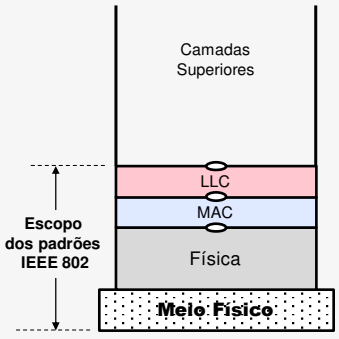
DEPARTAMENTO DE INFORMÁTICA
PUC-RIO
PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Detecção de Erros

Cálculo do CRC

TeleMídia

Nível de Enlace

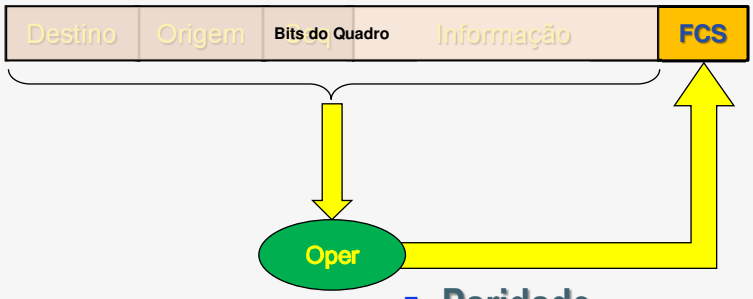


- Transmite e Recebe de quadros
  - *Quadro = PDU do Nível de Enlace*
  - *Delimitação de Quadros (Enquadramento)*
- Detecta erros
- Endereça
- Controla o Acesso ao Meio (MAC)

- Opcionalmente
  - *Corrige erros ou perdas*
  - *Controla o fluxo*

TeleMídia

Detecção de Erros: Operações



- Paridade
- CheckSum
- CRC

CRC (Cyclic Redundancy Check)

- Código de Redundância Cíclica.
- É comum recorrer-se a uma *representação polinomial* para as sequencias de bits
  - *utiliza-se polinômios de uma variável arbitrária X para de tal forma que a sequencia de bits $c_{n-1},\ldots c_1, c_0$ é representada pelo polinômio*

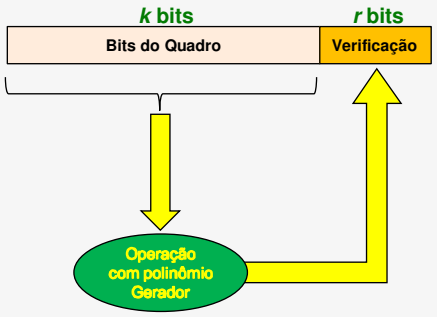
$$c(X) = c_{n-1}X^{n-1} + c_{n-2}X^{n-2} + \ldots + c_1X + c_0$$

  - Exemplo: A palavra 10110001 é representada pelo polinômio $X^7 + X^5 + X^4 + 1$.

• *Polinômios de grau n representam sequencias de n + 1 bits*

CRC

- Transmissor e receptor têm, previamente acordado, um padrão de bits cuja representação polinomial é conhecida pelo nome de ***polinômio gerador***.



CRC: Polinômio Gerador

- Alguns polinômios geradores são largamente utilizados e padronizados. Como exemplos, temos os seguintes:

$$CRC-12 = X^{12}+X^{11}+X^3+X^2+X+1$$

$$CRC-16 = X^{16}+X^{15}+X^2+1$$

$$CRC-CCITT = X^{16}+X^{12}+X^5+1$$

$$CRC-32 = X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^8+X^7+X^5+X^4+X^2+X+1$$

CRC: Polinômio Gerador

- **Para gerar uma sequencia de *r* bits, o grau do polinômio gerador deve ser *r*,**
  - *o tamanho da sequência de bits referente ao polinômio gerador será r + 1.*
    - O CRC-32, por exemplo,
      - *escolhido pelo IEEE 802 para ser utilizado em protocolos de redes locais,*
        - Polinômio de grau 32
          - Sequência de 33 bits
          - Gera FCS de 32 bits.

$$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X+1$$ **CRC-32**

Cálculo para Geração do CRC

- **Utiliza Aritmética em módulo 2.**

Aritmética em Módulo 2

- Adição bit a bit (representada pelo símbolo ⊕) é realizada sem se considerar o "vai um", como representado a seguir:

  0⊕0=0

  0⊕1=1

  1⊕0=1

  1⊕1=0

  - *Operação* **XOR**
  - *Como 1⊕1 = 0 , tem-se que, nessa aritmética, 1= −1.*
    - *Logo, a subtração módulo-2 é igual à soma.*

Aritmética em Módulo 2

- As regras para a multiplicação em aritmética módulo-2 são as mesmas da multiplicação tradicional:

  0×0=0

  0×1=0

  1×0=0

  1×1=1

  - *Operação* **AND**
- Tomando-se a divisão como a operação inversa da multiplicação e excluindo os casos de divisão por zero, obtém-se as seguintes operações possíveis:

  0÷1=0

  1÷1=1

Aritmética em Módulo 2 com Vários Algarismos

```
  1010        1010        110110
+ 0110      - 0110      110110
  1100        1100      101011110
```

```
  110110
× 1101
  110110
 000000
```


Divisão em Módulo 2 com Vários Algarismos

Dividendo	Divisor
101011000	1101

Divisão em Módulo 2 com Vários Algarismos

```
1010|11000 | 1101
```


Divisão em Módulo 2 com Vários Algarismos

```
1010|11000 | 1101
    1
```

Divisão em Módulo 2 com Vários Algarismos

```
1010 11000 | 1101
1101       |  1
 1111
```


Divisão em Módulo 2 com Vários Algarismos

```
1010 11000 | 1101
1101       |  11
 1111
```

Divisão em Módulo 2 com Vários Algarismos

```
1010 11000 | 1101
1101       |  11
 1111      |
 1101      |
  0101     |
```


Divisão em Módulo 2 com Vários Algarismos

```
1010 11000 | 1101
1101       |  110
 1111      |
 1101      |
  0101     |
```

Divisão em Módulo 2 com Vários Algarismos

```
1010 11000 | 1101
1101       | 110
 1111
 1101
  0101
  0000
   1010
```


Divisão em Módulo 2 com Vários Algarismos

```
1010 11000 | 1101
1101       | 1101
 1111
 1101
  0101
  0000
   1010
```

Divisão em Módulo 2 com Vários Algarismos

```
1010 11000 | 1101
1101       | 1101
 1111
 1101
  0101
  0000
   1010
   1101
    1110
```


Divisão em Módulo 2 com Vários Algarismos

```
1010 11000 | 1101
1101       | 11011
 1111
 1101
  0101
  0000
   1010
   1101
    1110
```

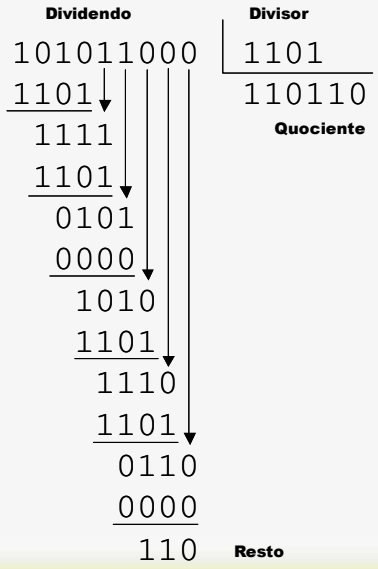
Divisão em Módulo 2 com Vários Algarismos

```
1010111000 | 1101
1101       | 11011
 1111
 1101
  0101
  0000
   1010
   1101
    1110
    1101
     0110
```


Divisão em Módulo 2 com Vários Algarismos

```
1010111000 | 1101
1101       | 110110
 1111
 1101
  0101
  0000
   1010
   1101
    1110
    1101
     0110
```

Divisão em Módulo 2 com Vários Algarismos

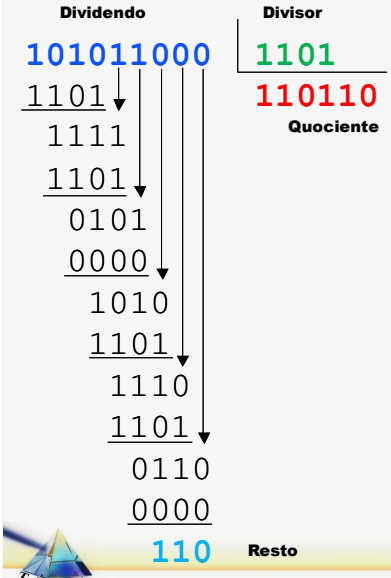


Propriedade da Divisão

$$\begin{array}{l|l} \textit{Dividendo} & \textit{Divisor} \\ \hline \textit{Resto} & \textit{Quociente} \end{array}$$

$$Dividendo = Quociente \times Divisor + Resto$$

Divisão em Módulo 2 com Vários Algarismos



$$Dividendo = Quociente \times Divisor + Resto$$

$$\begin{array}{r} 110110 \\ \times\ 1101 \\ \hline 110110 \\ 000000 \\ 110110 \\ 110110 \\ \hline 101011110 \\ +\ \quad 110 \\ \hline 101011000 \end{array}$$

Colocação de zeros a direita

- Em qualquer Base Numérica (B), acrescentar zero à direita significa multiplicar pela base
  - *Exemplo na base B = 10*
    - 3 acrescido de zero à direita = 30 (multiplica por 10)
  - *Exemplo na base B = 2*
    - 11 acrescido de zero à direita = 110
      - *3 vira 6 (multiplica por 2)*
- *Logo, colocar n zeros à direita corresponde a multiplicar por B^n*
  - *Na Base B = 2*
    - 1101 acrescido de 00 à direita = 110100
      - *13 vira 52 (13×2^2)*

Cálculo para Geração do CRC

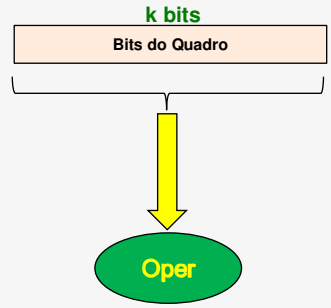
1. Toma-se a sequencia original (com *k* bits) do quadro e acrescenta-se *r* zeros à direita.
   - *corresponde à multiplicação do bloco original por 2^r,*
     - *r* é o número de bits previsto para o CRC.
     - Gera uma nova sequencia de *n* = (*k* + *r*) bits
2. A nova sequencia de *n* bits é então dividida, utilizando-se aritmética módulo-2, pela seqüência correspondente ao polinômio gerador.
   - *O resto dessa divisão, que conterá no máximo r bits, é a seqüência de verificação (**FCS**).*
   - *O FCS é então somado (em aritmética módulo-2) ao bloco de n bits anterior,*
     - Ou seja: os *r* zeros introduzidos são substituídos pelo *FCS* calculado.

Geração do CRC

Polinômio Gerador: $x^3 + x^2 + 1$

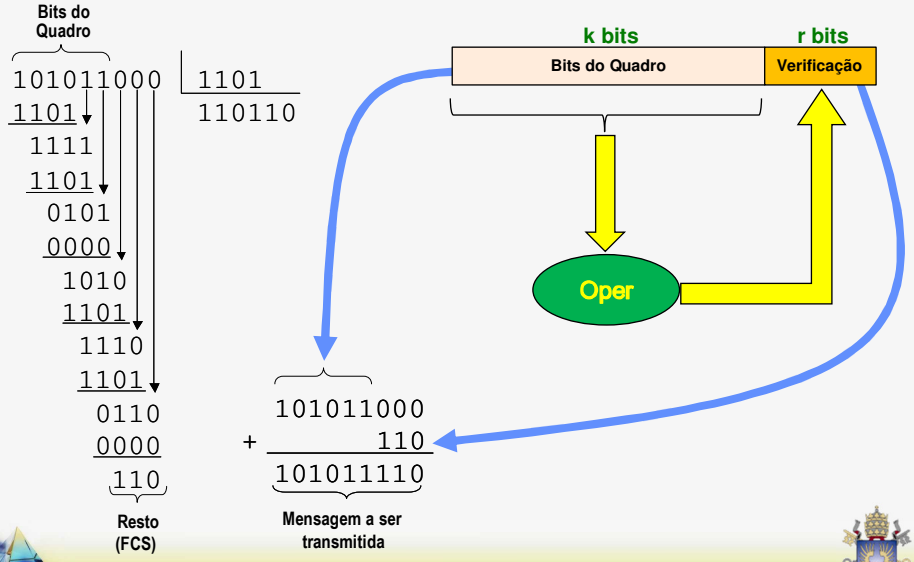
Bits do Quadro

101011



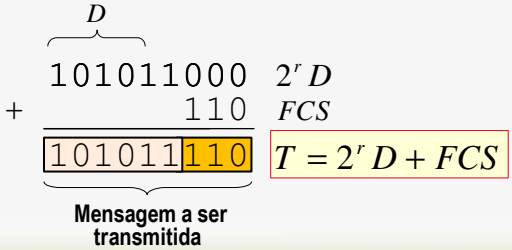
Geração do CRC

Polinômio Gerador: $x^3 + x^2 + 1$



CRC na Recepção

- Considere que
  - *T é a seqüência de bits (de tamanho n = k+r) transmitida,*
  - *D é a informação dos bits do quadro (com k bits) e*
  - *P é a seqüência de bits (de tamanho r + 1) que corresponde ao polinômio gerador.*
- *T* é a concatenação de *D* com o *FCS* à direita.



CRC

- **Sabemos que o FCS é o resto da divisão de 2^rD por P**
  - *2^rD é o dividendo*
  - *P é o divisor*
  - *Chamaremos o quociente de Q,*
  - *FCS é o resto*
- ***Então:*** $2^r D = P \cdot Q + FCS$

CRC

$$2^r D = P \cdot Q + FCS$$

CRC

$$T = 2^r D + FCS$$

$$2^r D = P \cdot Q + FCS \quad \Rightarrow \quad T = P \cdot Q + FCS + FCS$$

$$\boxed{T = P \cdot Q}$$

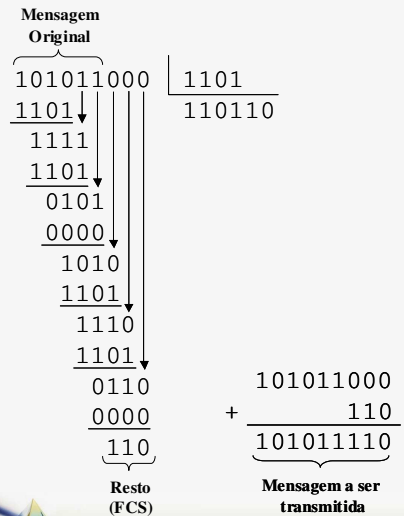
A soma é um XOR bit a bit

Isso significa que a mensagem transmitida é divisível por P.

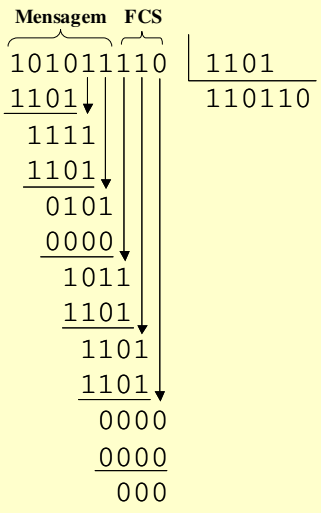
Ou seja: uma transmissão sem erros tem que ser divisível por *P.*

Logo, na recepção, basta que se efetue a divisão do quadro inteiro recebido (incluindo o FCS) pela seqüência *P.* ⇨ ***Um erro será detectado toda vez que o resto dessa divisão for diferente de zero.***

Verificação do CRC na Recepção



Na Recepção

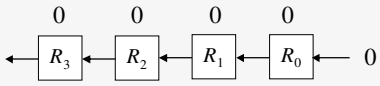


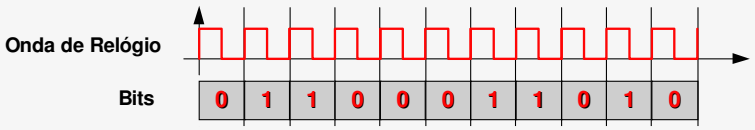
CRC

- **Códigos cíclicos apresentam algumas características interessantes.**
  - ***Uma delas é decorrência de uma propriedade que permite circuitos simples como método de obtenção do resto da divisão em aritmética módulo-2***
    - **Registros de Deslocamento com Realimentação Linear (Linear Feedback Shift Registers — LFSRs).**

Registro de Deslocamento com 4 bits

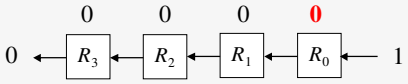
Registros de Deslocamento : a cada período de relógio completado, cada flip-flop torna disponível na sua saída o valor que estava previamente armazenado e armazena o novo valor disponível na sua entrada.

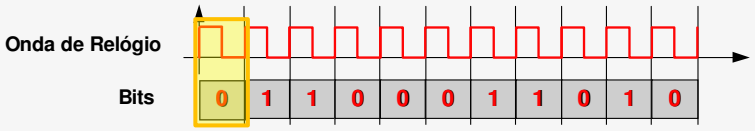




Registro de Deslocamento com 4 bits

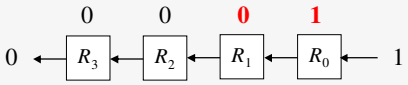
Registros de Deslocamento : a cada período de relógio completado, cada flip-flop torna disponível na sua saída o valor que estava previamente armazenado e armazena o novo valor disponível na sua entrada.

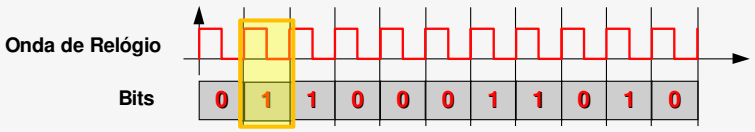




Registro de Deslocamento com 4 bits

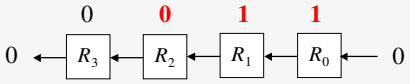
Registros de Deslocamento : a cada período de relógio completado, cada flip-flop torna disponível na sua saída o valor que estava previamente armazenado e armazena o novo valor disponível na sua entrada.

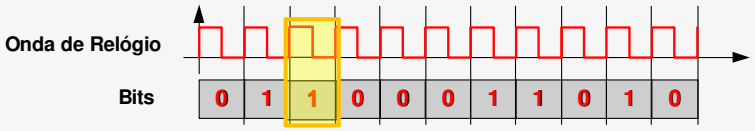




Registro de Deslocamento com 4 bits

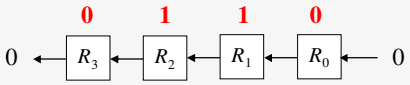
Registros de Deslocamento : a cada período de relógio completado, cada flip-flop torna disponível na sua saída o valor que estava previamente armazenado e armazena o novo valor disponível na sua entrada.

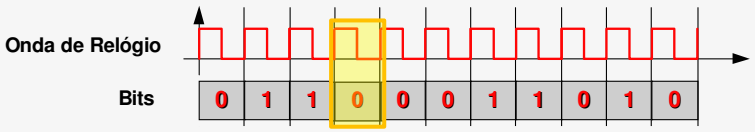




Registro de Deslocamento com 4 bits

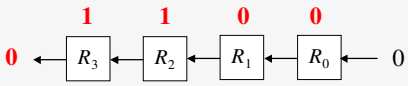
Registros de Deslocamento : a cada período de relógio completado, cada flip-flop torna disponível na sua saída o valor que estava previamente armazenado e armazena o novo valor disponível na sua entrada.

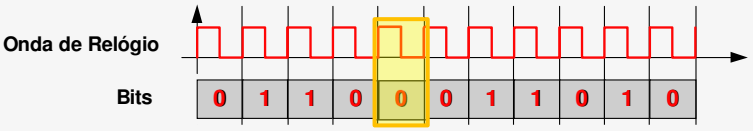




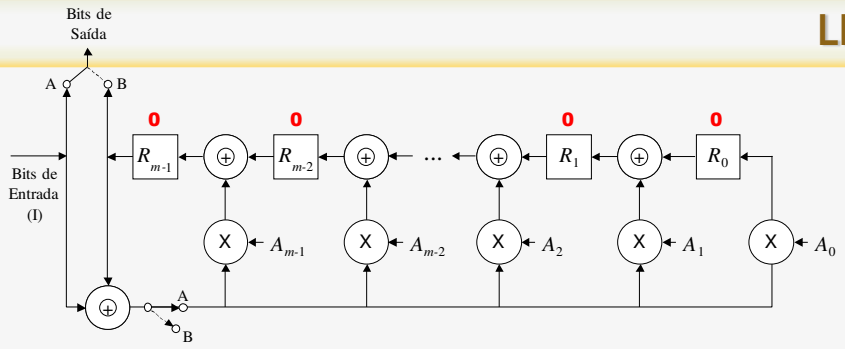
Registro de Deslocamento com 4 bits

Registros de Deslocamento : a cada período de relógio completado, cada flip-flop torna disponível na sua saída o valor que estava previamente armazenado e armazena o novo valor disponível na sua entrada.



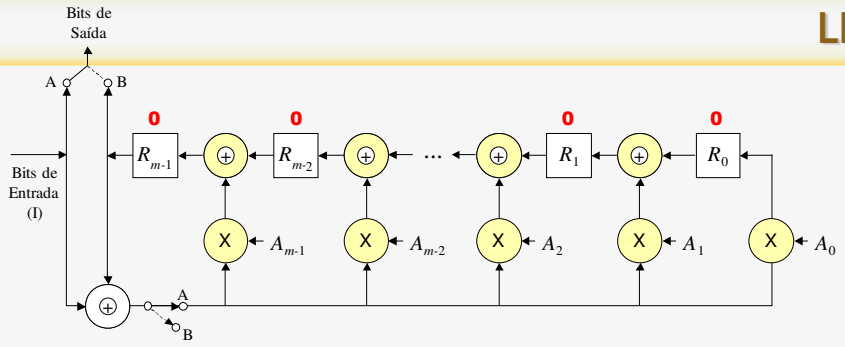


LFSR



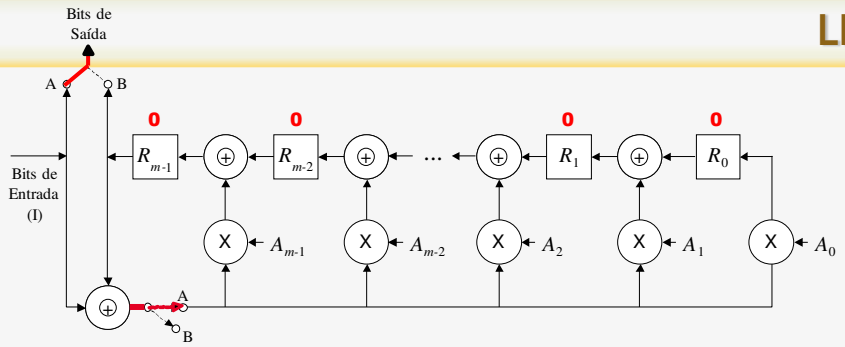
- Inicialmente, antes do cálculo, todos os flip-flops são colocados em 0.
- **Os elementos circulares da figura correspondem a operadores bit a bit para as operações de XOR e de AND conforme o símbolo indicado.**
- **A cada período de relógio, um bit da seqüência de bits de entrada (*I*) é "empurrado" para dentro do circuito.**
  - *Enquanto houver bits de I a serem processados, as duas chaves indicadas na figura permanecem na posição "A",*
    - **permitindo a entrada de novos bits e simultaneamente colocando na saída os próprios bits da mensagem original.**
- **Depois de todos os bits de *I* terem entrado no circuito, essas chaves mudam para a posição "B",**
  - *Fazendo com que não sejam mais lidos novos bits na entrada e que os bits armazenados nos registros R_{m-1} a R_0 sejam colocados, um a um, para fora do circuito.*
    - **Esses serão os bits que corresponderão ao resto da divisão dos bits de entrada pela seqüência geradora (A_m, A_{m-1}, ..., A_1, A_0) e serão colocados na saída logo após toda a seqüência de entrada.**

LFSR



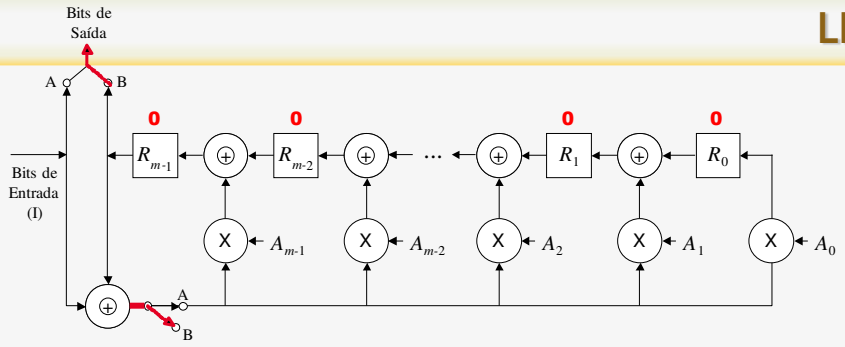
- **Inicialmente, antes do cálculo, todos os flip-flops são colocados em 0.**
- **Os elementos circulares da figura correspondem a operadores bit a bit para as operações de XOR e de AND conforme o símbolo indicado.**
- **A cada período de relógio, um bit da seqüência de bits de entrada (*I*) é "empurrado" para dentro do circuito.**
  - ***Enquanto houver bits de I a serem processados, as duas chaves indicadas na figura permanecem na posição "A",***
    - **permitindo a entrada de novos bits e simultaneamente colocando na saída os próprios bits da mensagem original.**
- **Depois de todos os bits de *I* terem entrado no circuito, essas chaves mudam para a posição "B",**
  - ***Fazendo com que não sejam mais lidos novos bits na entrada e que os bits armazenados nos registros R_{m-1} a R_0 sejam colocados, um a um, para fora do circuito.***
    - **Esses serão os bits que corresponderão ao resto da divisão dos bits de entrada pela seqüência geradora (A_m, A_{m-1}, ..., A_1, A_0) e serão colocados na saída logo após toda a seqüência de entrada.**

LFSR



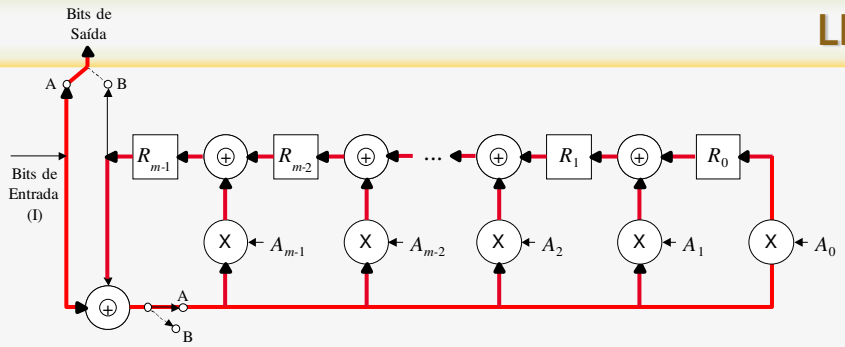
- **Inicialmente, antes do cálculo, todos os flip-flops são colocados em 0.**
- **Os elementos circulares da figura correspondem a operadores bit a bit para as operações de XOR e de AND conforme o símbolo indicado.**
- **A cada período de relógio, um bit da seqüência de bits de entrada (*I*) é "empurrado" para dentro do circuito.**
  - ***Enquanto houver bits de I a serem processados, as duas chaves indicadas na figura permanecem na posição "A",***
    - **permitindo a entrada de novos bits e simultaneamente colocando na saída os próprios bits da mensagem original.**
- **Depois de todos os bits de *I* terem entrado no circuito, essas chaves mudam para a posição "B",**
  - ***Fazendo com que não sejam mais lidos novos bits na entrada e que os bits armazenados nos registros R_{m-1} a R_0 sejam colocados, um a um, para fora do circuito.***
    - **Esses serão os bits que corresponderão ao resto da divisão dos bits de entrada pela seqüência geradora (A_m, A_{m-1}, ..., A_1, A_0) e serão colocados na saída logo após toda a seqüência de entrada.**

LFSR



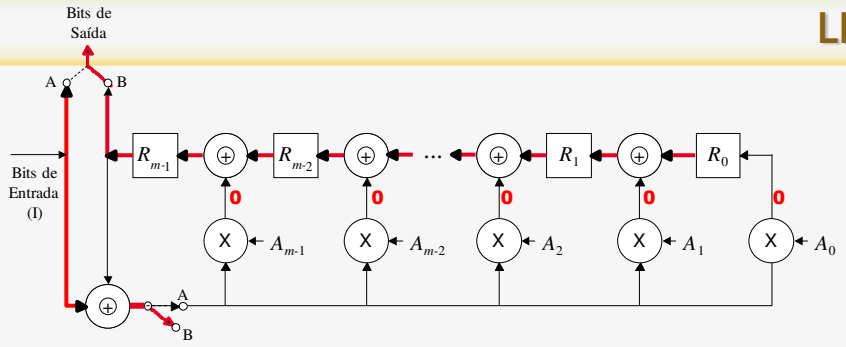
- **Inicialmente, antes do cálculo, todos os flip-flops são colocados em 0.**
- **Os elementos circulares da figura correspondem a operadores bit a bit para as operações de XOR e de AND conforme o símbolo indicado.**
- **A cada período de relógio, um bit da seqüência de bits de entrada (*I*) é "empurrado" para dentro do circuito.**
  - *Enquanto houver bits de I a serem processados, as duas chaves indicadas na figura permanecem na posição "A",*
    - permitindo a entrada de novos bits e simultaneamente colocando na saída os próprios bits da mensagem original.
- **Depois de todos os bits de *I* terem entrado no circuito, essas chaves mudam para a posição "B",**
  - *Fazendo com que não sejam mais lidos novos bits na entrada e que os bits armazenados nos registros R_{m-1} a R_0 sejam colocados, um a um, para fora do circuito.*
    - **Esses serão os bits que corresponderão ao resto da divisão dos bits de entrada pela seqüência geradora (A_m, A_{m-1}, ..., A_1, A_0) e serão colocados na saída logo após toda a seqüência de entrada.**

LFSR



- **Inicialmente, antes do cálculo, todos os flip-flops são colocados em 0.**
- **Os elementos circulares da figura correspondem a operadores bit a bit para as operações de XOR e de AND conforme o símbolo indicado.**
- **A cada período de relógio, um bit da seqüência de bits de entrada (*I*) é "empurrado" para dentro do circuito.**
  - *Enquanto houver bits de I a serem processados, as duas chaves indicadas na figura permanecem na posição "A",*
    - permitindo a entrada de novos bits e simultaneamente colocando na saída os próprios bits da mensagem original.
- **Depois de todos os bits de *I* terem entrado no circuito, essas chaves mudam para a posição "B",**
  - *Fazendo com que não sejam mais lidos novos bits na entrada e que os bits armazenados nos registros R_{m-1} a R_0 sejam colocados, um a um, para fora do circuito.*
    - **Esses serão os bits que corresponderão ao resto da divisão dos bits de entrada pela seqüência geradora (A_m, A_{m-1}, ..., A_1, A_0) e serão colocados na saída logo após toda a seqüência de entrada.**

LFSR



- Inicialmente, antes do cálculo, todos os flip-flops são colocados em 0.
- Os elementos circulares da figura correspondem a operadores bit a bit para as operações de XOR e de AND conforme o símbolo indicado.
- A cada período de relógio, um bit da seqüência de bits de entrada (*I*) é "empurrado" para dentro do circuito.
  - *Enquanto houver bits de I a serem processados, as duas chaves indicadas na figura permanecem na posição "A",*
    - permitindo a entrada de novos bits e simultaneamente colocando na saída os próprios bits da mensagem original.

- Depois de todos os bits de *I* terem entrado no circuito, essas chaves mudam para a posição "B",
  - *Fazendo com que não sejam mais lidos novos bits na entrada e que os bits armazenados nos registros R_{m-1} a R_0 sejam colocados, um a um, para fora do circuito.*
    - Esses serão os bits que corresponderão ao resto da divisão dos bits de entrada pela seqüência geradora ($A_m, A_{m-1}, ..., A_1, A_0$) e serão colocados na saída logo após toda a seqüência de entrada.

Exemplo

- **Transmissão do quadro 101011 utilizando um polinômio gerador igual a X^3+X^2+1.**

Mensagem Original

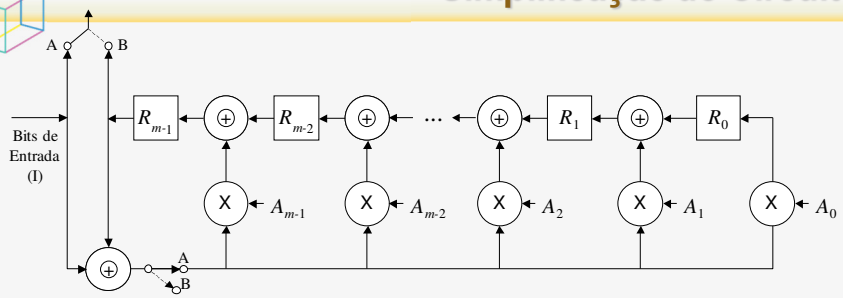
```
1010110000 | 1101
1101       | 110110
 1111
 1101
  0101
  0000
   1010
   1101
    1110
    1101
     0110
     0000
      110
```

Resto (FCS)

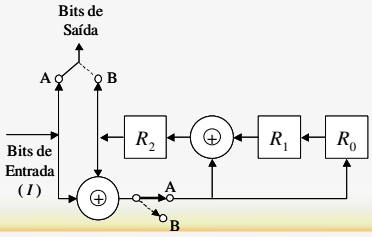
```
  101011000
+       110
-----------
  101011110
```

Mensagem a ser transmitida

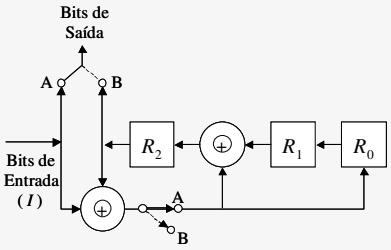
Simplificação do Circuito



- Uma simplificação pode ser feita considerando que A_1=0 (fazendo com que o XOR posicionado entre R_0 e R_1 seja desnecessário) e A_0=A_2=1 (fazendo com que os multiplicadores por A_0 e A_2 sejam desnecessários).



Funcionamento

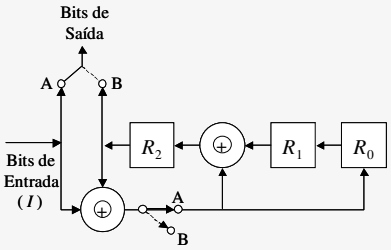


- $R_0(t+1) = R_2(t) \oplus I(t)$
- $R_1(t+1) = R_0(t)$
- $R_2(t+1) = R_2(t) \oplus R_1(t) \oplus I(t)$

t	R_2	R_1	R_0	$R_2 \oplus R_1 \oplus I$	$R_2 \oplus I$	I

Resto

Funcionamento

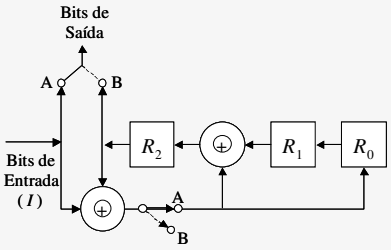


- $R_0(t+1) = R_2(t) \oplus I(t)$
- $R_1(t+1) = R_0(t)$
- $R_2(t+1) = R_2(t) \oplus R_1(t) \oplus I(t)$

t	R_2	R_1	R_0	$R_2 \oplus R_1 \oplus I$	$R_2 \oplus I$	I
0	0	0	0			1
1						0
2						1
3						0
4						1
5						1
6						

Resto

Funcionamento

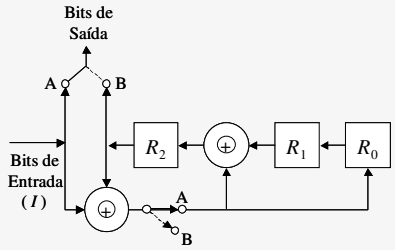


- $R_0(t+1) = R_2(t) \oplus I(t)$
- $R_1(t+1) = R_0(t)$
- $R_2(t+1) = R_2(t) \oplus R_1(t) \oplus I(t)$

t	R_2	R_1	R_0	$R_2 \oplus R_1 \oplus I$	$R_2 \oplus I$	I
0	0	0	0	1	1	1
1						0
2						1
3						0
4						1
5						1
6						

Resto

Funcionamento

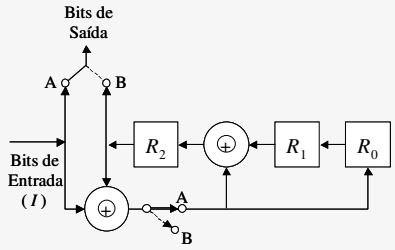


t	R_2	R_1	R_0	$R_2 \oplus R_1 \oplus I$	$R_2 \oplus I$	I
0	0	0	0	1	1	1
1	1	0	1			0
2						1
3						0
4						1
5						1
6						

Resto

- $R_0(t+1) = R_2(t) \oplus I(t)$
- $R_1(t+1) = R_0(t)$
- $R_2(t+1) = R_2(t) \oplus R_1(t) \oplus I(t)$

Funcionamento

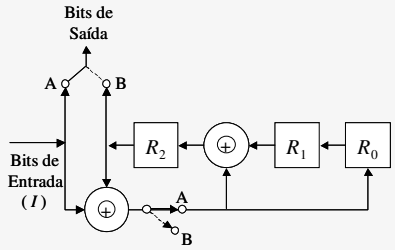


t	R_2	R_1	R_0	$R_2 \oplus R_1 \oplus I$	$R_2 \oplus I$	I
0	0	0	0	1	1	1
1	1	0	1	1	1	0
2						1
3						0
4						1
5						1
6						

Resto

- $R_0(t+1) = R_2(t) \oplus I(t)$
- $R_1(t+1) = R_0(t)$
- $R_2(t+1) = R_2(t) \oplus R_1(t) \oplus I(t)$

Funcionamento

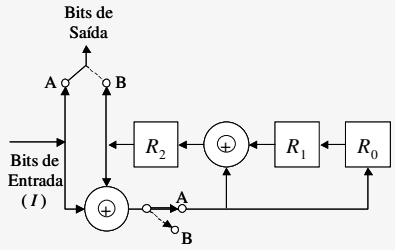


- $R_0(t+1) = R_2(t) \oplus I(t)$
- $R_1(t+1) = R_0(t)$
- $R_2(t+1) = R_2(t) \oplus R_1(t) \oplus I(t)$

t	R_2	R_1	R_0	$R_2 \oplus R_1 \oplus I$	$R_2 \oplus I$	I
0	0	0	0	1	1	1
1	1	0	1	1	1	0
2	1	1	1	1	0	1
3						0
4						1
5						1
6						

Resto

Funcionamento

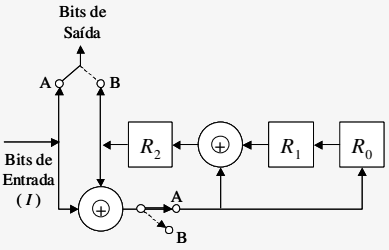


- $R_0(t+1) = R_2(t) \oplus I(t)$
- $R_1(t+1) = R_0(t)$
- $R_2(t+1) = R_2(t) \oplus R_1(t) \oplus I(t)$

t	R_2	R_1	R_0	$R_2 \oplus R_1 \oplus I$	$R_2 \oplus I$	I
0	0	0	0	1	1	1
1	1	0	1	1	1	0
2	1	1	1	1	0	1
3	1	1	0	0	1	0
4	0	0	1	1	1	1
5	1	1	1	1	0	1
6	1	1	0			

Resto

Funcionamento



R_2	R_1	R_0	$R_2 \oplus R_1 \oplus I$	$R_2 \oplus I$	I
0	0	0	1	1	1
1	0	1	1	1	0
1	1	1	1	0	1
1	1	0	0	1	0
0	0	1	1	1	1
1	1	1	1	0	1
1	1	0			

Resto

Mensagem Original

```
1010111000 | 1101
1101       | 110110
 1111
 1101
  0101
  0000
   1010
   1101
    1110
    1101
     0110
     0000
      110
```

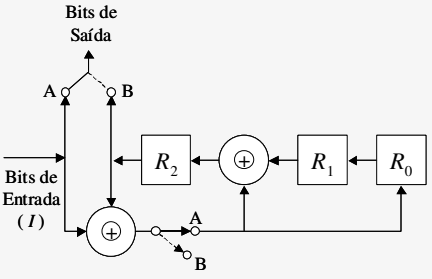
Resto (FCS)

Verificação do CRC na Recepção

Na Recepção

Mensagem FCS

```
1010111110 | 1101
1101       | 110110
 1111
 1101
  0101
  0000
   1011
   1101
    1101
    1101
     0000
     0000
      000
```



t	R_2	R_1	R_0	$R_2 \oplus R_1 \oplus I$	$R_2 \oplus I$	I
0						1
1						0
2						1
3						0
4						1
5						1
6						1
7						1
8						0
9	0	0	0			

Resto


DEPARTAMENTO
DE INFORMÁTICA
PUC-RIO
PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Obrigado


TeleMídia