# PCS 3115 (PCS2215)

## Sistemas Digitais I

**Hamming e Correção de erros**

***Prof. Dr. Marcos A. Simplicio Jr.***

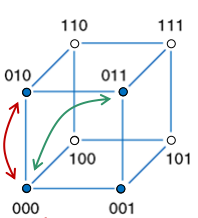
*versão: 4.1 (Mar/2020)*

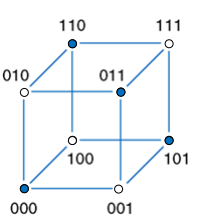
# Recapitulando: Detecção de erros

- **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





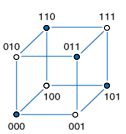



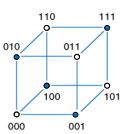

2ª tentativa:
valores distantes por
pelo menos 2 vértices

# Recapitulando: Detecção de erros (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



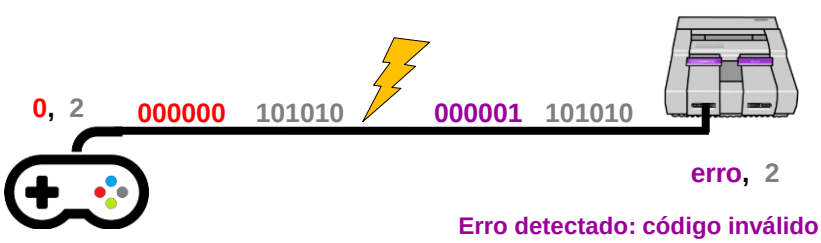




Paridade ímpar



# Correção de erros

- Como corrigir (não apenas detectar) erros?
  - Ideia: podemos corrigir uma palavra inválida para a palavra de código válida mais próxima dela!




**0: $000000_2$** **1: $010101_2$**
**2: $101010_2$** **3: $111111_2$**

**Erro detectado: código inválido**

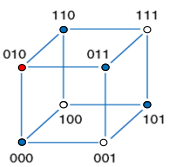
**Hipótese: foi só 1 bit de erro...**
**Qual o valor original?**

➔ **000000 (única possibilidade!)**

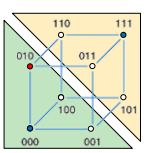
# Correção de erros

- 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? Ex.: 010
  - Não: palavras inválidas são equidistantes das palavras válidas

000 ← ? 010 → ? 110



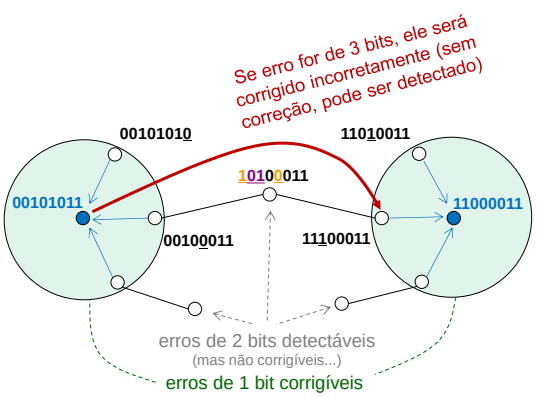

- E se a distância for 3?
  - Sim: 010, 100, 001 ➔ 000
    110, 011, 101 ➔ 111





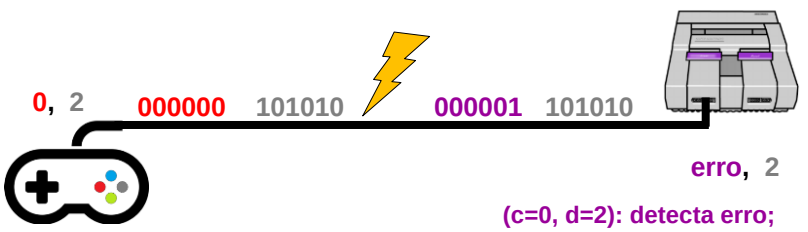

# Correção de erros: um pouco de teoria

- 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)



# Correção de erros

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




0: $000000_2$ ↔ (3) 1: $010101_2$
2: $101010_2$ ↔ (6) 3: $111111_2$

distância **mínima** (m): 3 bits

**(c=0, d=2): detecta erro; não tenta corrigir**

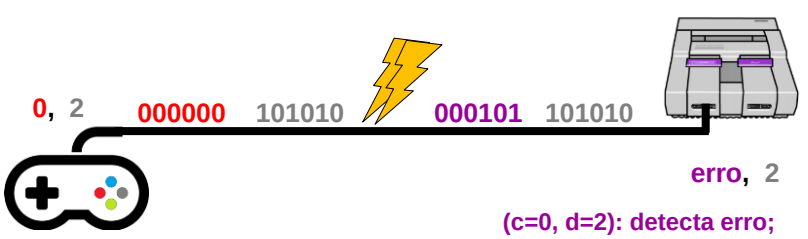
**(c=1, d=0): corrige para 000000**
**➔ e de fato levou a resultado desejado**

mas...



# Correção de erros

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




0: $000000_2$ ↔ (3) 1: $010101_2$
2: $101010_2$ ↔ (6) 3: $111111_2$

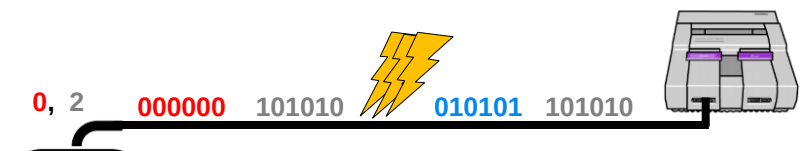
distância **mínima** (m): 3 bits

**(c=0, d=2): detecta erro; não tenta corrigir**

**(c=1, d=0): corrige para 010101**
**➔ não leva a resultado desejado, porque #erros > capacidade de correção (c)**

# Correção de erros

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




erro, 2

(c=0, d=2): não detecta erro;
#erros > capacidade de detecção (d)

(c=1, d=0): não corrige
➔ e não leva a resultado desejado, porque #erros > capacidade de correção (c)

0: $000000_2$ ⟷ (3) 1: $010101_2$
2: $101010_2$ ⟷ (6) 3: $111111_2$

distância **mínima** (m): 3 bits



# Correção de erros

- 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)


erro, 2

(c=0, d=3): detecta erro;
não tenta corrigir

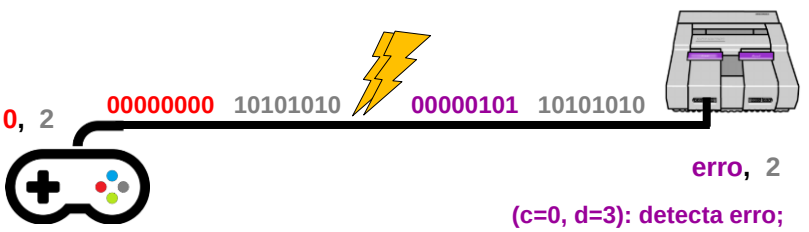
(c=1, d=1): corrige para 0000000
➔ e leva a resultado desejado

0: $00000000_2$ ⟷ (4) 1: $01010101_2$
2: $10101010_2$ ⟷ (8) 3: $11111111_2$

distância **mínima** (m): 4 bits

# Correção de erros

- 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)




0: $00000000_2$ ⟷ (4) 1: $01010101_2$
2: $10101010_2$ ⟷ (8) 3: $11111111_2$

distância **mínima** (m): 4 bits

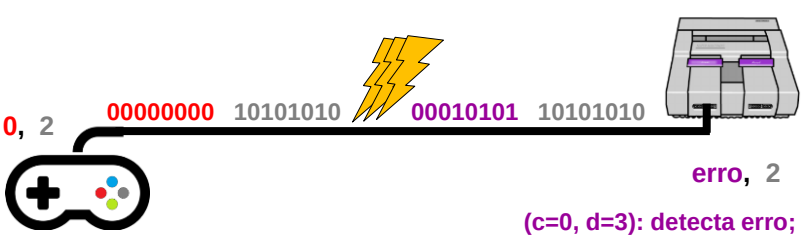
**(c=0, d=3): detecta erro; não tenta corrigir**

**(c=1, d=1): assuma erro, pois correção é dúbia -- pode ser p/ 0000000 ou p/ 01010101**
**➔ d é útil quando #erros − c ≤ d**



# Correção de erros

- 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)




0: $00000000_2$ ⟷ (4) 1: $01010101_2$
2: $10101010_2$ ⟷ (8) 3: $11111111_2$

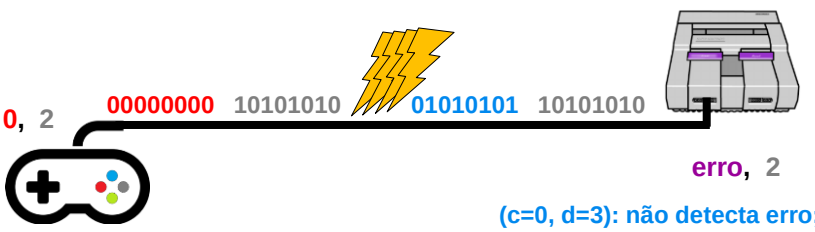
distância **mínima** (m): 4 bits

**(c=0, d=3): detecta erro; não tenta corrigir**

**(c=1, d=1): corrige p/ 01010101**
**➔ não leva a resultado desejado, e problema passa desapercebido porque #erros após correção > d**

# Correção de erros

- 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)




**0: $00000000_2$** ⟷ (4) **1: $01010101_2$**

**2: $10101010_2$** ⟷ (8) **3: $11111111_2$**

distância **mínima** (m): 4 bits

**(c=0, d=3): não detecta erro;**
**#erros > capacidade de detecção (d)**

**(c=1, d=1): não corrige**
**➔ e não leva a resultado desejado, porque #erros após correção > d**



# Devo corrigir ou detectar?

- Ou seja: ao usar código com distância m, dar foco em "c" ou em "d"?
- Resposta: decisão de projeto
  - Ex.: quando custo de retransmissão é baixo, não se corrigem erros (apenas detecta-se) ➔ TCP/IP na Internet
  - Ex.: quando custo de retransmissão é elevado, aplica-se correção ➔ transmissão via satélite

# 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 $0 \leq n < i$
  - 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. (4 bits, n = 1): Bit de paridade na posição 2 calculado com base nos bits de 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: Exemplo

- Transmissão de **0101** usando Hamming com distância 3:
  - Passo 1: Coloque os bits de dados $\mathbf{d_4d_3d_2d_1}$ nas posições que não são potências de 2
  - Passo2: Calcule os bits de paridade

| | 7:111 | 6:110 | 5:101 | 4:100 | 3:011 | 2:010 | 1:001 |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 0 | | 1 | | |
| $p_3$: | 0 | 1 | 0 | 1 | | | |
| $p_2$: | 0 | 1 | | | 1 | 0 | |
| $p_1$: | 0 | | 0 | | 1 | | 1 |
| | 0 | 1 | 0 | 1 | 1 | 0 | 1 |

**bits de paridade**

- Resultado: $\mathbf{d_4d_3d_2p_3d_1p_2p_1 = 0101101}$

  $\mathbf{d_4d_3d_2d_1p_3p_2p_1 = 0101101}$ ← *representação mais usual*

# Código de Hamming: Exemplo

- Transmissão de **10101** usando Hamming e distância 3:

| | 9:1001 | 8:1000 | 7:0111 | 6:0110 | 5:0101 | 4:0100 | 3:0011 | 2:0010 | 1:0001 |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | | 0 | 1 | 0 | | 1 | | |
| $p_4$: | 1 | 1 | | | | | | | |
| $p_3$: | | | 0 | 1 | 0 | 1 | | | |
| $p_2$: | | | 0 | 1 | | | 1 | 0 | |
| $p_1$: | 1 | | 0 | | 0 | | 1 | | 0 |
| | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |

bits de paridade

- Resultado: $d_4p_4d_4d_3d_2p_3d_1p_2p_1 = 1101011100$

  $d_4d_4d_3d_2d_1p_4p_3p_2p_1 = 1010111100$

  *representação mais usual* ↗



# Código de Hamming: racionalizando...

- 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: Receptor

- 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
  - Hardware: combinar bits de paridade com erro e inverter bit de entrada na posição indicada por eles

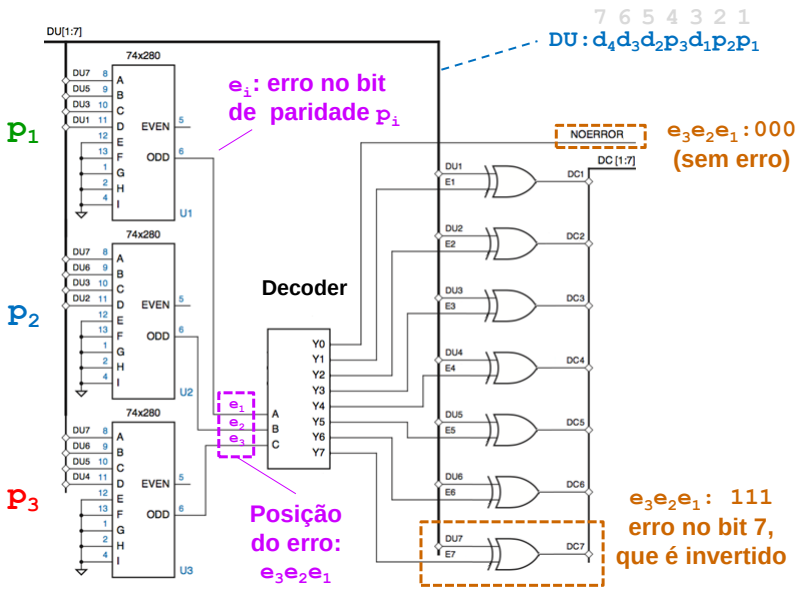
| posições: | 7:111 | 6:110 | 5:101 | 4:100 | 3:011 | 2:010 | 1:001 |
|---|---|---|---|---|---|---|---|
| paridade sobre: | * | | * | 5,6,7 | | 3,6,7 | 3,5,7 |

| 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 Hamming: 7 bits



# Código de Hamming

- Alguns detalhes adicionais
  - Distância 3 pode ser **estendida para distância 4**: basta adicionar um **bit de paridade calculado sobre <u>todos</u> os bits**
    - Com m = 4, pode-se usar (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 p1 e p3 (posições 1 e 4) 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 p1 e p3 (posições 1 e 4) no código com distância 3 indicam erro, qual bit está errado?

➔ 1+4 = 5



# 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?

| 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 e 4 no código com distância 3 indicam erro, qual bit está errado?

➔ 1+4 = 5 **Exemplo desse caso: dado recebido é 0111101**

| | 7:111 | 6:110 | 5:101 | 4:100 | 3:011 | 2:010 | 1:001 |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| $p_3$: | 0 | 1 | 1 | 0 | | | |
| $p_2$: | 0 | 1 | | | 1 | 0 | |
| $p_1$: | 0 | | 1 | | 1 | | 0 |
| | | | | ≠ recebido | | = recebido | ≠ recebido |
| | | | | ✗ | | ✓ | ✗ |

## 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 "0010101", 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 “00101010”. 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.

# 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 resultante: | 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 resultante: | 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

# Gerador/Detector de Paridade

- **Para 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 | | |



# Exercício (P2-2018)

- O seguinte código VHDL deveria verificar palavras de 8 bits usando código de Hamming com distância 4 e paridade par, agrupando os bits de paridade nas posições menos significativas (i.e., a partir do índice 0). Porém, ele não está funcionando. Aponte a(s) linha(s) onde há erros e explique o que está errado.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  entity receptorHamming is
4    port (code    : in std_logic_vetor(7 downto 0); -- palavra
5          notgood : out std_logic);           -- '1' se erro
6  end receptorHamming;
7  architecture hammingFlow of receptorHamming is
8  signal p3, p2, p1, p0 : std_logic;
9  begin
10    p3 <= code(7) xor code(6) xor code(5) xor code(3);
11    p2 <= code(7) xor code(6) xor code(4) xor code(2);
12    p1 <= code(7) xor code(5) xor code(4) xor code(1);
13    p0 <= code(3) xor code(2) xor code(1) xor code(0);
14    notgood <= p3 or p2 or p1 or p0;
15 end hammingFlow;
```

# Exercício (P2-2018)

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  entity receptorHamming is
4    port (code    : in std_logic_vetor(7 downto 0); -- palavra
5          notgood : out std_logic);           -- '1' se erro
6  end receptorHamming;
7  architecture hammingFlow of receptorHamming is
8  signal p3, p2, p1, p0 : std_logic;
9  begin
10    p3 <= code(7) xor code(6) xor code(5) xor code(3);
11    p2 <= code(7) xor code(6) xor code(4) xor code(2);
12    p1 <= code(7) xor code(5) xor code(4) xor code(1);
13    p0 <= code(3) xor code(2) xor code(1) xor code(0);
14    notgood <= p3 or p2 or p1 or p0;
15 end hammingFlow;
```

A linha 13 está errada: embora p3, p2 e p1 sejam calculados a partir dos bits corretos da palavra de entrada, p0 deveria ser calculado a partir de TODOS os bits da palavra, não apenas dos outros bits de paridade. O correto seria algo como "p0 <= code(7) xor code(6) xor code(5) xor code(4) xor code(3) xor code(2) xor code(1) xor code(0);".



# Gerador Hamming - VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
entity emissorHamming4 is
  port (d    : in  std_logic_vetor(4 downto 1);  -- dados
        code : out std_logic_vetor(7 downto 0)); -- palavra
end emissorHamming4;
architecture hammingFlow of emissorHamming4 is
signal p3, p2, p1, p0 : std_logic; -- distância 4
begin
  p3 <= d(4) xor d(3) xor d(2); -- posições 7, 6, 5
  p2 <= d(4) xor d(3) xor d(1); -- posições 7, 6, 3
  p1 <= d(4) xor d(2) xor d(1); -- posições 7, 5, 3
  p0 <= d(3) xor d(2) xor d(1) xor code(0) xor p3 xor p2 xor p1;
  code <= (d & p3 & p2 & p1 & p0); -- d4d3d2d1p3p2p1p0
end hammingFlow;
```

## Gerador Hamming - VHDL

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

entity emissorHamming3 is
  port (data : in  std_logic_vetor(4 downto 1);  -- dados
        code : out std_logic_vetor(7 downto 1)); -- palavra
end emissorHamming3;

architecture hammingFlow of emissorHamming3 is
signal p3, p2, p1 : std_logic; -- distância 3
begin
    p3 <= data(4) xor data(3) xor data(2); -- posições 7, 6, 5
    p2 <= data(4) xor data(3) xor data(1); -- posições 7, 6, 3
    p1 <= data(4) xor data(2) xor data(1); -- posições 7, 5, 3
    code <= (data & p3 & p2 & p1); -- d4d3d2d1p3p2p1
end hammingFlow;
```

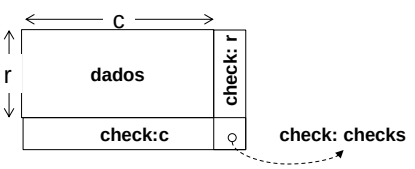


# APÊNDICE

Exemplos de outros códigos

# 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;






# 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 → 1100**1**10000 **dist = 5**

2 bits → 1100**01**1111 **dist = 2**