	0011		1011
4	0100		1100
5	0101		1101
6	0110		1110
7	0111		1111

BCD – Binary Coded Decimal

- **Adição**: similar à de números binários de 4 bits
  - Mas com correção se resultado ultrapassar 1001: somar 6



- Exemplos:

correção

5	0101		8	1000		4	0100
+ 9	+ 1001		+ 8	+ 1000		+ 5	+ 0101
14	1110	> 1001	16	1 0000	> 1001	9	1001
	+ 0110			+ 0110			
10+4	1 0100		10+6	1 0110			

Código 2421

- Nome indica peso de cada um dos bits na cadeia
  - Ex.: 5 = 1011, pois 2*1+4*0+2*1+1+1 = 5
- **Propriedade**: **auto-complementar**
  - Ao inverter os bits do código de um determinado dígito, obtém-se o código do complemento de 9 daquele dígito
  - Ex.: $Compl_9(8) = 1$ ➔ código de 8 é 1110 (o inverso de 0001)

Dígito decimal	2421	Dígito decimal	2421
0	0000	9	1111
1	0001	8	1110
2	0010	7	1101
3	0011	6	1100
4	0100	5	1011
Não usado (mas poderia ser)	0101 (5) 0110 (6) 0111 (7)	Não usado (mas poderia ser)	1010 (4) 1001 (3) 1000 (2)

Código Excesso-3

- Equivalente a BCD + 3
- **Propriedade**: **auto-complementar**
  - Ex.: $Compl_9(8) = 1$ ➔ código de 8 é 1110 (o inverso de 0001)
- Vantagem sobre 2421: aritmética similar a BCD
  - **Lição de casa**: pesquisar/deduzir como fazer

Dígito decimal	Excesso-3	Dígito decimal	Excesso-3
Não usado	0000	5	1000
	0001	6	1001
	0010	7	1010
0	0011	8	1011
1	0100	9	1100
2	0101	Não usado	1101
3	0110		1110
4	0111		1111

Código 2 entre 5

- Código de 5 bits:
  - Usam-se todos os códigos em que (número de bits 1) = 2
- **Propriedade**: correção de erros
  - Em uma transmissão digital, sempre se esperam dois bits '1': se houver um erro na transmissão e um bit for invertido, erro é facilmente detectado.

Dígito decimal	2 entre 5	Dígito decimal	2 entre 5
0	00011	6	10001
1	00101	7	10010
2	00110	8	10100
3	01001	9	11000
4	01010	Não usados: todos os códigos que não tenham dois bits 1	
5	01100		

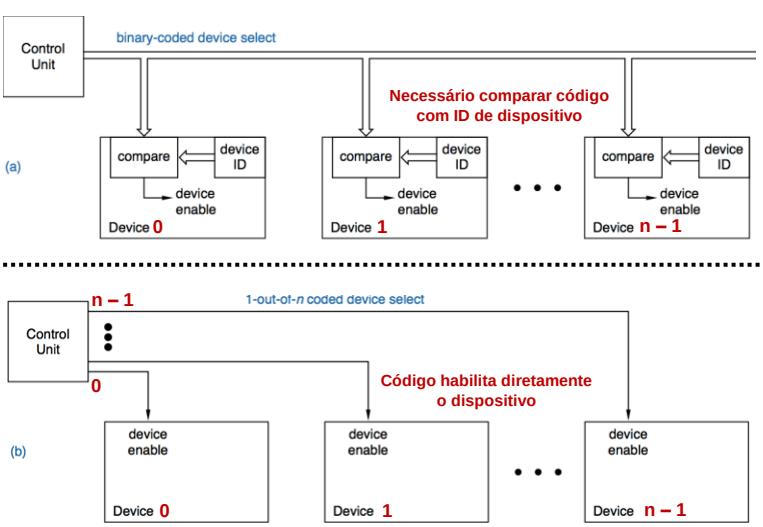
Código 1 entre n

- Código de n bits:
  - A posição do bit indica o valor representado
- **Propriedade**: correção de erros; facilita seleção
  - Grande possibilidade de detectar inversão de bits
    - Apenas 10 códigos válidos dentre 1024 possíveis
  - Sinal de habilitação de circuito de seleção pode ser ligado diretamente a código

Dígito decimal	1 entre 10	Dígito decimal	1 entre 10
0	0000000001	5	0000100000
1	0000000010	6	0001000000
2	0000000100	7	0010000000
3	0000001000	8	0100000000
4	0000010000	9	1000000000
Não usados: todos os códigos que não tenham somente um bit 1			

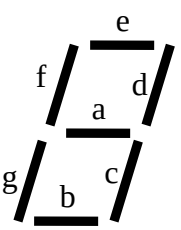
Código 1 entre n

- Habilitação de circuito de seleção: ligável diretamente a código

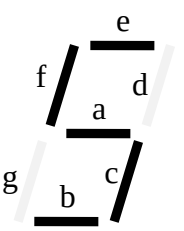


Código 7 Segmentos

- **Problema**: criar código para iluminar LEDs correspondentes em display de 7 segmentos
  - Objetivo: evitar necessidade de decodificadores



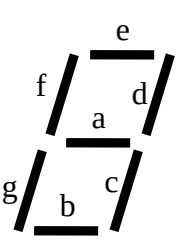
Exemplo: 5



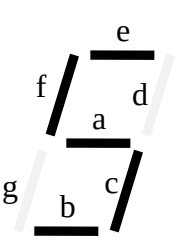
- Quantos bits? Qual o código para cada valor decimal?

Código 7 Segmentos

- Utilizado para iluminar LEDs correspondentes em display de 7 segmentos



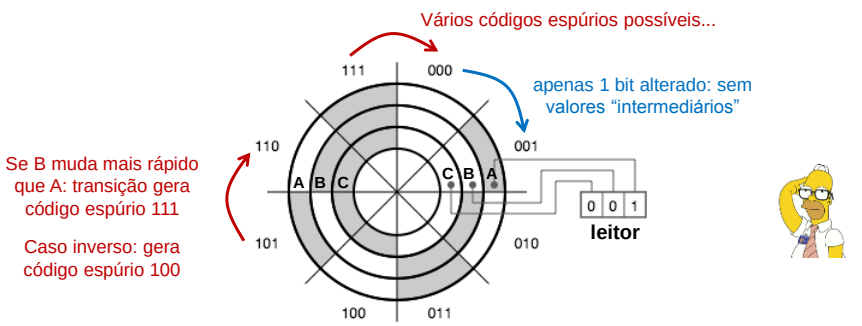
Exemplo: 5
➔**1110110**
abcdefg



Dígito decimal	7 Segmentos	Dígito decimal	7 Segmentos
0	0111111	5	1110110
1	0011000	6	1110011
2	1101101	7	0011100
3	1111100	8	1111111
4	1011010	9	1011110

Código de Gray

- Cenário: aplicações eletromecânicas (ex.: copiadora, freio automotivo, etc.)
  - Valor digital no leitor indica posição mecânica
- **Problema**:
  - **Valor "na transição"** pode ser interpretado (incorretamente) como a posição atual



Código de Gray

- Propriedades:
  - Apenas **um bit alterado** entre um código e seu sucessor
- Construção:
  - 0 | 1 ← **Parte-se de 0 e 1**
  - 00 01 | 11 10 ← **A 2a. metade é a sequência invertida**
  - 000 001 011 010 | 110 111 101 100

0s **1s**

Dígito decimal	BCD	Gray	Dígito decimal	BCD	Gray
0	000	000	4	100	110
1	001	001	5	101	111
2	010	011	6	110	101
3	011	010	7	111	100

Código de Gray

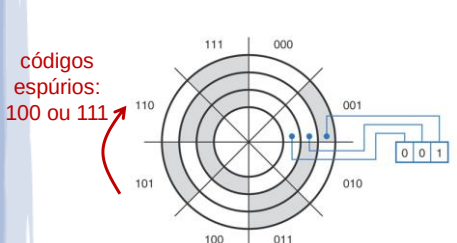
- Propriedades:
  - Apenas **um bit alterado** entre um código e seu sucessor
- Construção alternativa (baseada no BCD):
  - Enumere os bits da direita para a esquerda
  - Inspecione código BCD: se, em BCD, bit i = bit (i+1), então bit i em Gray é **0**; senão, é **1**. Manter bit mais significativo

(alguns exemplos abaixo)

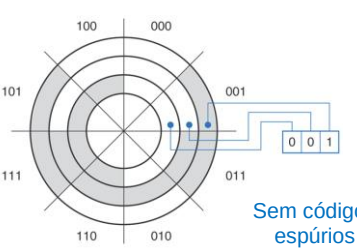
Dígito decimal	BCD	Gray	Dígito decimal	BCD	Gray
0	000	000	4	100	110
1	001	001	5	101	111
2	010	011	6	110	101
3	011	010	7	111	100

Código de Gray

- Não aparecem valores espúrios na transição entre posições do disco



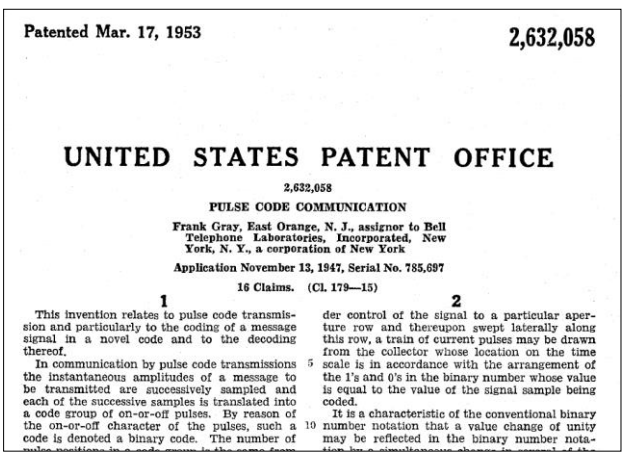
Codificação BCD



Codificação Gray

Código de Gray

- Patente US 2,632,058 – “Pulse Code Communication” (1953). Inventor: Frank Gray, Bell Labs



Patented Mar. 17, 1953 2,632,058

UNITED STATES PATENT OFFICE

2,632,058

PULSE CODE COMMUNICATION

Frank Gray, East Orange, N. J., assignor to Bell Telephone Laboratories, Incorporated, New York, N. Y., a corporation of New York

Application November 13, 1947, Serial No. 785,697

16 Claims. (Cl. 179—15)

1

This invention relates to pulse code transmission and particularly to the coding of a message signal in a novel code and to the decoding thereof.

In communication by pulse code transmissions the instantaneous amplitudes of a message to be transmitted are successively sampled and each of the successive samples is translated into a code group of on-or-off pulses. By reason of the on-or-off character of the pulses, such a code is denoted a binary code. The number of

2

der control of the signal to a particular aperture row and thereupon swept laterally along this row, a train of current pulses may be drawn from the collector whose location on the time scale is in accordance with the arrangement of the 1's and 0's in the binary number whose value is equal to the value of the signal sample being coded.

It is a characteristic of the conventional binary number notation that a value change of unity may be reflected in the binary number nota-

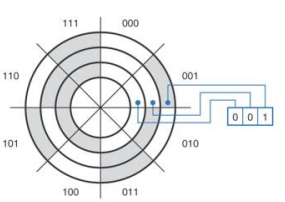
Exercícios

1) Converta os números decimais para BCD e 2421:

a) 742 b) 268

2) Some os números 742 e 268 em BCD

3) Quantas “fronteiras ruins” existem no disco de codificação BCD de 3 bits?)



4) Responda a questão 3 para um disco de n bits, em função de n

Exercícios: Respostas

1) Converta os números decimais para BCD e 2421:

a) 742 ➔ BCD: 0111 0100 0010 2421: 1101 0100 0010

b) 268 ➔ BCD: 0010 0110 1000 2421: 0010 1100 1110

2) Some os números 742 e 268 em BCD

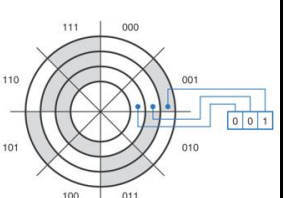
```
  742          0111      0100      0010
+ 268        +  0010      0110      1000
 1010          1001      1010      1010   > 1001
             +  0000    + 0110    + 0110   ➔ +6
                1010      0001      0000
             +  0110
              1 0000
              1    0         1         0
```


Exercícios: Respostas

3) Quantas “fronteiras ruins” existem no disco de codificação BCD de 3 bits?)

001 → 010 101 → 110

011 → 100 111 → 000



4) Responda a questão 3 para um disco de n bits, em função de n

- Quando bit menos significativo (LSB) muda de 0 para 1, não há problema. O problema é quando ele muda de 1 para 0, gerando carries, o que altera pelo menos um bit mais alto. Logo, metade das fronteiras (2^{n-1}) geram problemas
  - ➔ Obs.: no caso patológico de n =1, nenhuma fronteira gera problemas

CÓDIGOS PARA CARACTERES

Código ASCII

- Informação processada por computador: bits
- Então como representar texto...?
  - Usa-se um código: tabela que especifica representação binária para um determinado conjunto de símbolos.
  - Comumente: ASCII (*American Standard Code for Information Interchange*),
    - Código alfanumérico: letras do alfabeto, números, símbolos, sinais e alguns caracteres não-imprimíveis de controle
    - 7 bits: 128 caracteres diferentes (letras com acentos não inclusas)
    - Pronúncia: o correto é “ASKI”, **não** “ASK2”
  - Hum... Mas por que não 8 bits (= 1 byte)?
    - 8º bit usado normalmente como bit de paridade
    - O ASCII-estendido usa 8º bit para codificação extra (e.g., acentos)

Código ASCII

$b_3b_2b_1b_0$	Row (hex)	$b_6b_5b_4$ (column) 000 0	001 1	010 2	011 3	100 4	101 5	110 6	111 7
0000	0	NUL	DLE	SP	0	@	P	`	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	"	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	'	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	,	<	L	\	l	|
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	_	o	DEL

Código ASCII

Control codes

NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronize
BEL	Bell	ETB	End transmitted block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete or rubout

Unicode

- ASCII não suporta caracteres com acentos...
  - Apenas 7 bits: desenvolvido para alfabeto inglês
- Unicode: suporte a caracteres em múltiplos idiomas
  - 32 bits: ~4 bilhões de símbolos possíveis
  - Construído para ser compatível com ASCII: códigos de 00 a 7F representam os mesmos símbolos em Unicode e ASCII
  - Padrão ISO/IEC 10646: define 3 métodos de codificação UTF-8, UTF-16 e UTF-32
    - UTF = Unicode Transformation Format
  - Maiores informações: www.unicode.org

DETECÇÃO E CORREÇÃO DE ERROS

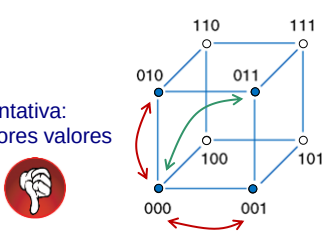
Detecção de erros

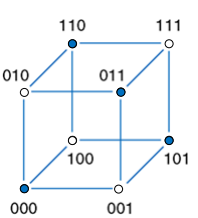
- Erro: alteração de um ou mais bits de um dado
  - Ex.: Interferência eletromagnética durante comunicação
  - Ex.: Eletricidade estática alterando dados armazenados
- **Pergunta**: como detectar esses erros?
  - Resposta: detecção de erros possível se nem todas as palavras do código são válidas!
    - Deve-se usar mais bits do que número mínimo necessário para representar todos os dados possíveis (**redundância**)
    - Ex.: código 1-de-10 tem 10 palavras válidas dentre 1024
- **Problema**: projete um código otimizado para:
  - Representar 4 símbolos diferentes (ex.: A, T, C, G), e
  - Detectar qualquer erro simples (i.e., em apenas 1 bit)

Detecção de erros

- **Problema**: projete um código otimizado para:
  - Representar 4 símbolos diferentes: quantos bits?
    - Pelo menos 2 bits
  - Detectar qualquer erro simples (i.e., em apenas 1 bit) ➔ quantos bits adicionais...?
    - Vamos tentar com 1 bit para ver se funciona...
    - Palavras possíveis (3 bits): 000, 001, 010, 011, 100, 101, 110, 111
  - Quais palavras definir como válidas/inválidas...?

1ª tentativa:
menores valores



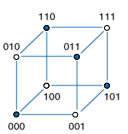


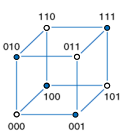
2ª tentativa:
valores distantes por
pelo menos 2 vértices

Detecção de erros: um pouco de teoria

- Código com **distância** mínima *m*: quaisquer pares de palavras do código diferem em pelo menos *m* bits
  - Conseguem detectar erros em até $d = m - 1$ bits!
- Para detectar todos os **erros de 1 bit**: $m = 2$
  - Para *n* bits de informação, é necessário um código de *n*+1 bits: bit adicional denominado "bit de paridade"
    - **Paridade par**: bit adicional é 1 se isso faz com que palavra tenha **número par de bits 1**; caso contrário, esse bit é 0
    - **Paridade ímpar**: bit adicional é 1 se isso se isso faz com que palavra tenha **número ímpar de bits 1**; caso contrário, esse bit é 0
  - Também permite detectar erros em **número ímpar de bits**

Paridade par





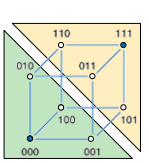
Paridade ímpar

Correção de erros

- Como corrigir (não apenas detectar) erros?
  - Ideia: podemos corrigir uma palavra inválida para a palavra de código válida mais próxima dela!
- É possível corrigir erros de 1 bit com um código de distância 2?
  - Não: palavras inválidas são equidistantes das palavras válidas

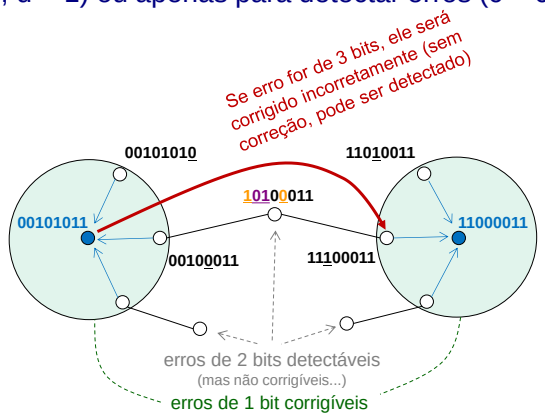
000 ←? 010 →? 110

- E se a distância for 3?
  - Sim: 010, 100, 001 ➔ 000
    110, 011, 101 ➔ 111



Correção de erros: um pouco de teoria

- Código com distância mínima m = 2*c*+*d*+1: corrige até *c* erros e é capaz de detectar *d* erros
  - Ex.: m = 4 ➔ receptor pode ser configurado para corrigir erros (c = 1, d = 1) ou apenas para detectar erros (c = 0, d = 3)



Código de Hamming

- Código com as seguintes características:
  - Distância mínima m = 3
  - Palavras de até (2^i –1) bits, dentre eles *i* bits de verificação
- Método de construção:
  - Enumere os bits de 1 a 2^i –1
  - Posições que são potências de 2 são bits de paridade *p*
    - Ou seja, pos(p) = 2^n, para 0 ≤ n < i
  - Cada bit de paridade *p* abrange todos os bits para os quais o AND lógico da posição de *p* e do bit de informação for ≠ 0
    - Ex. (n = 1): O bit de paridade na posição 2 é calculado considerando os bits informação nas posições 3 (0**1**1), 6 (1**1**0) e 7 (1**1**1)

posições:	7:111	6:110	5:101	4:100	3:011	2:010	1:001
paridade sobre:				5,6,7		3,6,7	3,5,7

bits de paridade

Código de Hamming

- A distância é no mínimo 3 porque
  - Trocar 1 bit na posição *j* qualquer leva a palavra inválida: posição j está associada a pelo menos um bit de paridade
  - Trocar 2 bits nas posições *j* e *k* também: bits de paridade envolvendo *j* **e** *k* não detectam erro, mas existe ao menos um bit de paridade que não depende de ambos *j* e *k*
    - Afinal, *j* e *k* diferem em pelo menos 1 bit

posições:	7:111	6:110	5:101	4:100	3:011	2:010	1:001
paridade sobre:	*		*	5,6,7		3,6,7	3,5,7

Exemplo:

erro em j = 7 ➔ invalida bits de paridade nas posições 4, 2 e 1
erro em j = 7 e k = 5➔ invalida bit de paridade na posição 2

Código de Hamming

- Correção de erros de 1 bit é simples (c = 1, d =0) :
  - Posição do bit em que houve a inversão é dada pela representação binária dos bits de paridade

posições:	7:111	6:110	5:101	4:100	3:011	2:010	1:001
paridade sobre:	*		*	5,6,7		3,6,7	3,5,7

Posição do erro	Bits de paridade afetados	Posição do erro	Bits de paridade afetados
7	4+2+1 = 7	3	2+1 = 3
6	4+2 = 6	2	2
5	4+1 = 5	1	1
4	4 = 4		

=

Código de Hamming

- Alguns detalhes adicionais
  - Distância 3 pode ser **estendida para distância 4**: basta adicionar um **bit de paridade calculado sobre todos os bits**
    - Obtém-se: (c = 1, d = 1), ou então (c=0, d=3)
  - Normalmente, em uma comunicação os bits de paridade são colocados nas **posições menos significativas** da palavra
    - Ou seja: bit de paridade da posição 2^i colocado na posição i

	Código de distância mínima 3		**Código de distância mínima 4**	
Bits de dados	Bits de paridade	Bits totais	Bits de paridade	Bits totais
1	2	3	3	4
≤ 4	3	≤ 7	4	≤ 8
≤ 11	4	≤ 15	5	≤ 16
≤ 26	5	≤ 31	6	≤ 32
≤ 57	6	≤ 63	7	≤ 64
≤ 120	7	≤ 127	8	≤ 128

Código de Hamming: Exercícios

1) Qual o Código de Hamming (distância mínima 3) com paridade par que representa a cadeia de informação 0101? E no caso do código de Hamming com distância 4?

2) Se os bits de paridade nas posições 1, 2 e 8 no código com distância 3 indicam erro, qual bit está errado?

Código de Hamming: Exercícios

1) Qual o Código de Hamming (distância mínima 3) com paridade par que representa a cadeia de informação 0101? E no caso do código de Hamming com distância 4?

➔ Comece construindo o código da direita para a esquerda, preenchendo os bits de informação e saltando os de paridade

➔ Preencha os bits de paridade usando a regra de abrangência previamente apresentada

7:111 6:110 5:101 **4**:100 3:011 **2**:010 **1**:001 **0**: 000 ← todos os bits

d4	d3	d2	p3	d1	p2	p1	p0
0	1	0	1	1	0	1	0

2) Se os bits de paridade nas posições 1, 2 e 8 no código com distância 3 indicam erro, qual bit está errado?

➔ 1+2+8 = 11

Código de Hamming: Exercícios

3) Envia-se a cadeia "0101101" (os 3 últimos bits são de paridade par, usando Hamming com distância mínima 3). Porém, o receptor recebe "0011101". O receptor é capaz de detectar e/ou corrigir esse erro?

Código de Hamming: Exercícios

3) Envia-se a cadeia "0101101" (os 3 últimos bits são de paridade par, usando Hamming com distância mínima 3). Porém, o receptor recebe "0011101". O receptor é capaz de detectar e/ou corrigir esse erro?

➔ **Resposta teórica**: foram 2 bits de erro: 0**01**1101, portanto tentativas de correção darão errado.

m = 2c + d + 1 = 3 ➔ c = 1, d = 0 (corrigem-se erros de 1 bit, mas se houver mais erros, não é possível detectar que a correção falhou)

Obs.: seria possível detectar o erro se não fosse feita qualquer tentativa de correção pois o código é inválido!

m = 2c + d + 1 = 3 ➔ c = 0, d = 2

Código de Hamming: Exercícios

3) Envia-se a cadeia "0101101" (os 3 últimos bits são de paridade par, usando Hamming com distância mínima 3). Porém, o receptor recebe "0011101". O receptor é capaz de detectar e/ou corrigir esse erro?

➔ **Resposta prática**: ordenando os bits de paridade nas posições que são potência de 2, para facilitar a visualização:

7:111	6:110	5:101	**4**:100	3:011	**2**:010	**1**:001
d4	d3	d2	p3	d1	p2	p1
0	0	1	1	1	0	1

Avaliando os bits de paridade com erro:

p3: 0⊕0⊕1⊕1 = 0 (OK); p2: 0⊕0⊕1⊕0 = 1 (erro); p1: 0⊕1⊕1⊕1 = 1 (erro)

A correção feita no receptor seria no bit 1+2 = 3 ➔ alterar d3. Assim, o receptor transforma "0011101" no código "0111101", que é válido mas não foi o código enviado pelo emissor...

Código de Hamming: Exercícios

4) Repita o exercício anterior, para um código de Hamming com distância 4: “01011010” é transformado em “00111010”

Código de Hamming: Exercícios

4) Repita o exercício anterior, para um código de Hamming com distância 4: “01011010” é transformado em “00111010”

➔ **Resposta teórica**: foram 2 bits de erro: 0**01**1101, portanto tentativas de correção darão errado, mas serão detectadas.

m = 2c + d + 1 = 4 ➔ c = 1, d = 1 (corrigem-se erros de 1 bit, e se ainda houver um erro adicional em 1 bit, é possível detectar que a correção falhou)

Código de Hamming: Exercícios

4) Repita o exercício anterior, para um código de Hamming com distância 4: "01011010" é transformado em "00111010"

➔ **Resposta prática**: ordenando os bits de paridade nas posições que são potência de 2, para facilitar a visualização:

7:111	6:110	5:101	**4**:100	3:011	**2**:010	**1**:001	0
d4	d3	d2	p3	d1	p2	p1	p0
0	0	1	1	1	0	1	0

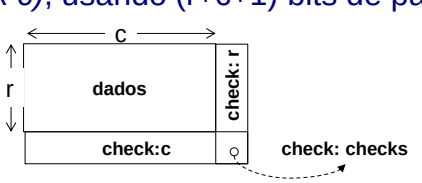
Avaliando os bits de paridade com erro:

p3: 0⊕0⊕1⊕1 = 0 (OK); p2: 0⊕0⊕1⊕0 = 1 (erro); p1: 0⊕1⊕1⊕1 = 1 (erro)

A correção feita no receptor seria no bit 1+2 = 3 ➔ alterar d3. Assim, o receptor transforma "00111010" no código "01111010". Porém, esse código não é válido, pois o bit de paridade p0 está incorreto. Logo, o receptor sabe que houve pelo menos 2 bits de erro.

Detecção/Correção de erros: outros

- Checksum: soma dos valores de grupamentos de bits (e.g., bytes); **detecção** de erro se soma incorreta
  - Comum em protocolos de comunicação, como TCP/IP
- CRC (Cyclic Redundancy Check): cálculo do resto da divisão; **detecção** de erro se resto incorreto
  - Obs.: cálculos são feitos usando teoria de corpos finitos em base 2, nos quais os números são vistos como polinômios
- Códigos bidimensionais: organizam bits de informação em matriz (*r x c)*, usando (r+c+1) bits de paridade;



Gerador/Detector de Paridade

- Recap: portas lógicas de OU-exclusivo (XOR)
  - $X \oplus Y = (X' \cdot Y) + (X \cdot Y')$
  - Resultados (equivalentes)
    - 1 se **apenas uma** das entradas for 1, 0 caso contrário
    - 1 se **ambas as entradas são diferentes**, 0 caso contrário

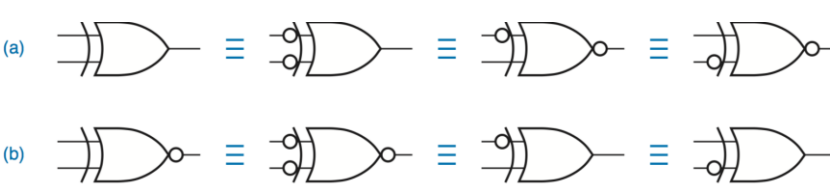
▪ Tabela-verdade

X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0

▪ Porta lógica
(representação gráfica)

Gerador/Detector de Paridade

- Símbolos alternativos para XOR (a) e XNOR (b)



Gerador/Detector de Paridade

- Exemplo: 2 bits
- $\sum_{odd}$: indica que entrada tem número ímpar de bits 1

iguais

X	Y	X⊕Y	$\sum_{odd}$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

- $\sum_{even}$: indica que entrada tem número par de bits 1

X	Y	(X⊕Y)'	$\sum_{even}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	1	1

Gerador/Detector de Paridade

- E para mais bits...?
  - Ex.: 3 bits

X	Y	Z			$\sum_{odd}$	# 1s
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

Gerador/Detector de Paridade

- E para mais bits...?
  - Ex.: 3 bits

X	Y	Z	W=Y⊕Z		$\sum_{odd}$	# 1s
0	0	0	0		0	0
0	0	1	1		1	2
0	1	0	1		1	2
0	1	1	0		0	2
1	0	0	0		1	2
1	0	1	1		0	2
1	1	0	1		0	2
1	1	1	0		1	4

Gerador/Detector de Paridade

- E para mais bits...?
  - Ex.: 3 bits

X			W=Y⊕Z	X⊕W	$\sum_{odd}$	# 1s
0			0	0	0	0
0			1	1	1	2
0			1	1	1	2
0			0	0	0	2
1			0	1	1	2
1			1	0	0	2
1			1	0	0	2
1			0	1	1	4

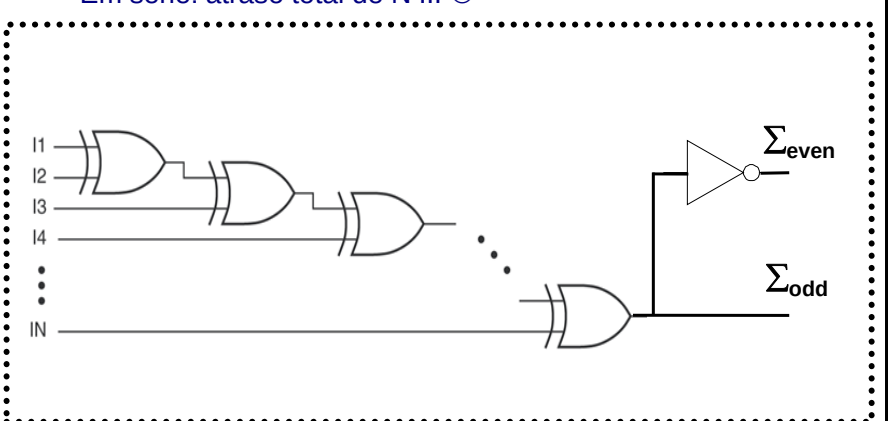
Gerador/Detector de Paridade

- E para mais bits...?
  - Basta cascatear os geradores de paridade de 2 bits!

X	Y	Z	W=Y⊕Z	X⊕W	$\sum_{odd}$	# 1s
0	0	0	0	0	0	0
0	0	1	1	1	1	2
0	1	0	1	1	1	2
0	1	1	0	0	0	2
1	0	0	0	1	1	2
1	0	1	1	0	0	2
1	1	0	1	0	0	2
1	1	1	0	1	1	4

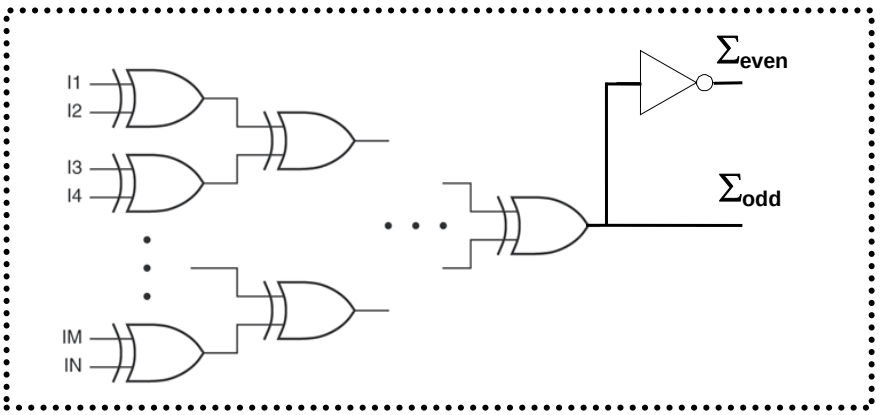
Gerador/Detector de Paridade

- E para mais bits...?
  - Basta cascatear os geradores de paridade de 2 bits!
  - Em série: atraso total de N ... ☹



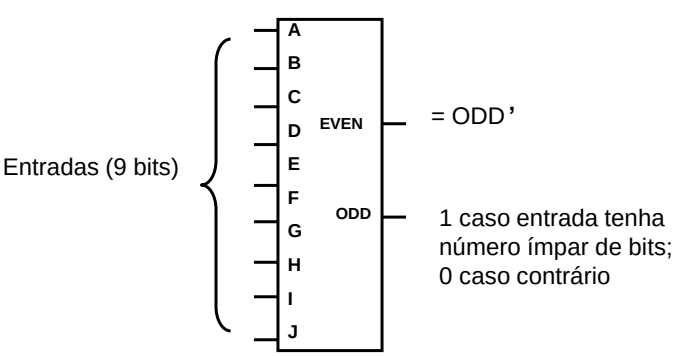
Gerador/Detector de Paridade

- E para mais bits...?
  - Basta cascatear os geradores de paridade de 2 bits!
  - Em árvore: atraso de ~lg(N) ☺



Gerador/Detector de Paridade (PAR)

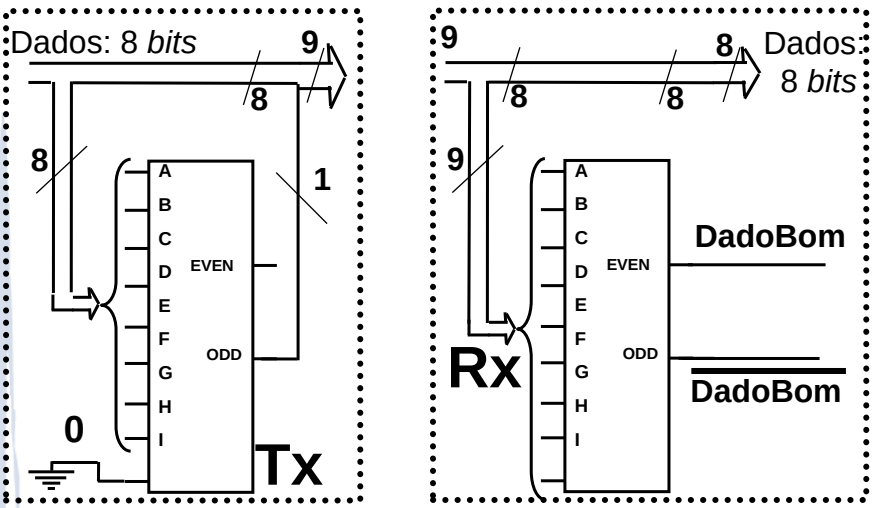
- Bloco gerador/detector de paridade



Gerador/Detector de Paridade

Notação: paridade par

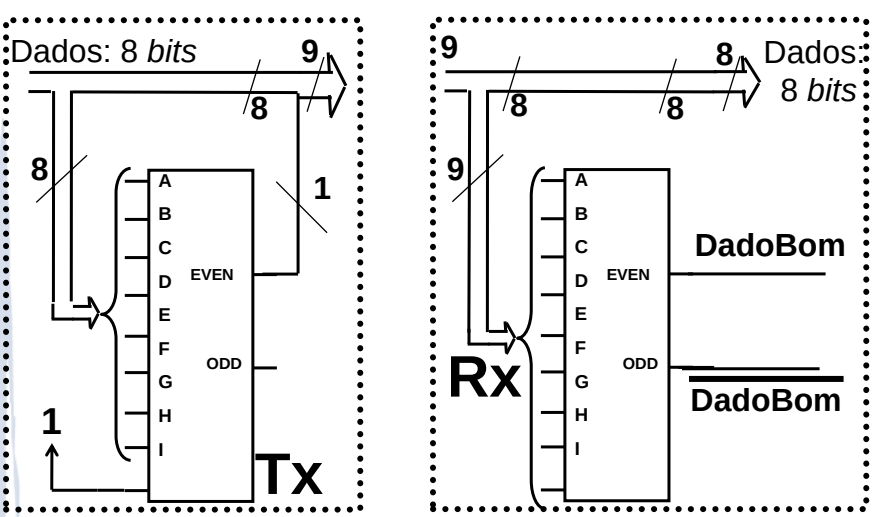
Transmissão de **8 *bits*** e **verificação** na **Recepção [1/2]**:



Gerador/Detector de Paridade

Notação: paridade par

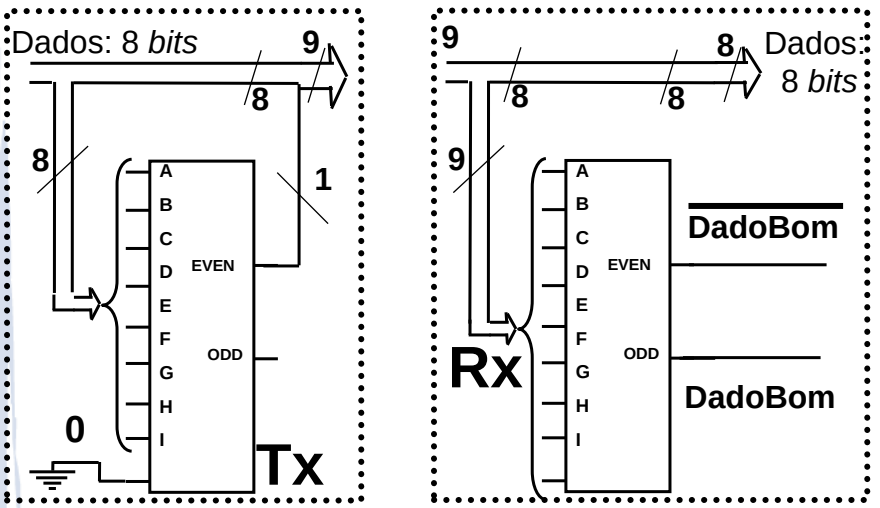
Transmissão de **8 *bits*** e **verificação** na **Recepção [2/2]**:



Gerador/Detector de Paridade

Notação: paridade <u>ímpar</u>

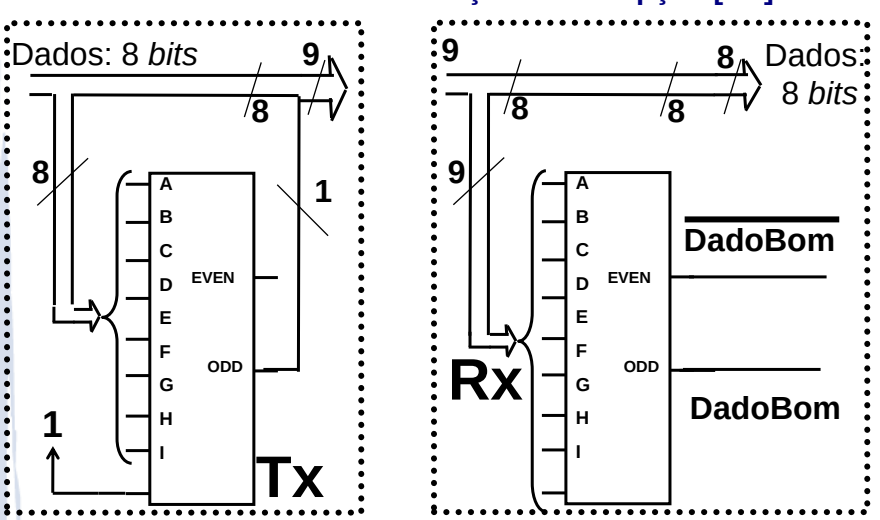
Transmissão de **8 *bits*** e **verificação** na **Recepção [1/2]**:



Gerador/Detector de Paridade

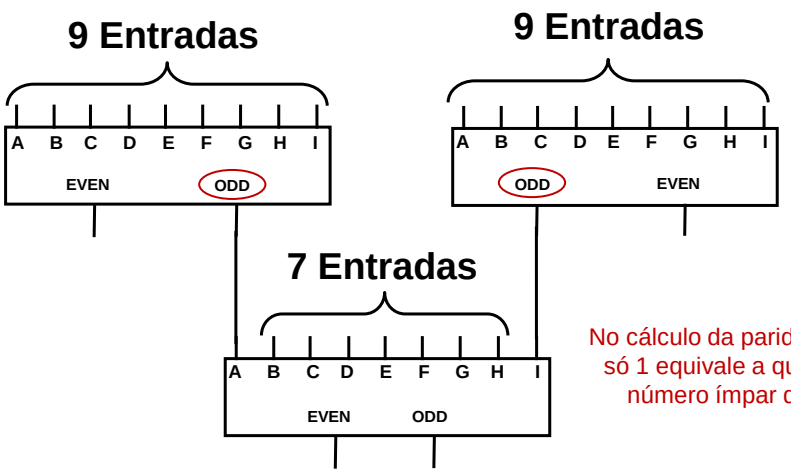
Notação: paridade <u>ímpar</u>

Transmissão de **8 *bits*** e **verificação** na **Recepção [2/2]**:



Gerador/Detector de Paridade

- **Associação** de Blocos Calculadores de Paridade:



No cálculo da paridade, um só 1 equivale a qualquer número ímpar de 1s

Gerador/Detector de Paridade

- XOR de 3 bits em VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity vxor3 is
    port (
        A, B, C: in STD_LOGIC;
        Y: out STD_LOGIC
    );
end vxor3;

architecture vxor3 of vxor3 is
begin
  Y <= A xor B xor C;
end vxor3;
```

Gerador/Detector de Paridade

```
library IEEE;
use IEEE.std_logic_1164.all;

entity V74x280 is
    port (
        I: in STD_LOGIC_VECTOR (1 to 9);
        EVEN, ODD: out STD_LOGIC
    );
end V74x280;

architecture V74x280s of V74x280 is
component vxor3
 port (A, B, C: in STD_LOGIC; Y: out STD_LOGIC);
end component;
signal Y1, Y2, Y3, Y3N: STD_LOGIC;
begin
  U1: vxor3 port map (I(1), I(2), I(3), Y1);
  U2: vxor3 port map (I(4), I(5), I(6), Y2);
  U3: vxor3 port map (I(7), I(8), I(9), Y3);
  Y3N <= not Y3;
  U4: vxor3 port map (Y1, Y2, Y3, ODD);
  U5: vxor3 port map (Y1, Y2, Y3N, EVEN);
end V74x280s;
```

Verificador de paridade de 9 bits: abordagem **estrutural**

Gerador/Detector de Paridade

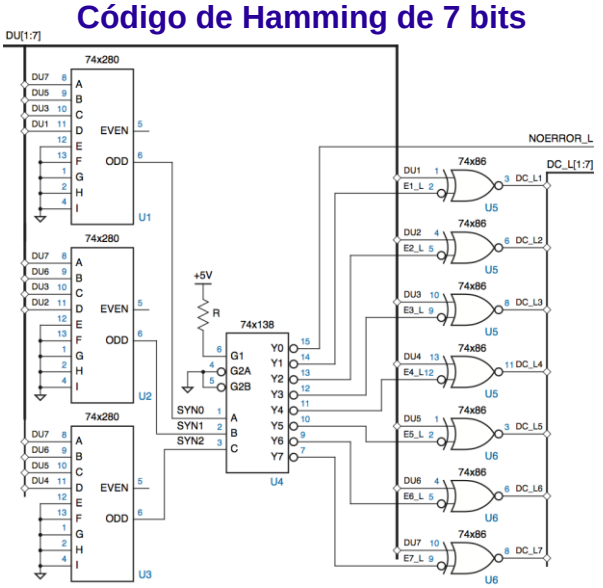
- **Código de Hamming:**
  - Basta fazer a interconexão correta entre os bits de entrada que entram na composição de cada bit de paridade

posições:	7:111	6:110	5:101	4:100	3:011	2:010	1:001
paridade sobre:				5,6,7		3,6,7	3,5,7

Posição do erro	Bits de paridade afetados	Posição do erro	Bits de paridade afetados
7	4+2+1 = 7	3	2+1 = 3
6	4+2 = 6	2	2
5	4+1 = 5	1	1
4	4 = 4		

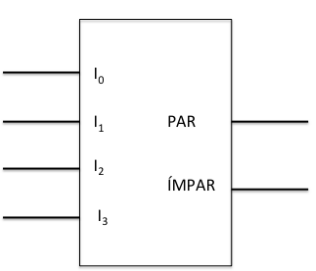
Gerador/Detector de Paridade

Código de Hamming de 7 bits

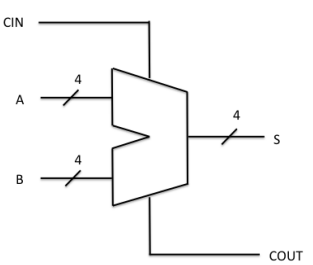


Exercício (PREC 2016)

- Seja A um número binário de 4 bits. Projete um circuito que calcule $A - 1_{10}$ caso A tenha paridade par e $A - 2_{10}$ caso A tenha paridade ímpar. A subtração deve ser calculada em Complemento de 2. Utilize (somente!) um gerador de paridade de 4 bits (Figura 1) e um somador completo de 4 bits (Figura 2).



Gerador de Paridade



Somador Completo de 4 bits

Exercícios: Códigos

- Distância de códigos
  - Bi-dimensional :
    - Distância = 3

1	**0**	1
0	**1**	1
1	1	0

1 bit

1	**1**	0
0	**1**	1
1	0	0

2 bits

0	**1**	1
0	**1**	1
0	0	0

  - Repetição anti-burst :
    - Exemplo (repetição x4): 10010 ➔ 10010 10010 10010 10010
      10011 ➔ 10011 10011 10011 10011
    - Distância = 4 (=número de repetições)
  - Bit de paridade repetido:
    - Ex. (repetição x4): 1100101111
    - Distância = 2

1 bit → 1100110000 **dist = 5**

2 bits → 1100011111 **dist = 2**

Exercício (PSUB 2017)

- Um sistema usa códigos de Hamming para corrigir erros de transmissão. Em um certo momento, deseja-se enviar a sequência de bits “1010”. Responda:

a) Qual o número mínimo de bits que serão necessários para representar a palavra de código resultante caso a distância de código desejada seja 3? Qual a palavra de código resultante, considerando paridade par?

b) Qual o número mínimo de bits que serão necessários para representar a palavra de código resultante caso a distância de código desejada seja 4? Qual a palavra de código resultante, considerando paridade par?

c) Suponha que seja usado o código com distância 3, como no item (a), que a palavra recebida tenha o número de bits determinado naquele item e que todos esses bits sejam 1s exceto pelo bit mais significativo, que tem valor 0. Por exemplo, se sua resposta no item (a) foi que são necessários 4 bits para representar a informação, então a palavra recebida foi “0111”. Essa palavra código é válida? Caso não seja, para qual palavra código ela será corrigida de acordo com o método de correção de Hamming?

Exercício (PSUB 2017)

- Um sistema usa códigos de Hamming para corrigir erros de transmissão. Em um certo momento, deseja-se enviar a sequência de bits “1010”. Responda:

a) Qual o número mínimo de bits que serão necessários para representar a palavra de código resultante caso a distância de código desejada seja 3? Qual a palavra de código resultante, considerando paridade par?

7 bits	7	6	5	4	3	2	1
Palavra:	1	0	1	**0**	0	**1**	**0**
	1	0	1	**0**			
	1	0			0	**1**	
	1		1		0		**0**
Também aceito:	1	0	1	0	**0**	**1**	**0**
						paridade	

Exercício (PSUB 2017)

- Um sistema usa códigos de Hamming para corrigir erros de transmissão. Em um certo momento, deseja-se enviar a sequência de bits “1010”. Responda:

b) Qual o número mínimo de bits que serão necessários para representar a palavra de código resultante caso a distância de código desejada seja 4? Qual a palavra de código resultante, considerando paridade par?

7 bits	7	6	5	4	3	2	1	0
Palavra:	1	0	1	**0**	0	**1**	**0**	**1**
	1	0	1	**0**				
	1	0			0	**1**		
	1		1		0		**0**	
Também aceito:	1	0	1	0	**0**	**1**	**0**	**1**
						paridade		

Exercício (PSUB 2017)

c) Suponha que seja usado o código com distância 3, como no item (a), que a palavra recebida tenha o número de bits determinado naquele item e que todos esses bits sejam 1s exceto pelo bit mais significativo, que tem valor 0. Por exemplo, se sua resposta no item (a) foi que são necessários 4 bits para representar a informação, então a palavra recebida foi "0111". Essa palavra código é válida? Caso não seja, para qual palavra código ela será corrigida de acordo com o método de correção de Hamming?

7 bits	7	6	5	4	3	2	1
Palavra:	0	1	1	**1**	1	**1**	**1**
	0	1	1	**0 (erro)**			
	0	1			1	**0 (erro)**	
	0		1		1		**0 (erro)**

Erro na posição 4+2+1 = 7 ➔ correção para 1111111

Exercício (PREC 2017)

- Deseja-se implementar um sistema que tenha suporte a números de 0 a 7, usando um código numérico. Preencha a seguinte tabela de códigos para cada um dos digitos possíveis, usando o código indicado na primeira linha da tabela.

Dígito Decimal	Bits correspondentes a **código BCD**	Bits para **código Gray**	**Bit de paridade par** para código BCD	**Bit de paridade par** para código Gray
0	000			
1				
2				
3				
4				
5				
6				
7				

Exercício (PREC 2017)

- Deseja-se implementar um sistema que tenha suporte a números de 0 a 7, usando um código numérico. Preencha a seguinte tabela de códigos para cada um dos digitos possíveis, usando o código indicado na primeira linha da tabela.

Dígito Decimal	Bits correspondentes a **código BCD**	Bits para **código Gray**	**Bit de paridade par** para código BCD	**Bit de paridade par** para código Gray
0	000	000	0	0
1	001	001	1	1
2	010	011	1	0
3	011	010	0	1
4	100	110	1	0
5	101	111	0	1
6	110	101	0	0
7	111	100	1	1

Exercício (PSUB 2017)

- Tentando interceptar uma mensagem do inimigo, obteve-se um conjunto de números e, com o histórico, um conjunto de regras de decodificação. Os caracteres recebidos foram:

50	57	36	43	51	51	20	20	73	83	32	79	72	59	6A	29

- As regras históricas dizem que as mensagens são transmitidas de quatro em quatro caracteres, usando números em diferentes bases; todos os números são convertidos em Hexadecimal para a conversão em caracteres do código ASCII, sempre utilizando apenas letras maiúsculas. Responda usando as seguintes regras

Exercício (PSUB 2017)

- O primeiro conjunto de oito números refere-se a quatro caracteres. Os números estão representados em **octal** de 2 dígitos e cada caractere é obtido pela soma dos números dados, dois a dois.

50	57	36	43	51	51	20	20	73	83	32	79	72	59	6A	29

$Octal_1$	$Octal_2$	Soma Octal	Hexa
50	57		
36	43		
51	51		
20	20		

Exercício (PSUB 2017)

- O primeiro conjunto de oito números refere-se a quatro caracteres. Os números estão representados em **octal** de 2 dígitos e cada caractere é obtido pela soma dos números dados, dois a dois.

50	57	36	43	51	51	20	20	73	83	32	79	72	59	6A	29

$Octal_1$	$Octal_2$	Soma Octal	Hexa
50	57	127	57
36	43	101	41
51	51	122	52
20	20	040	20

Exercício (PSUB 2017)

- O segundo conjunto está representado em decimal

50	57	36	43	51	51	20	20	73	83	32	79	72	59	6A	29

Decimal	73	83	32	79
Hexa				

- O terceiro está representado em base duodecimal (12)

Duodecimal	72	59	6A	29
Decimal				
Hexa				

Exercício (PSUB 2017)

- O segundo conjunto está representado em decimal

50	57	36	43	51	51	20	20	73	83	32	79	72	59	6A	29

Decimal	73	83	32	79
Hexa	49	53	20	4F

- O terceiro está representado em base duodecimal (12)

Duodecimal	72	59	6A	29
Decimal	86	69	82	33
Hexa	56	45	52	21

Exercício (PSUB 2017)

- Sabe-se que o caractere $(0010\ 0000)_2 = 32_{10} = 20_{16} =$ $(\text{espaço})_{ASCII}$ e que $(0010\ 0001)_2 = 33_{10} = 21_{16} = !_{ASCII}$ determina o final da mensagem.
- É preciso converter todos os conjuntos de caracteres para a base hexadecimal e consultar a tabela ASCII não dada) para a construção da mensagem toda. Sabe-se que as letras maiúsculas estão em ordem alfabética na tabela e que a primeira é o $A = 41_{HEX,}$

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

Exercício (PSUB 2017)

- Sabe-se que o caractere $(0010\ 0000)_2 = 32_{10} = 20_{16} =$ $(\text{espaço})_{ASCII}$ e que $(0010\ 0001)_2 = 33_{10} = 21_{16} = !_{ASCII}$ determina o final da mensagem.
- É preciso converter todos os conjuntos de caracteres para a base hexadecimal e consultar a tabela ASCII não dada) para a construção da mensagem toda. Sabe-se que as letras maiúsculas estão em ordem alfabética na tabela e que a primeira é o $A = 41_{HEX,}$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	X	Z							

Exercício (PSUB 2017)

- Mensagem de Hexa para ASCII:
  - 20_{16} = (espaço)$_{ASCII}$ e 21_{16} = ! $_{ASCII}$

1

$Octal_1$	$Octal_2$	Hexa
50	57	57
36	43	41
51	51	52
20	20	20

2

Decimal	73	83	32	79
Hexa	49	53	20	4F

3

Duodecimal	72	59	6A	29
Decimal	86	69	82	33
Hexa	56	45	52	21

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	X	Z							

➔ WAR IS OVER!

Exercício (PSUB 2017)

- Mensagem de Hexa para ASCII:
  - 20_{16} = (espaço)$_{ASCII}$ e 21_{16} = ! $_{ASCII}$

1

$Octal_1$	$Octal_2$	Hexa
50	57	57
36	43	41
51	51	52
20	20	20

2

Decimal	73	83	32	79
Hexa	49	53	20	4F

3

Duodecimal	72	59	6A	29
Decimal	86	69	82	33
Hexa	56	45	52	21

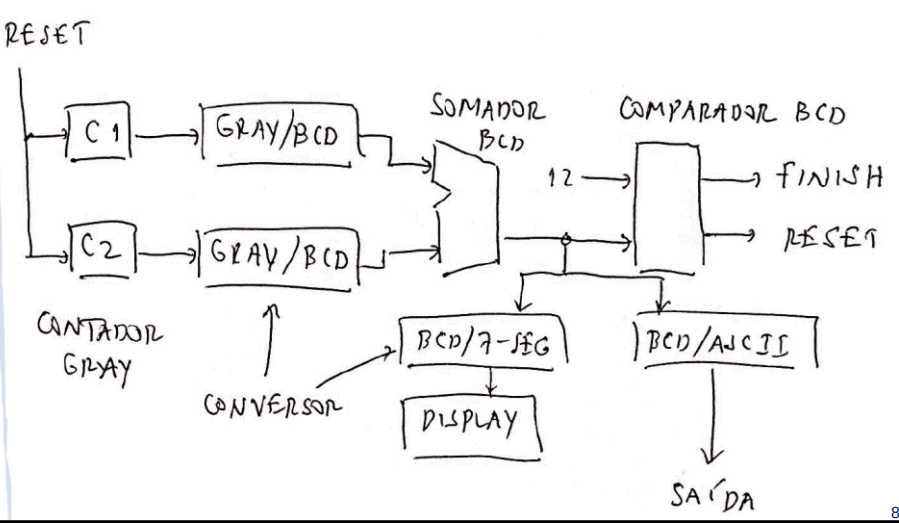
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
4		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	X	Z							

Exercício: Projeto

- Você é um engenheiro da ULABRAS, contratada por uma granja para projetar um sistema digital de controle para a colocação de ovos em embalagens contendo 1 dúzia de unidades. Os ovos são transportados por **2 esteiras**, que alimentam o embalador simultaneamente, em ritmo assíncrono. Na saída de cada esteira deve ser colocado um **contador Gray**, isto e, que conta em código Gray. Assuma que você terá a disposição o contador Gray projetado por outro engenheiro, mas para isso você precisara definir qual será o codigo Gray a ser usado. Represente a **quantidade de ovos colocados na embalagem em código BCD**. Quando a **soma dos ovos fornecidos pelas 2 esteiras resultar em 1 dúzia, o controlador deve emitir um sinal FINISH** de término da embalagem corrente e outro para reiniciar a contagem dos contadores Gray por meio de **um sinal RESET**. Também é preciso apresentar o **valor da contagem** em cada momento em displays de **7 segmentos** e numa sada paralela em **código ASCII**.

Exercício: Projeto

- Esboço de uma possível solução:



CÓDIGOS P/ TRANSMISSÃO E ARMAZENAMENTO SERIAIS

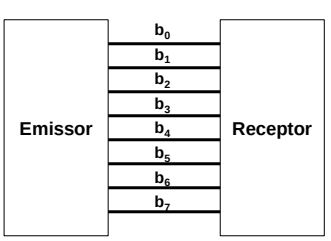
Paralelo vs. Serial

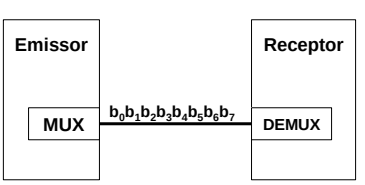
Comunicação paralela

- **Maior velocidade**
- **Maior custo** (cabos e circuitos de leitura/escrita)
- Ex.: impressoras, ATA, PCI

Comunicação serial

- **Menor velocidade**
- **Menor custo** (mas requer circuito (de)serializador)
- Ex.: USB, SATA, Firewire

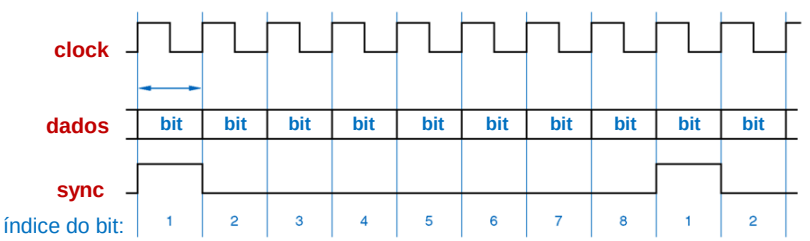




Transmissão serial

- Como saber quando um bit começa e termina?
  - Pode ser usado um sinal de relógio (clock) entre emissor e receptor, que dita a velocidade da transmissão (bit rate)
- E quando um conjunto de bits começa e termina?
  - Pode ser enviado um sinal de sincronismo (sync) indicando início e final da transmissão

3 fios… Sinais não podem ser combinado em 1 fio?

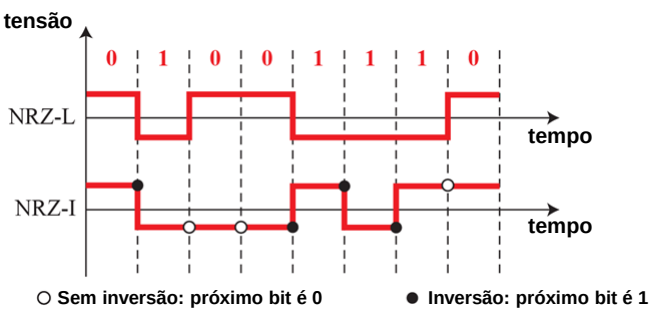


Transmissão serial

- Como saber quando um bit começa e termina?
  - Pode ser usado um sinal de relógio (clock) entre emissor e receptor, que dita a velocidade da transmissão (bit rate)
  - **OU**: o próprio sinal pode dar "dicas" da fronteira entre bits, interpretada por um circuito no receptor (denominado *digital phase-locked loop* – DPLL)
- E quando um conjunto de bits começa e termina?
  - Pode ser enviado um sinal de sincronismo (sync) indicando início e final da transmissão
  - **OU**: pode ser enviado repetidamente um sinal de "IDLE" entre transmissões de dados
    - Ex (Ethernet): uma longa sequência de pelo menos 7 bytes "10101010" indica que não há transmissão; a transmissão começa após um byte 10101011.

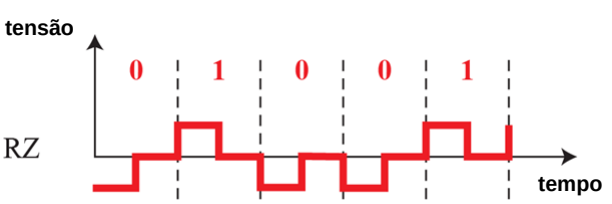
Transmissão serial: Exemplos

- Esquemas **polares**: codificação com tensão + ou –
  - **NRZ** (*Non-Return to Zero*): nível codifica valor do bit
    - **Con**: ainda **requer clock** entre emissor e receptor para identificar transição entre bits (pelo menos 2 fios)
    - **Con**: **nível DC** depende de equilíbrio entre 0s e 1s (ruim para longas distâncias: **atenuação**)



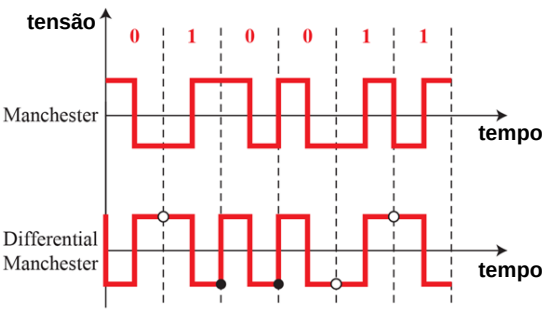
Transmissão serial: Exemplos

- Esquemas **polares**: codificação com tensão + ou –
  - **RZ** (*Return to Zero*): três valores
    - **+** (bit 0); **–** (bit 1); **neutro** (sincronismo apenas)
    - **Pró**: **não requer sincronismo** (sinal sempre se altera durante bit)
    - **Con**: ocupa **maior largura de banda** (1 bit = 2 alterações de sinal)
    - **Con**: **nível DC** depende de equilíbrio entre 0s e 1s



Transmissão serial: Exemplos

- Esquemas **polares**: codificação com tensão + ou –
  - **Manchester**: transição na metade do bit
    - Pró: não requer sincronismo (transição permite resincronização)
    - Pró: nível DC sempre 0
    - **Con**: ocupa **maior largura de banda** (1 bit = 2 alterações de sinal)

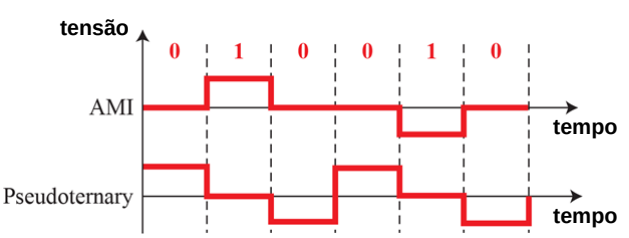


○ **Sem inversão do nível (inversão da borda): próximo bit é 1**

● **Inversão do nível (sem inversão da borda): próximo bit é 0**

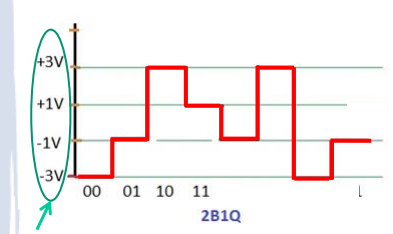
Transmissão serial: Exemplos

- Esquemas **bipolares**: três níveis para bits (+, –, neutro)
  - *Alternate Mark Inversion* (**AMI**): neutro=0; (+,– alternados)=1
  - **Pseudoternário**: neutro=1; (+,– alternados) = 0
  - Ambos:
    - **Prós**: **nível DC** aproximadamente 0 sempre.
    - **Cons**: **possível perda de sincronismo** após longa sequência de neutros

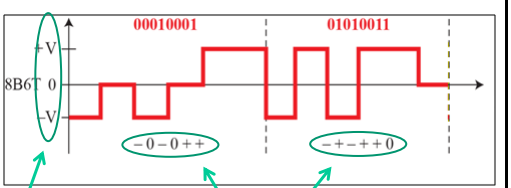


Transmissão serial: Exemplos

- Esquemas **multinível**: vários bits por nível/elemento de sinal (e.g., transição)
  - Notação: ***m*B*nL*** = ***m*** bits a cada ***n*** elementos de sinal, com ***L*** níveis de sinal (B: binário, T: ternário, Q: quaternário)
  - **2B1Q**: 2 bits a cada 1 elemento de sinal, com 4 níveis
  - **8B6T**: 8 bits a cada 6 elementos de sinal, com 3 níveis



4 níveis



3 níveis

$3^6 > 2^8$ códigos possíveis.
Redundância permite: (1) correção de erros
(2) combinações com componente DC = 0

Lição de Casa

- Leitura Obrigatória:
  - Capítulo 2 do Livro Texto.
- Exercícios Obrigatórios:
  - Capítulo 2 do Livro Texto;
  - Lista de Exercícios do Módulo 4.

Livro Texto

- Wakerly, J.F.; *Digital Design – Principles & Practices;* Fourth Edition, ISBN: 0-13-186389-4, Pearson & Prentice-Hall, Upper Saddle, River, New Jersey, 07458, 2006.

Bibliografia Adicional

- Giozza, William Ferreira; et all; *Redes Locais de Computadores: Tecnologia e Aplicações – Seção 3.2.3 Codificação em Banda Básica – Códigos*; Editora McGraw-Hill, 1.986;
- Hayes, J.P.; *Computer Architecture and Organization*; McGraw-Hill, 1988;