Hamming Codes[15, 11]

Digital Circuit Implementation

Pedro Abreu, 93240
João Gameiro, 93097
Grupo 2

universidade
de aveiro

Assignment 1 - Arquiteturas de Alto Desempenho
prof. António Rui Borges
Dez. 2021

Decoder - Parallel Implementation

The start point were the expressions given by the professor. We tried to find common operations that could be simplified in order to reduce the number of components. After the simplifications, an analysis was made in order to figure out the maximum number of operations that could be executed simultaneously so that the propagation delay could be minimized.

Stage-0

$$
\begin{aligned}
p_1 &= y_1 \oplus y_2 \oplus y_3 \oplus y_7 \oplus y_8 \oplus y_9 \oplus y_{11} \oplus y_{12} \\
p_2 &= y_1 \oplus y_4 \oplus y_5 \oplus y_7 \oplus y_8 \oplus y_{10} \oplus y_{11} \oplus y_{13} \\
p_3 &= y_2 \oplus y_4 \oplus y_6 \oplus y_7 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{14} \\
p_4 &= y_3 \oplus y_5 \oplus y_6 \oplus y_8 \oplus y_9 \oplus y_{10} \oplus y_{11} \oplus y_{15}
\end{aligned}
$$

$$
\begin{aligned}
A &= y_1 \oplus y_{11} \\
B &= y_6 \oplus y_{11} \\
C &= y_9 \oplus y_{10} \\
D &= y_7 \oplus y_8
\end{aligned}
$$

Stage-1

$$
\begin{aligned}
p_1 &= y_2 \oplus y_3 \oplus y_9 \oplus y_{12} \oplus A \oplus D \\
p_2 &= y_4 \oplus y_5 \oplus y_{10} \oplus y_{13} \oplus A \oplus D \\
p_3 &= y_2 \oplus y_4 \oplus y_7 \oplus y_{14} \oplus B \oplus C \\
p_4 &= y_3 \oplus y_5 \oplus y_8 \oplus y_{15} \oplus B \oplus C
\end{aligned}
$$

$$
\begin{aligned}
E &= A \oplus D \\
F &= B \oplus C
\end{aligned}
$$

Stage-2

$$
\begin{aligned}
p_1 &= y_2 \oplus y_3 \oplus y_9 \oplus y_{12} \oplus E \\
p_2 &= y_4 \oplus y_5 \oplus y_{10} \oplus y_{13} \oplus E \\
p_3 &= y_2 \oplus y_4 \oplus y_7 \oplus y_{14} \oplus F \\
p_4 &= y_3 \oplus y_5 \oplus y_8 \oplus y_{15} \oplus F
\end{aligned}
$$

Decoder - Parallel Implementation

Divided in three parts:

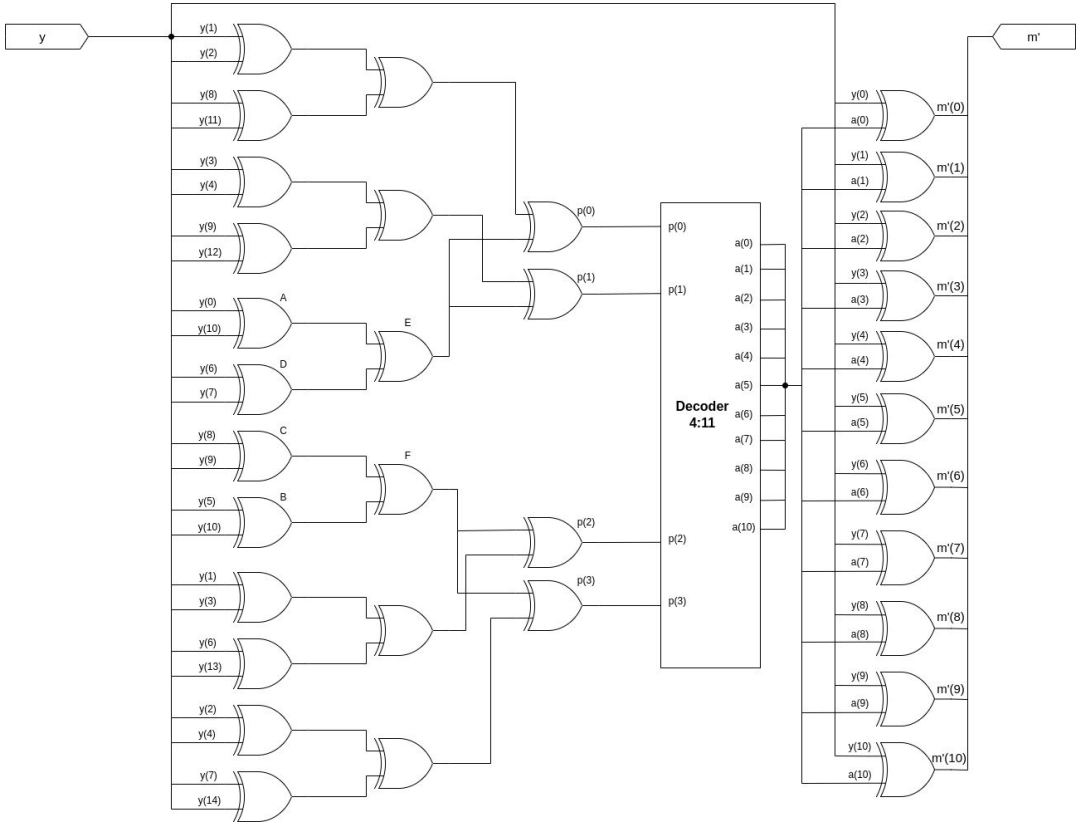
- **Parity check block**
  - Composed by 22 xors to generate the bits that check if there was an error or not.
- **Decoder 4:11**
  - To find the bit that needs to be corrected.
- **Bit error correction**
  - Composed by 11 xors to flip the erroneous bit.

Implementation Cost

- 33 XOR gates
- 19 AND gates
- 4 NOT gates

Propagation Delay

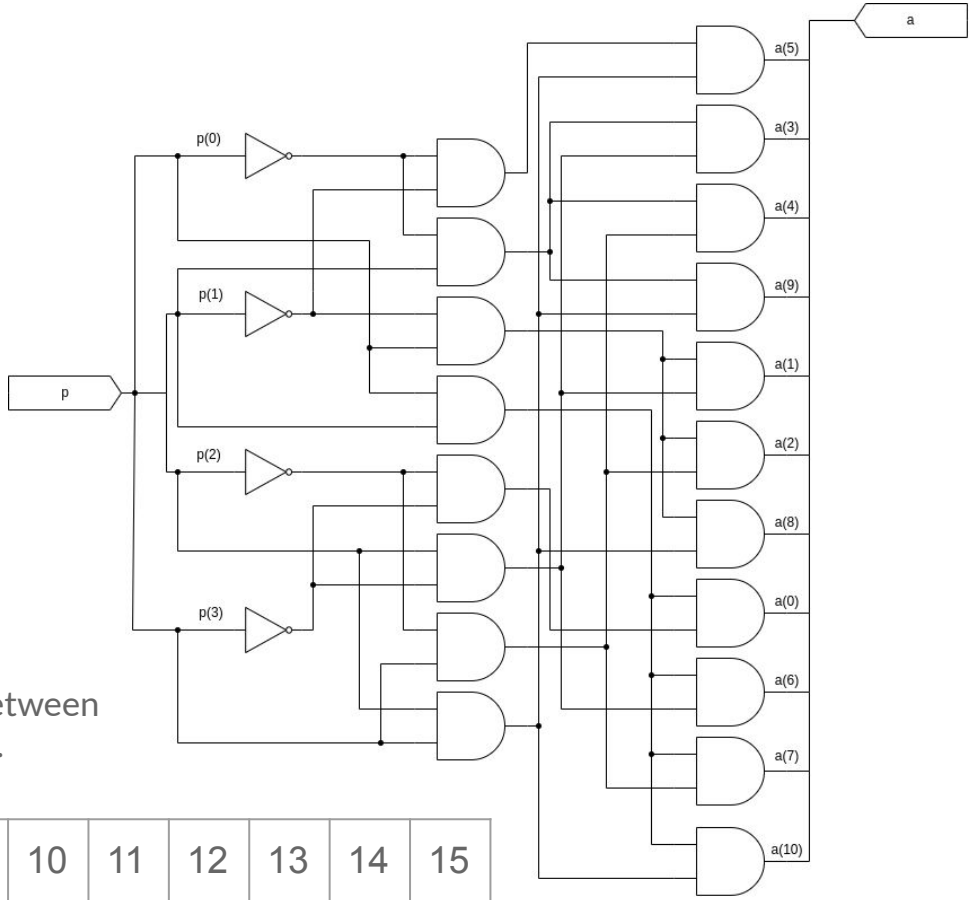
- 4 XOR gates + 2 AND gates + 1 NOT gate



Decoder - Decoder 4:11

A truth table was built and with the help of Boolean algebra the expressions that describe the relation between the input and output of the circuit were calculated.

Later on we tried to find common operations to reduce the number of components. Once again we made an analysis on the maximum number of operations that be executed simultaneously to reduce delays.



In the following table is presented the correspondence between the values of p and the bit of the message to be corrected.

p	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
bit	/	/	/	0	/	1	3	6	/	2	4	7	5	8	9	10

Encoder - Serial Implementation

State Machine

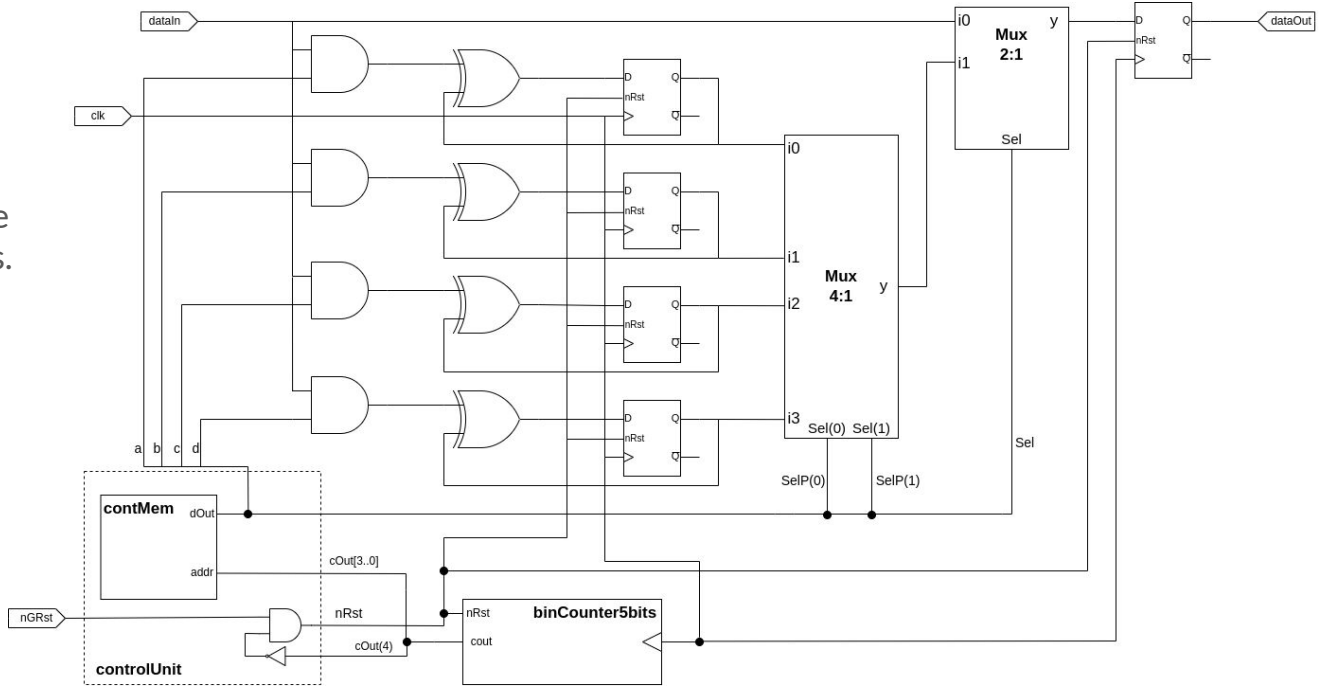
- 16 states
  - 1 setup/read first bit
  - 10 states to read bits from the message and calculate parity bits
  - 4 states to present the present the parity bits.

Reset

- General reset
- Counter reset

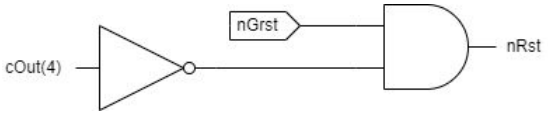
Implementation Cost

- 12 XOR gates
- 16 AND gates
- 5 NOT gates
- 10 D-Type Flip Flops



Encoder - Serial Implementation

During the implementation it was necessary to reset the system asynchronously right after encoding the 15 bit sequence.
With a 15 state machine, it was only necessary for a binary counter of 4 bits. However for the reset of the system we used a 5 bit binary counter so that the most significant bit resets the system when it reaches to '1'.



In the states we have each one of the following values

- sel -> used to select if the output will be a bit from the original message or one of the parity bits calculated from the message.
- selP -> used to select which one of the parity bits will be presented.
- a,b,c,d -> coefficients used to calculate the parity bits.

```
for(int i = 0; i < 12; i++){
    x12 = x12 xor (mi.ai);
    x13 = x13 xor (mi.bi);
    x14 = x14 xor (mi.ci);
    x15 = x15 xor (mi.di);
}
```

```
--          a b c d sel selp
--setup  -> 1 1 0 0 0   00
--E1     -> 1 0 1 0 0   00
--E2     -> 1 0 0 1 0   00
--E3     -> 0 1 1 0 0   00
--E4     -> 0 1 0 1 0   00
--E5     -> 0 0 1 1 0   00
--E6     -> 1 1 1 0 0   00
--E7     -> 1 1 0 1 0   00
--E8     -> 1 0 1 1 0   00
--E9     -> 0 1 1 1 0   00
--E10    -> 1 1 1 1 0   00
--E11    -> 0 0 0 0 1   00
--E12    -> 0 0 0 0 1   01
--E13    -> 0 0 0 0 1   10
--E14    -> 0 0 0 0 1   11
```