# Assignment 1 – Cyclic Redundancy Check

Prática P3, Grupo 6:

António Silva (76678)
Rafael Oliveira (76525)

## Arquitectura de Computadores Avançada

# 1. Encoder (solução assíncrona)

**a** 1100100 | 101 **b**
**r4(x)** 0110 11110 **q**
**r3(x)** 0111
**r2(x)** 0100
**r1(x)** 0010
**r0(x)** 010

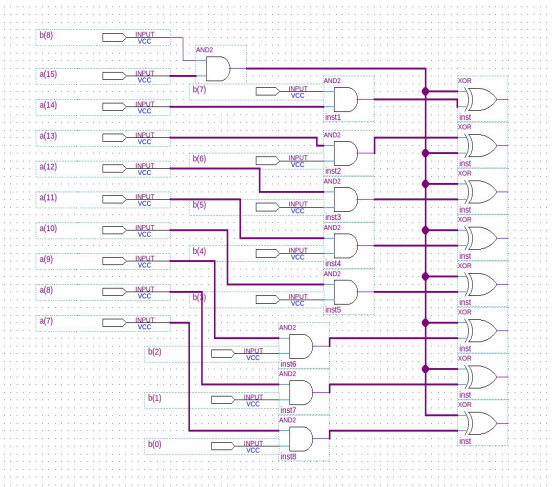
q4 =a4 x b2
r4(0) =q4 x b0 xor a2
r4(1) =q4 x b1 xor a3

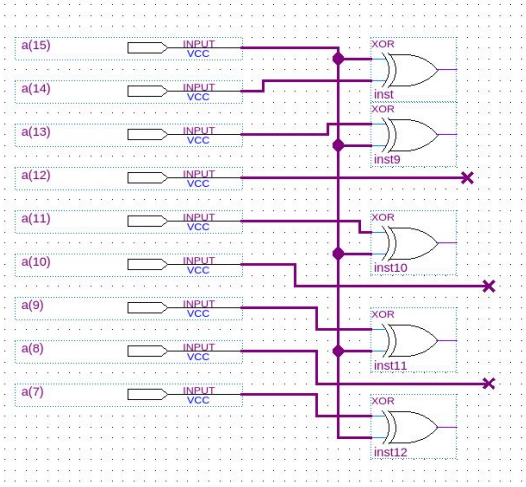
q4 =a4 x b2 ∩ b2 =1 > a4
r4(0) =q4 x b0 xor a2 ∩ b0=1 ∩ q4=a4 > a4 xor a2
r4(1) =q4 x b1 xor a3 ∩ b1=0> a3

# 1. Encoder (solução assíncrona)

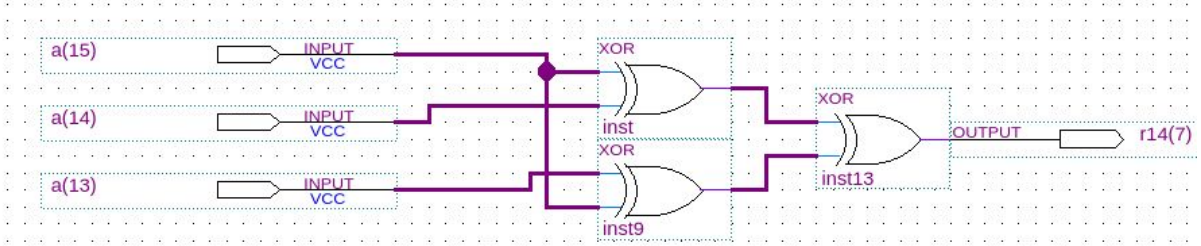
- A obtenção do valor de q(x) será desnecessária, visto que, resultará sempre do valor mais significativo do último r0-4(x) obtido.

- No cálculo de ry(x), a multiplicação com os valores de b(x) pode ser removida.
  - ry(x) =r[y-1](x-1)  nos casos que  b(z) == 0
  - ry(x) =r[y-1](x-1) xor r[y-1](1)  nos casos que  b(z) == 1



# 1. Encoder (solução assíncrona)




r14(7) = a(15) xor a(14) xor a(13) xor a(15)

r14(7) = a(14) xor a(13)

# 1. Encoder (solução assíncrona)

Aplicando a cada rx(7) temos:

```
r0(7) = a(0) xor a(2) xor a(5) xor a(6) xor a(7) xor a(12) xor a(13) xor a(14) xor a(15)
r1(7) = a(0) xor a(1) xor a(3) xor a(6) xor a(7) xor a(8) xor a(13) xor a(14) xor a(15)
r2(7) = a(1) xor a(2) xor a(4) xor a(7) xor a(8) xor a(9) xor a(14) xor a(15)
r3(7) = a(2) xor a(3) xor a(5) xor a(8) xor a(9) xor a(10) xor a(15)
                (...)
r10(7) = a(9) xor a(10) xor a(12) xor a(15)
r11(7) = a(10) xor a(11) xor a(13)
r12(7) = a(11) xor a(12) xor a(14)
r13(7) = a(12) xor a(13) xor a(15)
r14(7) = a(13) xor a(14)
r15(7) = a(14) xor a(15)
r16(7) = a(15)
```

Analisando as portas de saída podemos encontrar as dependências com rx(7):

```
r0(7) =a(0) xor a(2) xor a(5) xor a(6) xor a(7) xor a(12) xor a(13) xor a(14) xor a(15)
r0(6) =r1(7) xor r3(7) xor r5(7) xor r7(7)
r0(5) =r2(7) xor r4(7) xor r6(7)
r0(4) =r1(7) xor r3(7) xor r5(7)
r0(3) =r2(7) xor r4(7)
r0(2) =r1(7) xor r3(7)
r0(1) =r2(7)
r0(0) =r1(7)
```

# 1. Encoder (solução assíncrona)

```
r0(7) =a(0) xor a(2) xor a(5) xor a(6) xor a(7) xor a(12) xor a(13) xor a(14) xor a(15)
r0(6) = a(0) xor a(1) xor a(2) xor a(4) xor a(7) xor a(11)
r0(5) = a(1) xor a(2) xor a(3) xor a(5) xor a(7) xor a(10) xor a(12) xor a(13) xor a(14) xor a(15)
r0(4) = a(0) xor a(1) xor a(2) xor a(4) xor a(6) xor a(9) xor a(11) xor a(12) xor a(13) xor a(14)
r0(3) = a(1) xor a(2) xor a(3) xor a(7) xor a(8) xor a(10) xor a(11) xor a(14) xor a(15)
r0(2) = a(0) xor a(1) xor a(2) xor a(5) xor a(6) xor a(7) xor a(9) xor a(10) xor a(13) xor a(14)
r0(1) = a(1) xor a(2) xor a(4) xor a(7) xor a(8) xor a(9) xor a(14) xor a(15)
r0(0) = a(0) xor a(1) xor a(3) xor a(6) xor a(7) xor a(8) xor a(13) xor a(14) xor a(15)
```

**Imensas repetições de XOR`s!**

# 1. Encoder (solução assíncrona)

| | a(0) | a(1) | a(2) | a(3) | a(4) | a(5) | a(6) | a(7) | a(8) | a(9) | a(10) | a(11) | a(12) | a(13) | a(14) | a(15) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a(0) | | 4 | 4 | 1 | 2 | 2 | 4 | 4 | 1 | 2 | 1 | 2 | 2 | 4 | 4 | 2 |
| a(1) | 4 | | 6 | 3 | 3 | 2 | 4 | 6 | 3 | 3 | 3 | 3 | 2 | 4 | 6 | 4 |
| a(2) | 4 | 6 | | 2 | 3 | 3 | 4 | 6 | 3 | 2 | 3 | 3 | 3 | 4 | 6 | 4 |
| a(3) | 1 | 3 | 2 | | | 1 | 2 | 3 | 2 | | 2 | 1 | 1 | 2 | 3 | 3 |
| a(4) | 2 | 3 | 3 | | | | 1 | 2 | 1 | 2 | | 2 | 1 | 1 | 2 | 1 |
| a(5) | 2 | 2 | 3 | 1 | | | 2 | 3 | 1 | 1 | 2 | | 2 | 3 | 3 | 2 |
| a(6) | 4 | 4 | 4 | 2 | 1 | 2 | | 4 | 2 | 2 | 2 | 2 | 2 | 4 | 5 | 3 |
| a(7) | 4 | 6 | 6 | 3 | 2 | 3 | 4 | | 3 | 2 | 2 | 2 | 2 | 4 | 6 | 4 |
| a(8) | 1 | 3 | 3 | 2 | 1 | 1 | 2 | 3 | | 1 | 1 | 1 | | 1 | 3 | 3 |
| a(9) | 2 | 3 | 2 | | 2 | 1 | 2 | 2 | 1 | | 1 | 1 | 1 | 2 | 2 | 1 |
| a(10) | 1 | 3 | 3 | 2 | | 2 | 2 | 2 | 1 | 1 | | 1 | 1 | 2 | 3 | 2 |
| a(11) | 2 | 3 | 3 | 1 | 2 | | 2 | 2 | 1 | 1 | 1 | | 1 | 1 | 2 | 1 |
| a(12) | 2 | 2 | 3 | 1 | 1 | 2 | 2 | 2 | | 1 | 1 | 1 | | 3 | 3 | 2 |
| a(13) | 4 | 4 | 4 | 2 | 1 | 3 | 4 | 4 | 1 | 2 | 2 | 1 | 3 | | 5 | 3 |
| a(14) | 4 | 6 | 6 | 3 | 2 | 3 | 5 | 6 | 3 | 2 | 3 | 2 | 3 | 5 | | 5 |
| a(15) | 2 | 4 | 4 | 3 | 1 | 2 | 3 | 4 | 3 | 1 | 2 | 1 | 2 | 3 | 5 | |

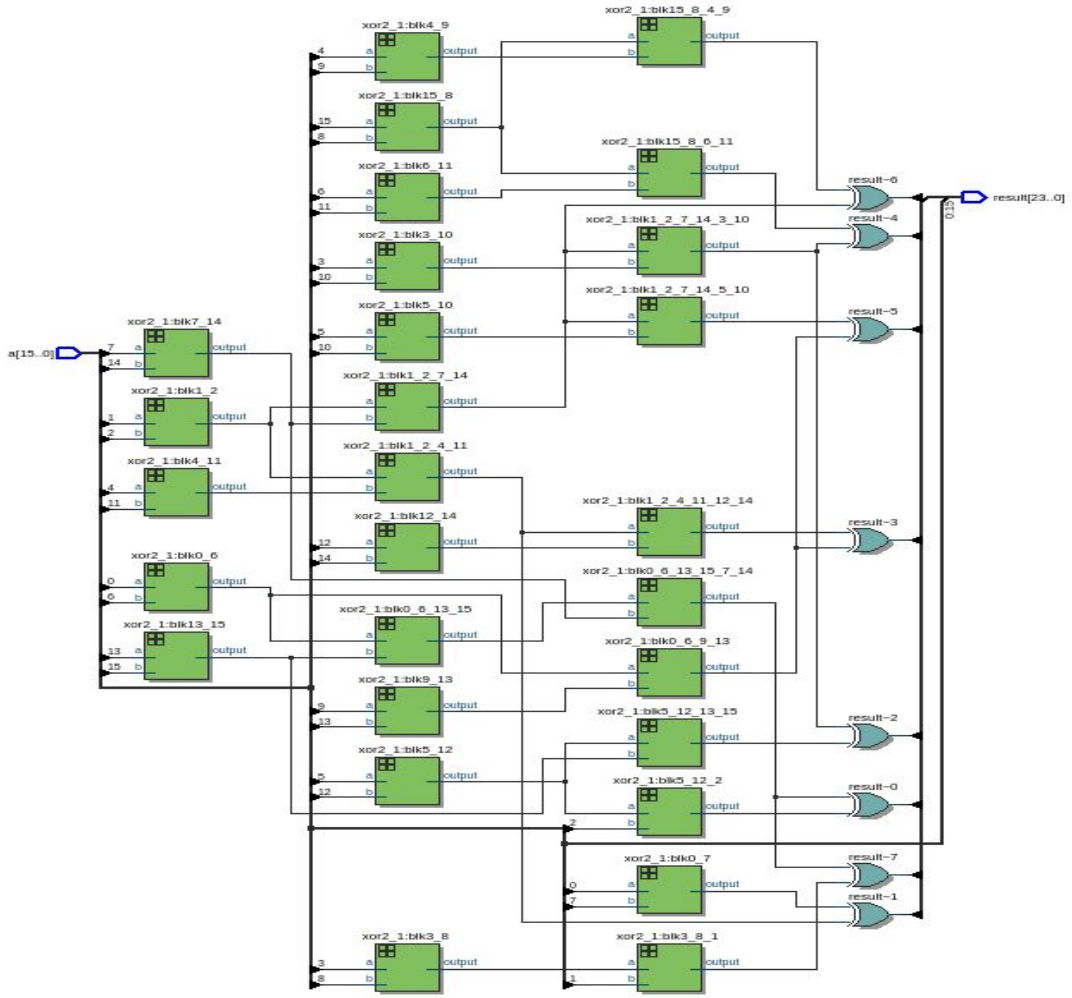
```
-- 1st level XOR SIGNALS
SIGNAL xor1w2: STD_LOGIC;
SIGNAL xor7w14: STD_LOGIC;
SIGNAL xor0w6: STD_LOGIC;
SIGNAL xor13w15: STD_LOGIC;
SIGNAL xor9w13: STD_LOGIC;
SIGNAL xor5w12: STD_LOGIC;
SIGNAL xor4w11: STD_LOGIC;
SIGNAL xor15w8: STD_LOGIC;
SIGNAL xor3w10: STD_LOGIC;
SIGNAL xor0w7: STD_LOGIC;
SIGNAL xor12w14: STD_LOGIC;
SIGNAL xor6w11: STD_LOGIC;
SIGNAL xor5w10: STD_LOGIC;
SIGNAL xor4w9: STD_LOGIC;
SIGNAL xor3w8: STD_LOGIC;
```

```
-- 2nd level XOR SIGNALS
SIGNAL xor3w8w1: STD_LOGIC;
SIGNAL xor5w12w2: STD_LOGIC;
SIGNAL xor1w2w7w14: STD_LOGIC;
SIGNAL xor0w6w13w15: STD_LOGIC;
SIGNAL xor1w2w4w11: STD_LOGIC;
SIGNAL xor0w6w9w13: STD_LOGIC;
SIGNAL xor5w12w13w15: STD_LOGIC;
SIGNAL xor15w8w6w11: STD_LOGIC;
SIGNAL xor15w8w4w9: STD_LOGIC;
-- 3rd level XOR SIGNALS
SIGNAL xor0w6w13w15w7w14: STD_LOGIC;
SIGNAL xor1w2w7w14w3w10: STD_LOGIC;
SIGNAL xor1w2w4w11w12w14: STD_LOGIC;
SIGNAL xor1w2w7w14w5w10: STD_LOGIC;
```

# 1. Encoder (solução assíncrona)

```
result(7) <= xor0w6w13w15w7w14 xor xor5w12w2;
result(6) <= xor1w2w4w11       xor xor0w7;
result(5) <= xor1w2w7w14w3w10  xor xor5w12w13w15;
result(4) <= xor0w6w9w13       xor xor1w2w4w11w12w14;
result(3) <= xor1w2w7w14w3w10  xor xor15w8w6w11;
result(2) <= xor0w6w9w13       xor xor1w2w7w14w5w10;
result(1) <= xor1w2w7w14       xor xor15w8w4w9;
result(0) <= xor0w6w13w15w7w14 xor xor3w8w1;
```

Com esta solução temos 36 XORs e um total de 4 níveis. O que defendemos que será a solução mais eficiente e com baixa complexidade.


a[15..0]
xor2_1:blk7_14
xor2_1:blk1_2
xor2_1:blk4_11
xor2_1:blk0_6
xor2_1:blk13_15
xor2_1:blk4_9
xor2_1:blk15_8
xor2_1:blk6_11
xor2_1:blk3_10
xor2_1:blk5_10
xor2_1:blk1_2_7_14
xor2_1:blk1_2_4_11
xor2_1:blk12_14
xor2_1:blk0_6_13_15
xor2_1:blk9_13
xor2_1:blk5_12
xor2_1:blk3_8
xor2_1:blk15_8_4_9
xor2_1:blk15_8_6_11
xor2_1:blk1_2_7_14_3_10
xor2_1:blk1_2_7_14_5_10
xor2_1:blk1_2_4_11_12_14
xor2_1:blk0_6_13_15_7_14
xor2_1:blk0_6_9_13
xor2_1:blk5_12_13_15
xor2_1:blk5_12_2
xor2_1:blk0_7
xor2_1:blk3_8_1
result~6
result~4
result~5
result~3
result~2
result~0
result~7
result~1
result[23..0]

# 2. Checker (solução síncrona)

- **Input:** Mensagem(16 bits) + Resultado encoder(8 bits).
- **Resultado:** Se obtermos “00000000”, garantimos que a mensagem não foi modificada.
- 8 *Flip-Flops* e 5 *XOR*’s
- XOR’s apenas à entrada dos bits a 1 do polinômio.
- Por ciclo de relógio é dado um bit da palavra de 24 bits.
- Verificação do resultado.
- Se “00000000”, *ready* = ‘1’, ou seja, a mensagem não foi modificada nem apresenta erros.