Report for The Lab

Group: **Sixth**

Student: **Ashraf AlAssi**

Date: **Wednesday 29-12-2021**

Project Name: Error-Correction Memory Path

- Introduction:

- Usually, the data transferred from one place to another may get distorted, so designing an algorithm which detects and correct that distortion (error) was high priority. Here where Hamming Code comes in play.

- Hamming Code: How does it work?:

1. Encoding:

- Hamming Code depends mainly on the idea of the Parity-Bit (i.e., XORing some bit in the bit-stream)

- Assume that we have n-bit to be sent (a sequence of 0s and 1s), which we denote as d_i. Hamming Code states that, we should add a k-bit to the sequence, which we denote as p_i. The sequence now has $n + k$ bits. The p_i-bits are added in the **position** corresponding to the power of 2:

Position	$1(1)$	$2(10)$	$3(11)$	...	$n + k$
Seq.	p_1	p_2	d_1	...	d_n

Those bits are calculated as follow:

The first bit is XORing all the other bits which their position, in binary representation, has a 1 in LSB.

$$p_1 = XOR(3, 5, 7, \dots)$$

The second bit is XORing all the other bits which comes after it, which their position, in binary representation, has a 1 in LSB:

$$p_2 = XOR(3, 6, 7, \dots)$$

All the other k parity bits are calculated in the same manner.

- The relationship between n and k is:

$$2^k - 1 \geq n + k$$

$\boldsymbol{k}$	$\boldsymbol{n}$
3	$[2, 4]$
4	$[5, 11]$
5	$[12, 26]$

- A bit in the zeroth position can be added and calculated by XORing the **whole** sequence.

2. Decoding:

- When receiving a binary sequence, Check-Bits are calculated as follow:

$$C_1 = XOR(1, 3, 5, 7, \dots)$$

$$C_2 = XOR(2, 3, 6, 7, \dots)$$

Those bits are calculated in the same manner as the parity bits.

- Check-Bits are now tested:

$$if\ C_i = 0 \Rightarrow no\ error$$

$$if\ C_i \neq 0 \Rightarrow error\ at\ C_i$$

Where C_i is the **position** which the data got distorted. We fix this error bit by complementing it.

- Note that Hamming Code detects and corrects only **one error**. The algorithm doesn't guarantee to fix more. The ratio between the parity bits and the data bits is as follow:

$$k = 4, n = 11 \Rightarrow x = 36\%$$

$$k = 5, n = 22 \Rightarrow x = 22\%$$

$$k = 21, n \approx 1000000 \Rightarrow x = 0.0021\%$$

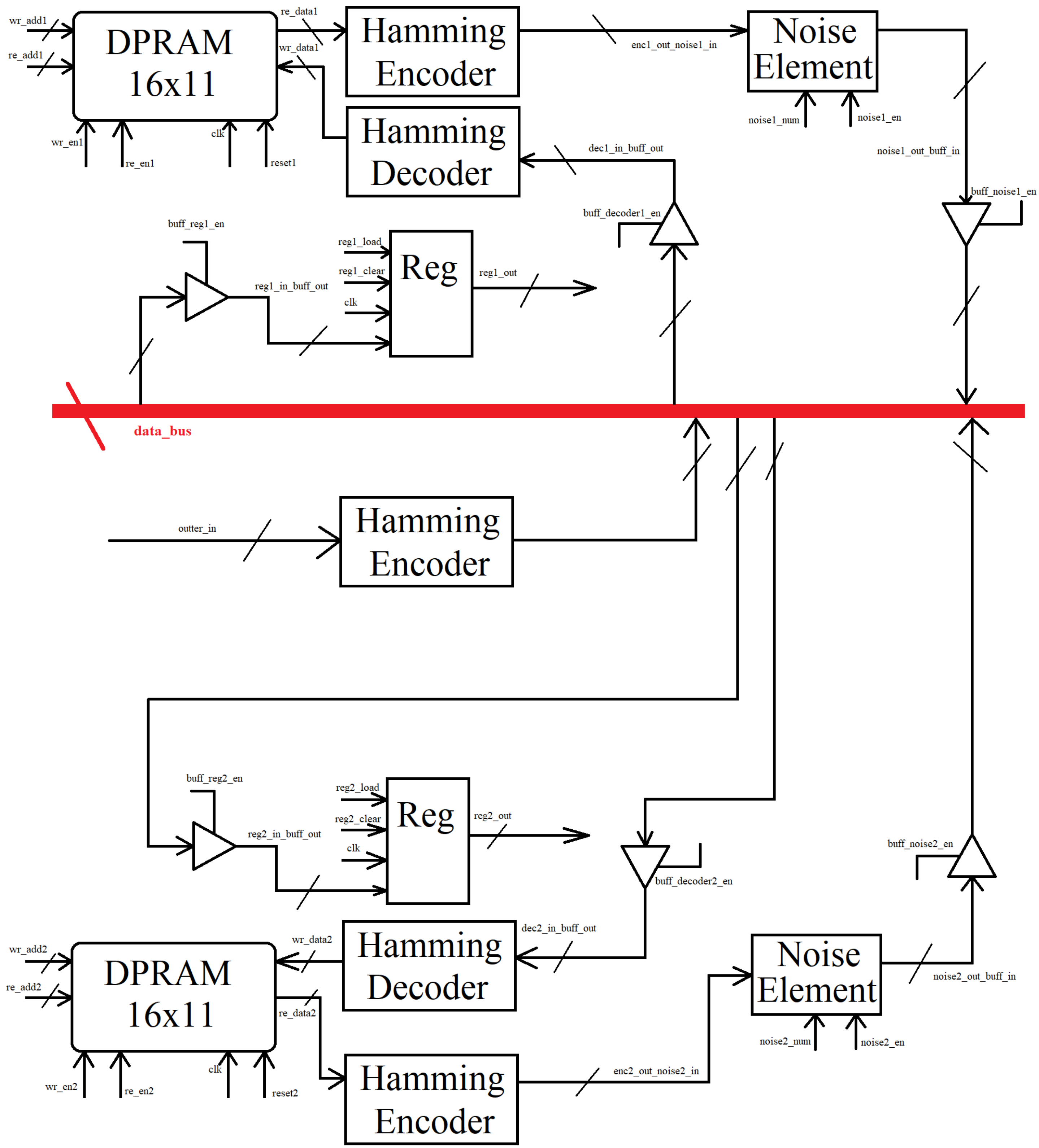
- More algorithms were devolved after the Hamming Code, one of which is Cross-Interleaved Reed-Solomon Codem which use only **25%** of the block for parity check. It's used in CDs to detects and corrects errors

- Project Diagram:

- The idea behind this project is to make (simulate) a RAM, which dedicates the Hamming Code before and after the data be written or read.

- In the diagram, we have 2 Dual Port RAM connected to a Data Path, with 2 Hamming Encoders and Decoders. In addition to Noise Element, which complements only one bit at a given position. Plus 2 Registers to hold any value needed at any desired time. The *outter_in* represents any sender like a storage device.

- The simulation consists of sending a sequence of bits from the sender to one of the DPRAM, and then send it again to the other one through Noise Element.


DPRAM 16x11
wr_add1
re_add1
wr_en1
re_en1
clk
reset1
re_data1
wr_data1
Hamming Encoder
Hamming Decoder
enc1_out_noise1_in
Noise Element
noise1_num
noise1_en
noise1_out_buff_in
buff_noise1_en
dec1_in_buff_out
buff_decoder1_en
buff_reg1_en
reg1_in_buff_out
reg1_load
reg1_clear
clk
Reg
reg1_out
data_bus
outter_in
Hamming Encoder
buff_reg2_en
reg2_in_buff_out
reg2_load
reg2_clear
clk
Reg
reg2_out
buff_decoder2_en
buff_noise2_en
dec2_in_buff_out
wr_data2
DPRAM 16x11
wr_add2
re_add2
wr_en2
re_en2
clk
reset2
re_data2
Hamming Decoder
Noise Element
noise2_out_buff_in
noise2_num
noise2_en
Hamming Encoder
enc2_out_noise2_in

- GitHub:

- The whole *code*, along with the diagram, were uploaded to GitHub:

https://github.com/AlAssi69/ErrorCorrectionMemoryPathFinal