|  |  |
| --- | --- |
| Error Correction Encoder & Decoder | **Digital Design and Logical Synthesis for Electrical Computer Engineering**  **(36113611)**  **Course Project** |
| **Digital High Level Design**  **Version 0.1** |

**Revision Log**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rev** | **Change** | **Description** | **Reason for change** | **Done By** | **Date** |
| 0.1 | Initial document | Verification Requirements | Lab2 | Refael Ben Maor  Tal Kapelnik | 14.12.2021 |
| 0.2 |  |  |  |  |  |
| 0.3 |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Table of Content**

[LISt of figures 3](#_Toc89376195)

[List of tables 4](#_Toc89376196)

[1. Blocks Functional Descriptions 5](#_Toc89376197)

[1.1 Top Level – Encoder Decoder & Error Fixer 5](#_Toc89376198)

[1.1.1 Interface 6](#_Toc89376199)

[1.1.2 Block Diagram 7](#_Toc89376200)

[1.1.3 Hierarchy 8](#_Toc89376201)

[1.2 Register Selector 8](#_Toc89376202)

[1.2.1 Interface 8](#_Toc89376203)

[1.2.2 Block Diagram 9](#_Toc89376204)

[1.2.3 Flow Diagram 10](#_Toc89376205)

[1.3 Encoder 11](#_Toc89376206)

[1.3.1 Interface 11](#_Toc89376207)

[1.3.2 Block Diagram 12](#_Toc89376208)

[1.3.3 Flow Diagram 13](#_Toc89376209)

[1.3.4 Main XOR gates for the module 14](#_Toc89376210)

[1.4 Num\_Of\_Errors 16](#_Toc89376211)

[1.4.1 Interface 16](#_Toc89376212)

[1.4.2 Block Diagram 17](#_Toc89376213)

[1.4.3 Flow Diagram 18](#_Toc89376214)

[1.5 Error\_Fix 19](#_Toc89376215)

[1.5.1 Interface 19](#_Toc89376216)

[1.5.2 Block Diagram 20](#_Toc89376217)

[1.5.3 Flow Diagram 21](#_Toc89376218)

[2. Rules in design checker 22](#_Toc89376219)

[2.1 Errors 22](#_Toc89376220)

[2.1.1 Rst error 22](#_Toc89376221)

[2.1.2 Size error 22](#_Toc89376222)

[2.2 Warning 23](#_Toc89376223)

[3. Appendix 24](#_Toc89376224)

[3.1 Terminology 24](#_Toc89376225)

[3.2 References 27](#_Toc89376226)

# LISt of figures

[Figure 1: ECC\_ENC\_DEC Interface. 6](#_Toc89376227)

[Figure 2: ECC\_ENC\_DEC Diagram. 7](#_Toc89376228)

[Figure 3: ECC\_ENC\_DEC Hierarchy. 8](#_Toc89376229)

[Figure 4: Register\_selector Interface. 8](#_Toc89376230)

[Figure 5: Register\_selector Diagram. 9](#_Toc89376231)

[Figure 6: Register\_selector flow diagram. 10](#_Toc89376232)

[Figure 7: Encoder Interface. 11](#_Toc89376233)

[Figure 8: Encoder Diagram. 12](#_Toc89376234)

[Figure 9: Encoder flow diagram. 13](#_Toc89376235)

[Figure 10: Xor logic. 14](#_Toc89376236)

[Figure 11: Num\_Of\_Errors Interface. 16](#_Toc89376237)

[Figure 12: Num\_Of\_Errors Diagram. 17](#_Toc89376238)

[Figure 13: Num\_Of\_Errors flow diagram. 18](#_Toc89376239)

[Figure 14: Error\_fix Interface. 19](#_Toc89376240)

[Figure 15: Error\_fix Diagram. 20](#_Toc89376241)

[Figure 16: Error\_fix flow diagram. 21](#_Toc89376242)

[Figure 17: Rst error. 22](#_Toc89376243)

[Figure 18: Size error. 22](#_Toc89376244)

[Figure 19: warning. 23](#_Toc89376245)

# List of tables

[Table 1: ECC\_ENC\_DEC. 7](#_Toc89376246)

[Table 2: Register\_selector. 9](#_Toc89376247)

[Table 3: Encoder. 12](#_Toc89376248)

[Table 4: How we made the xor logic. 15](#_Toc89376249)

[Table 5: Num\_Of\_errors. 17](#_Toc89376250)

[Table 6: Error\_fix. 20](#_Toc89376251)

[Table 7: Assignment Interface. 24](#_Toc89376252)

[Table 8: Parameters. 24](#_Toc89376253)

[Table 9: Register file description. 25](#_Toc89376254)

[Table 10: CTRL register. 25](#_Toc89376255)

[Table 11: Data In register. 26](#_Toc89376256)

[Table 12: Code width register. 26](#_Toc89376257)

[Table 13: Noise register. 27](#_Toc89376258)

# Verification plan

Diagram

Description automatically generated

For the verification plan we started by first planning how the structure of our test needs to be, and what are the common signals in our interface.  
The signal for us [add the signal]

The second step is creating the Stimulus – the simulator for this verification. This module is responsible on the design signals, that feeds the whole system – except clk and rst that will be an input for the system, and the output of the DUT.

Next we have the Golden Model, representing a machine that give us perfect outputs for the stimulus comment lines. We did it by making a matlab code that creates random inputs, and their parities. On the other hand, we used a System Verilog module to assert the right answer simultaneously with the DUT.

To check whether golden model’s and the DUT’s output match, we used the Functional checker. This module receives the output of both entities and continuously make sure both are on the same page.

Finally, the Coverage module is designed to monitor the input of our system, checking if the verity of the inputs was satisfying enough, for this verification plan.

Stimulusx

Stimulus

Golden Model

Golden Register Bank

Golden Register Bank

Stimulus

Sequence generator

APB Master

Overall\_tb

Fifth full checking

1) Standard Scenarios – Regular Input.

2) Extreme Scenarios – when we get reset

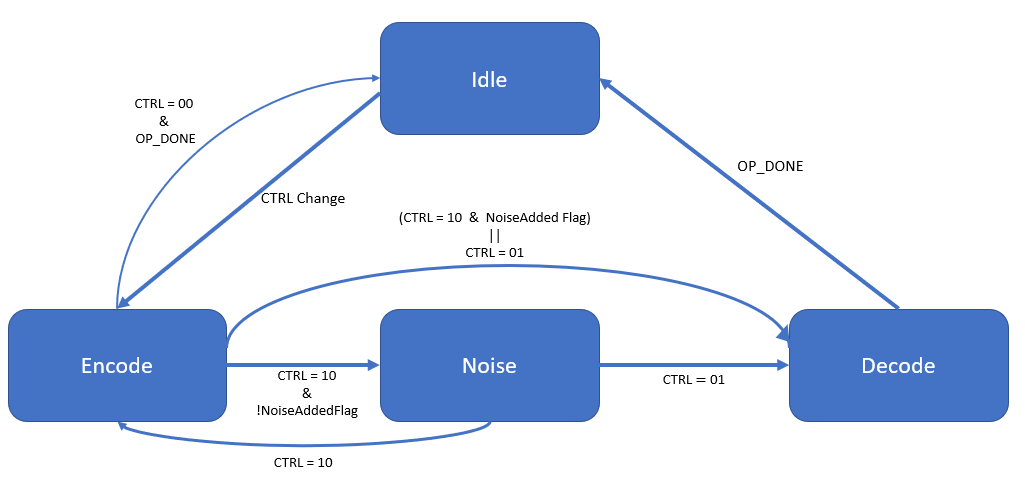
## General Description

**Functional Description**:

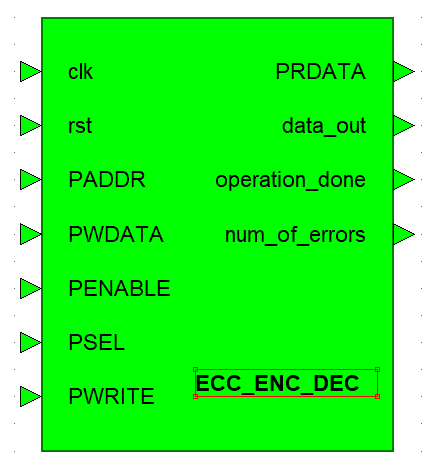
This module is a slave to the CPU , which control it with the APB bus and the register files.  
The CPU send encoder or decoder data to the Error correction Encoder & Decoder diagram block and it give back the data result, operation\_done out put ‘1’ when he finish, and num\_of\_errors that tell if was between 0-2 errors.

The Top-Level Error correction Encoder & Decoder consist 4 modules:  
1) Registor\_selctor – save the information about the Error correction encoder& decoder and initial parameters and send the date to require modules and also the CPU if he want to read from the registers.  
2) Encoder – Get the data from the Registers\_selector and the Top to encode the wanted data, also this module is use for the decoder part to save a space in the design the decoder part using the same components as the decoder.  
3) Num\_of\_errors – Get data from the Top and the encoder to tell us how much errors we have in the wanted data , this data we sending out with num\_of\_errors and sending it to the next module Error\_fix .  
4) Error\_fix – Get data from the Top and the Num\_of\_errors to fix the error in the data if he can, only when we have 1 error we fixing the data at the bit spot that need the correction.

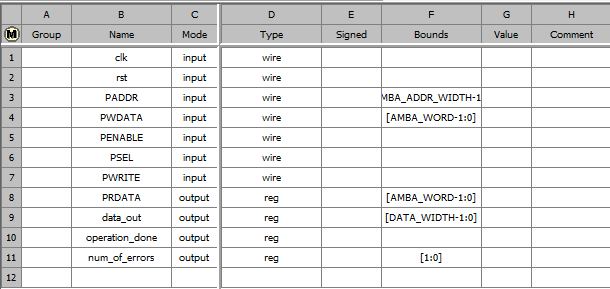
**Top State Machine:**



### Interface

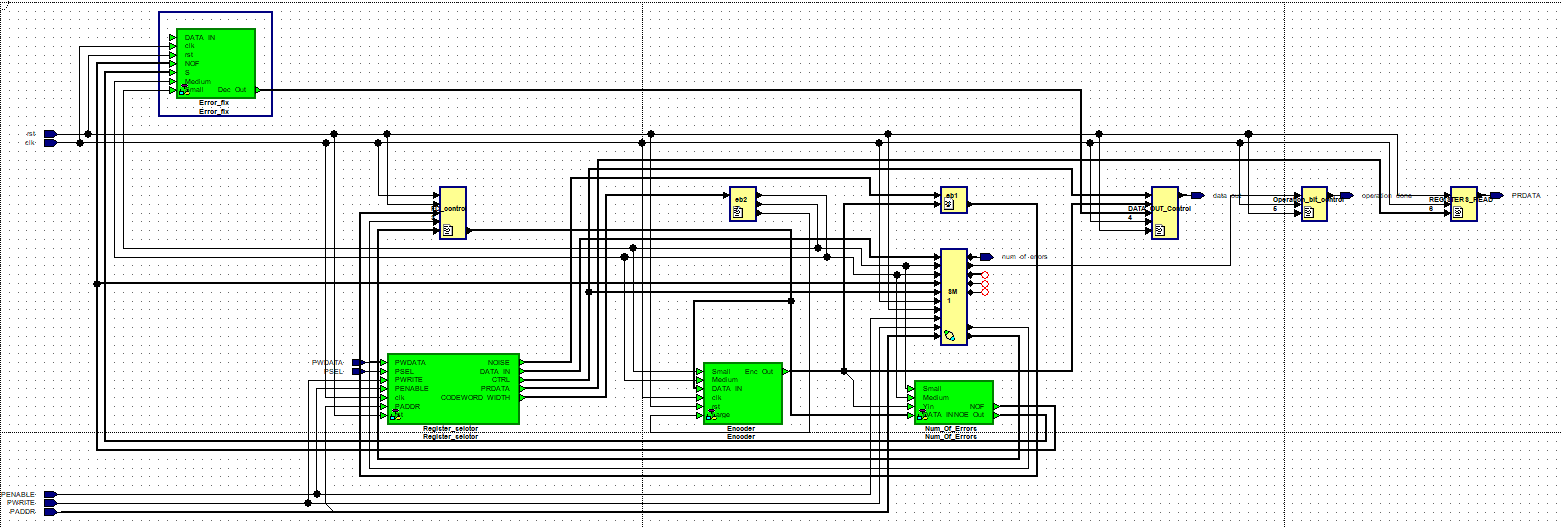


1. ECC\_ENC\_DEC Interface.



1. ECC\_ENC\_DEC.

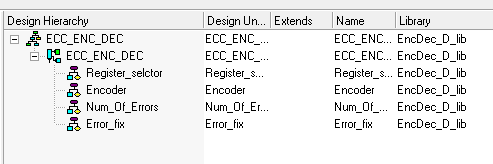
### Block Diagram



1. ECC\_ENC\_DEC Diagram.

### Hierarchy

As we can see the ECC\_ENC\_DEC is the Top-Level of the design



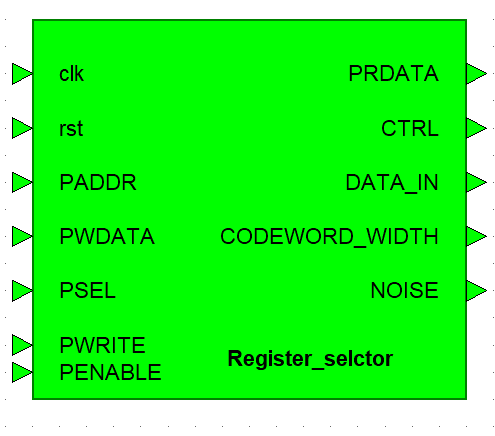
1. ECC\_ENC\_DEC Hierarchy.

## Register Selector

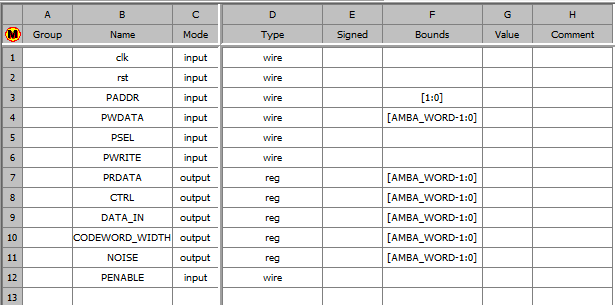
**Functional Description**:

This module has the registers files that contains all the data for the Top-Level to work, all the registers get the data from the CPU via the APB bus . The Error correction Encoder & Decoder module can only read from this module with the wires DATA\_IN\_REG, CTRL\_REG, CODEWORD\_WIDTH\_REG, NOISE\_REG, PRDATA\_REG when the PRDATA\_REG is used to send data from the registers to the CPU when he ask for it.

### Interface



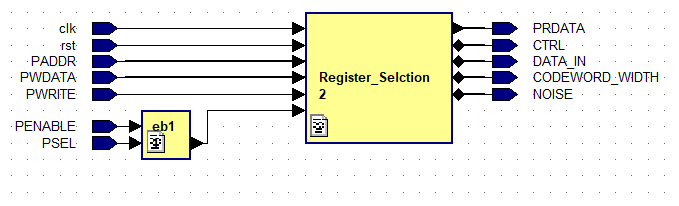
1. Register\_selector Interface.



1. Register\_selector.

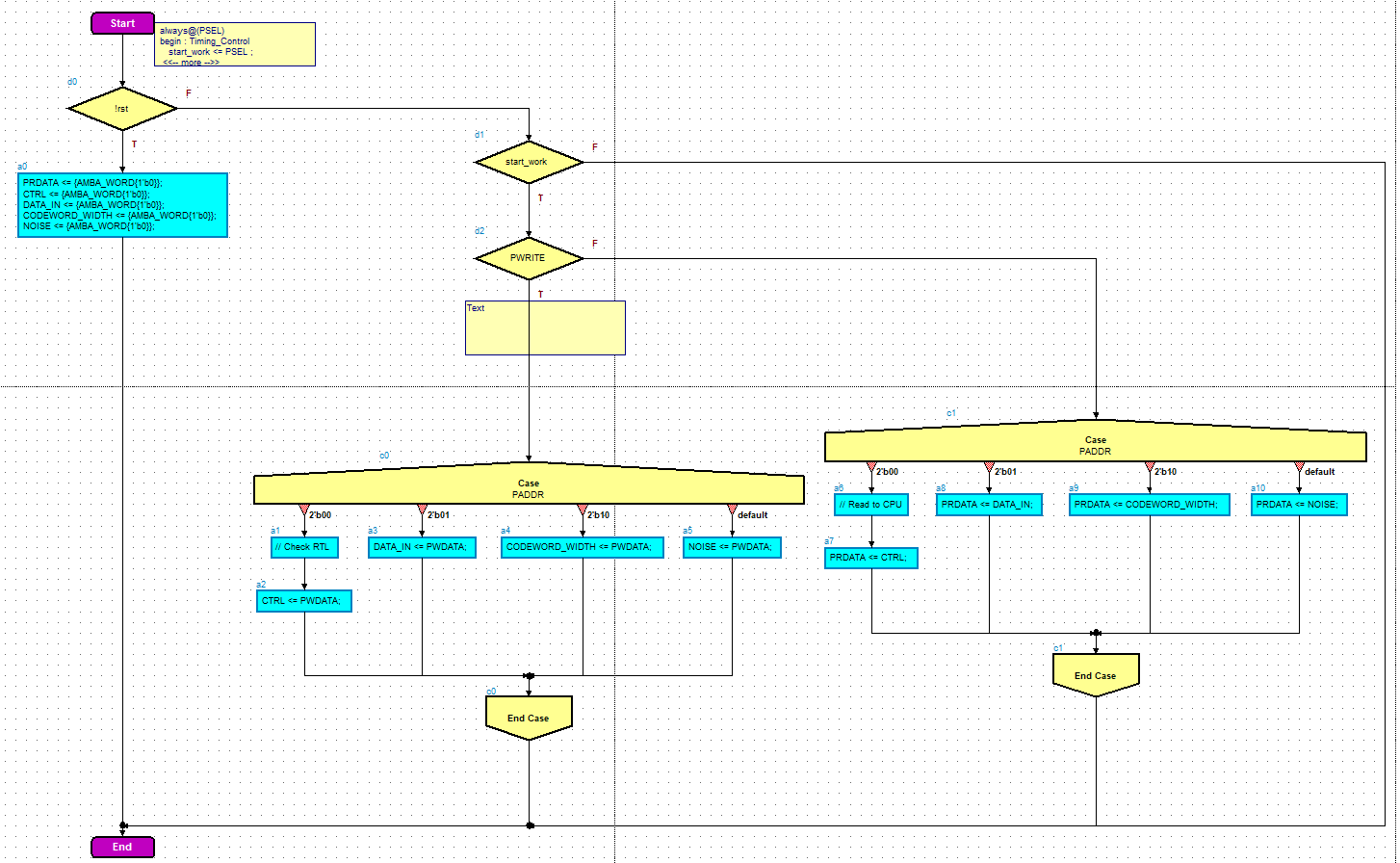
### Block Diagram

Eb1 telling to the block Register\_Selction to save the value that come from the CPU.



1. Register\_selector Diagram.

### Flow Diagram



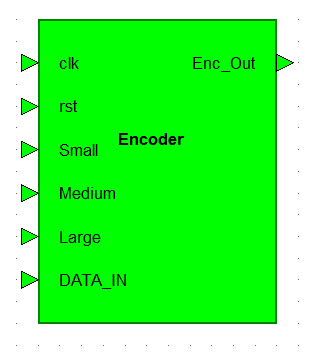
1. Register\_selector flow diagram.

## Encoder

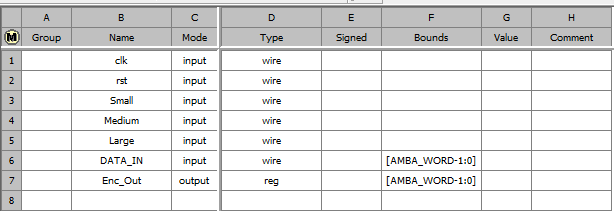
**Functional Description**:

This module gets the data from the top entity and from the Register\_Selctor module. This module is being used in each of the functions available, Encode, Decode, and full channel. For that reason, we put special attention to the details, and tried to minimize the calculation needed, in order to find the parity bits. To support all width sizes, the encoder works for each of the sizes possible. To do that, we set the data to the MSB in the input, padder with zeroes from the left LSBs.

### Interface



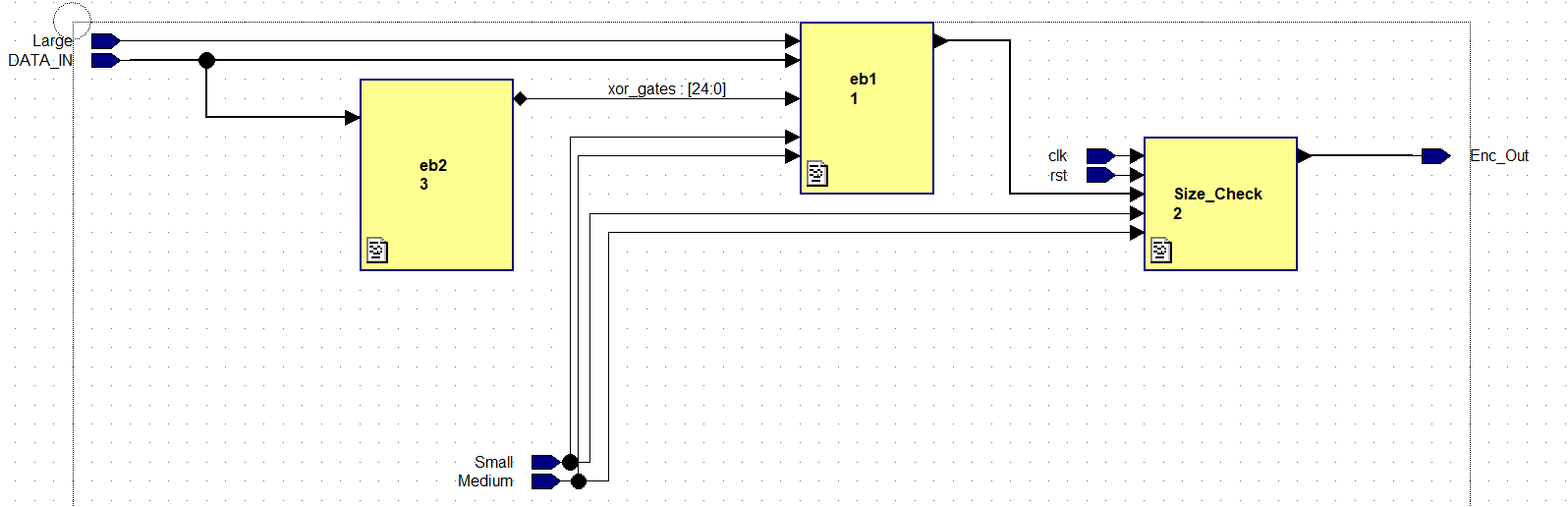
1. Encoder Interface.



1. Encoder.

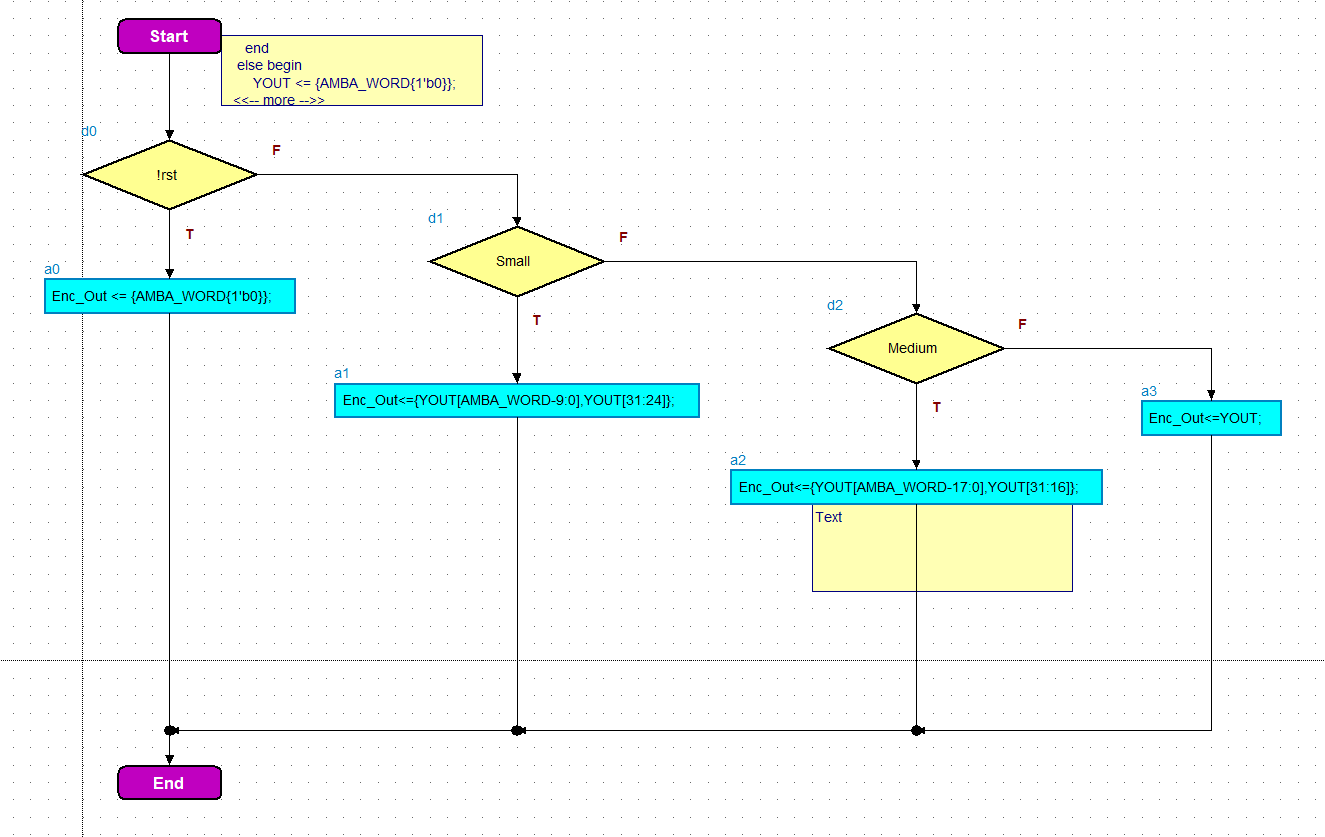
### Block Diagram

Eb1 and eb2 creating the the parity for the encoding part , Size Check is sending the right size to the top   
because encoder working on 32 bits



1. Encoder Diagram.

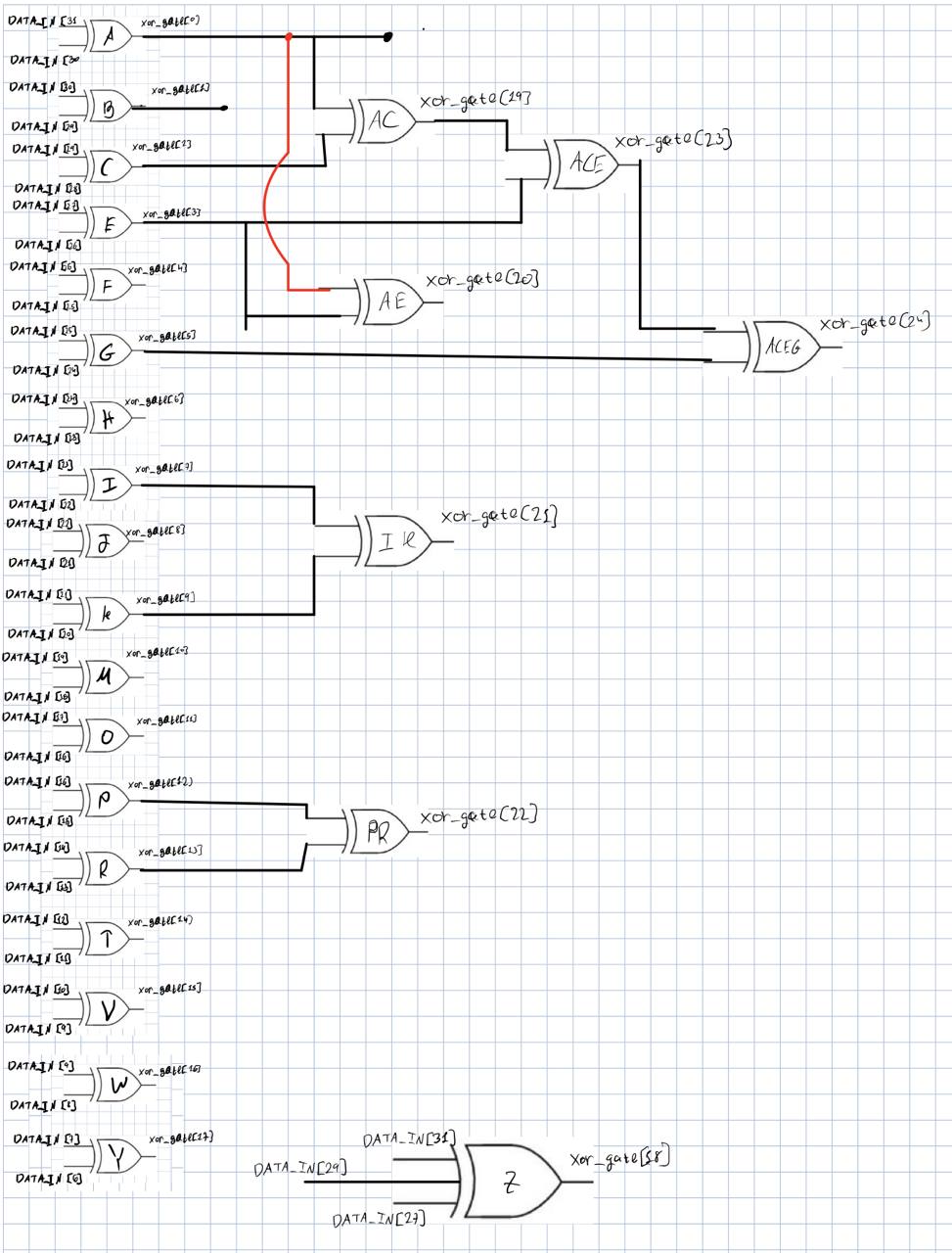
### Flow Diagram



1. Encoder flow diagram.

### Main XOR gates for the module

With this component we calculate the parity, in order to reduce calculation and space (XOR gates) in the design, we minimized the XOR gates by using repeating calculations for C1, … , Cn.   
The xor in the figure 8 is called eb2



1. Xor logic.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | c5 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | c6 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | c7 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | c8 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | c12 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | c13 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | c14 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | c15 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | c16 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | c27 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | c28 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | c29 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | c30 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | c31 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | c32 |
|  |  |  | |  | | Y | | W | | U | | S | | Q | | O | | M | | K | | I | | G | | E | | C | | A | |  |
|  |  |  |  |  |  |  | X | | V | | T | | R | | P | | N | | L | | J | | H | | F | | D | | B | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |  | 31/29/27 = Z | | | | |  |

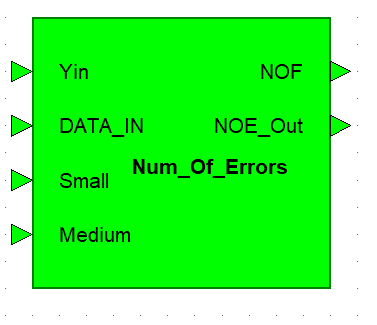
1. How we made the xor logic.

## Num\_Of\_Errors

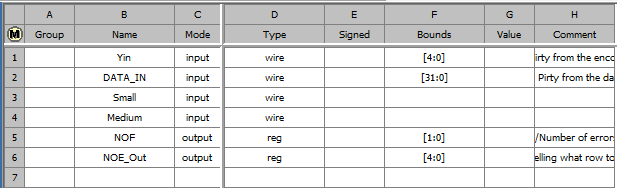
**Functional Description**:

This module gets the data from the Encoder and from the top entity. For the Input of a vector with its parity, the output will be the number of corrupted bits. The number of errors that calculated are sent to the top entity, and from there to Error\_Fix module for further calculation.

### Interface

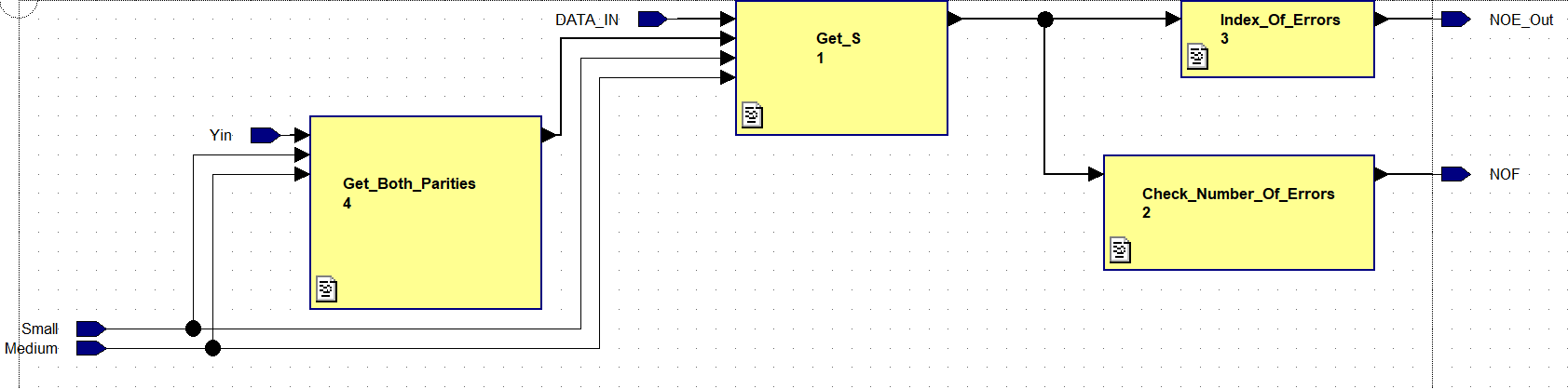


1. Num\_Of\_Errors Interface.



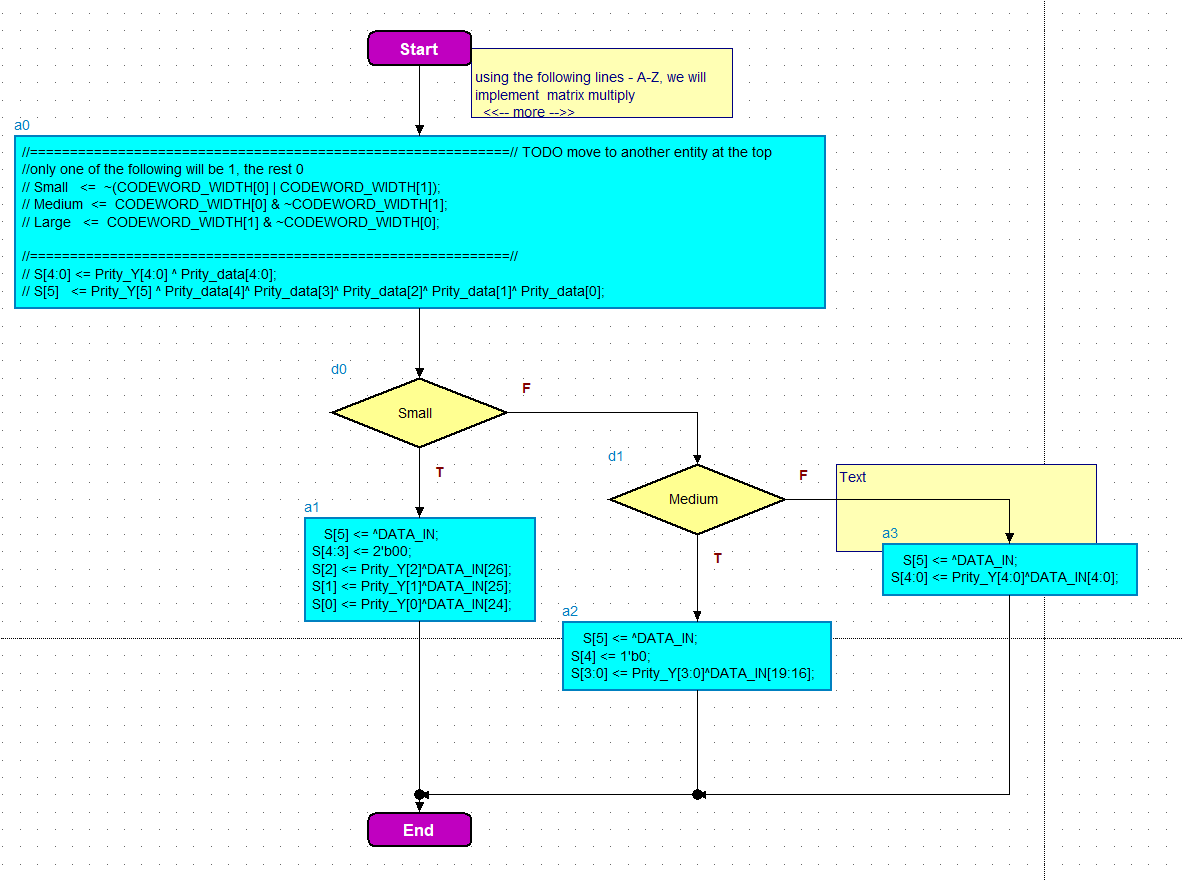
1. Num\_Of\_errors.

### Block Diagram



1. Num\_Of\_Errors Diagram.

### Flow Diagram



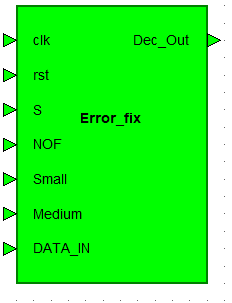
1. Num\_Of\_Errors flow diagram.

## Error\_Fix

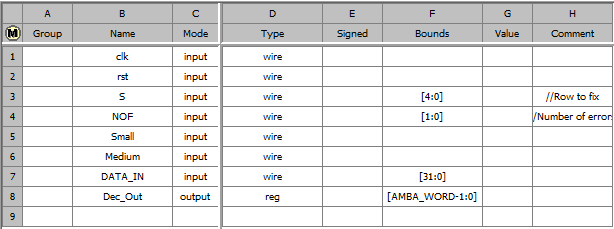
**Functional Description**:

This module gets the data from the Num\_Of\_Errors and Top modules . With the data from the Num\_Of\_Errors he know if he need to fix (only when we have one error) and also know what bit is corrupt and with the data from the Top he send the data\_out and also changing operation\_done to ‘1’ when he done.

### Interface

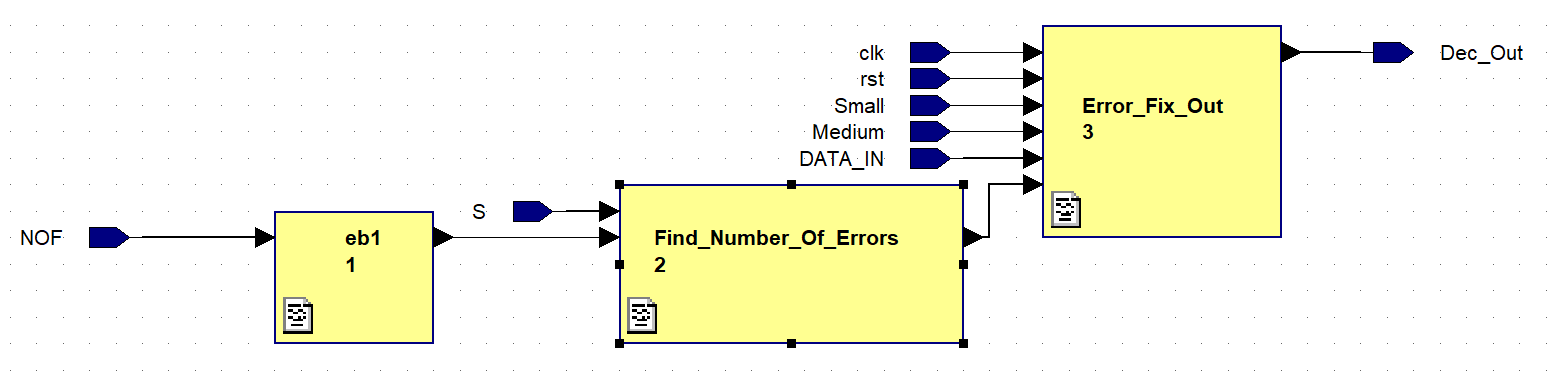


1. Error\_fix Interface.



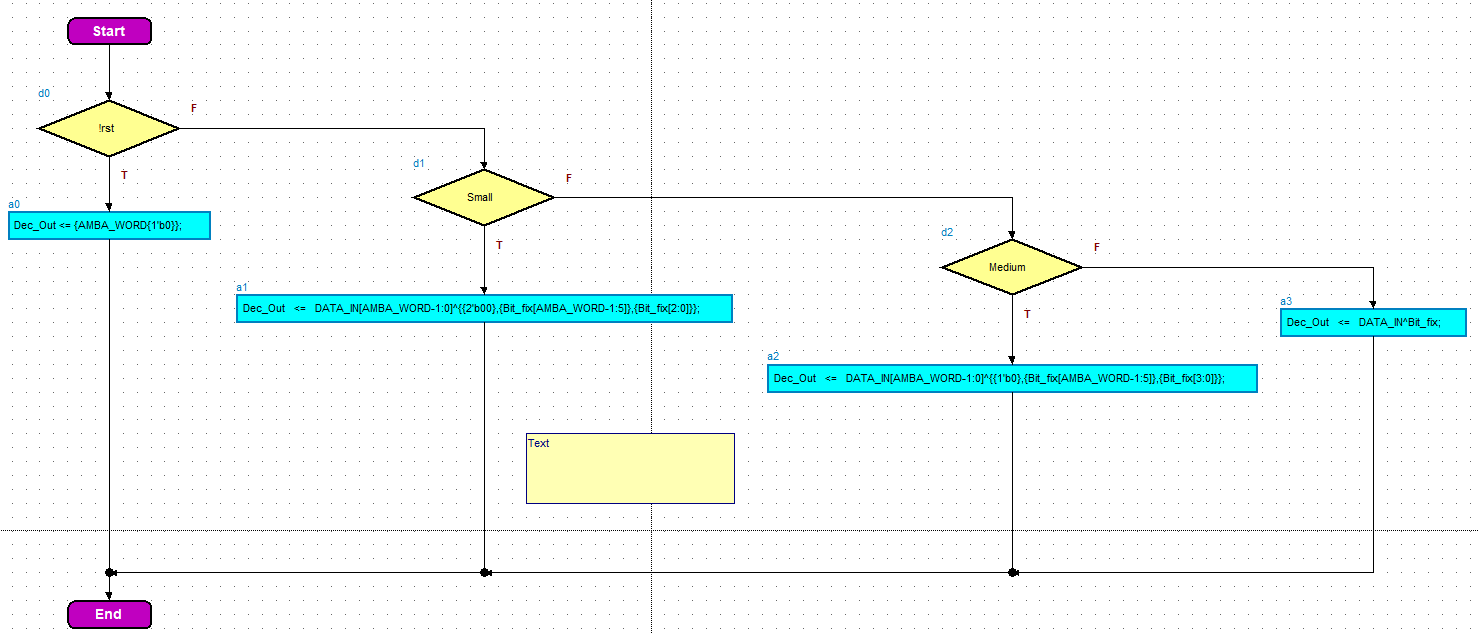
1. Error\_fix.

### Block Diagram



1. Error\_fix Diagram.

### Flow Diagram

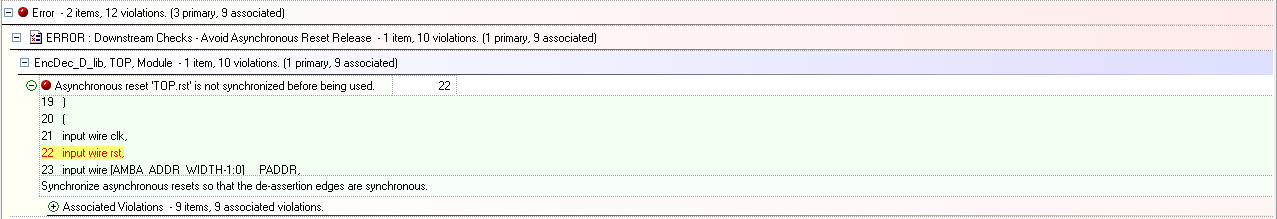


1. Error\_fix flow diagram.

# Rules in design checker

## Errors

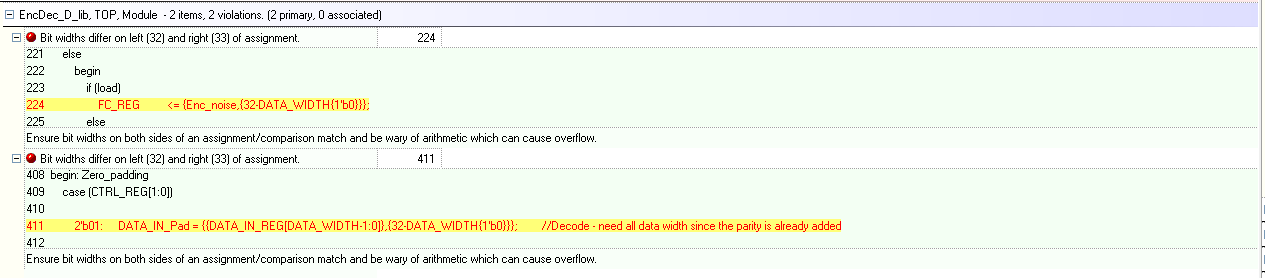
### Rst error



1. Rst error.

Error that was allowed to waver because that not need to show

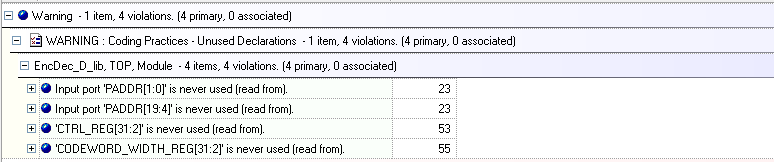
### Size error



1. Size error.

As we can see even though that say we have wrong bit amount we can se ethe even on both sides  
[DATA\_WIDTH-1:0] = DATA\_WIDTH amount of bits  
Data in pad is 32 bit  
so 32 = DATA\_WIDTH – (32 – DATA\_WIDTH) = 32 bits

## Warning

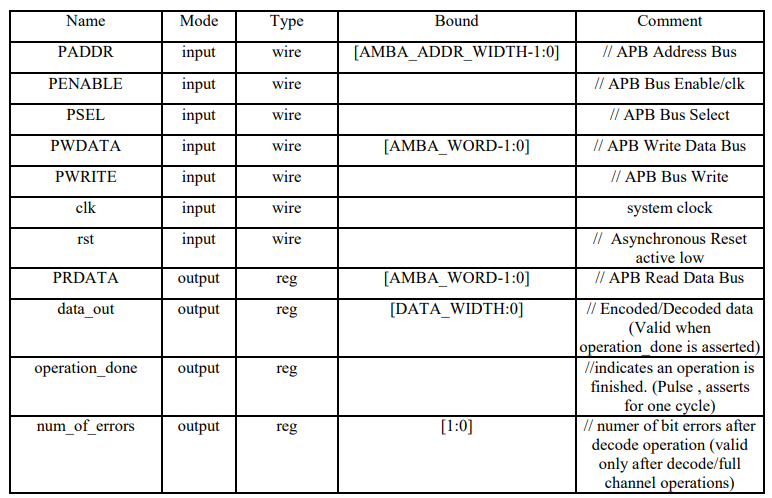


1. warning.

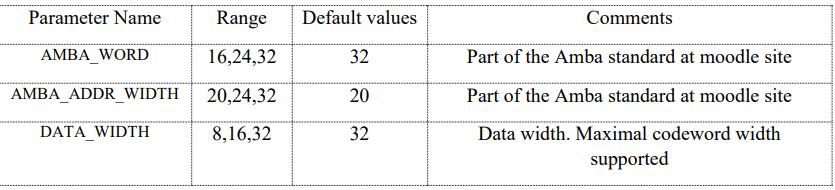
We keep those values for when the CPU reads from the registers, but we don’t use them inside the top structure.

# Appendix

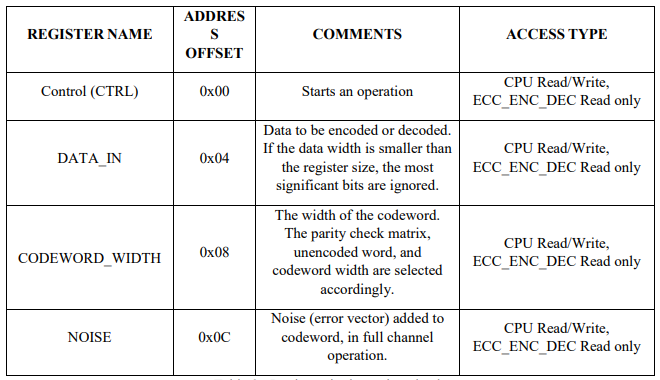
## Terminology



1. Assignment Interface.



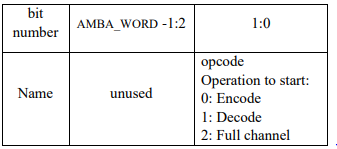
1. Parameters.



1. Register file description.

**CTRL register:**

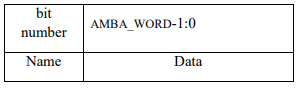
Control register. Writing to the "opcode" field in this register will trigger the chosen operation. Only valid opcodes are allowed to be written. Address: 0x00; default 0.



1. CTRL register.

**Data In Register:**

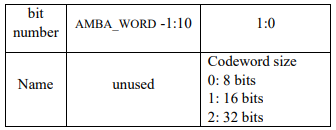
Data to be encoded or decoded. Encoding operation – the data is the unencoded word, that needs to be encoded. Decoding operation – the data is the erroneous data, that needs to be decoded. Full channel operation – the data is an unencoded word, that needs to be encoded, then corrupted, and then decoded.  
The input data may be shorter than Amba\_word – in that case the data is located at the least significant bits (LSB), the most significant bits are ignored. Address offset 0x04; default 0



1. Data In register.

**Codeword width register:**

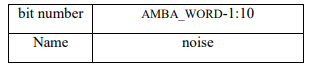
The size of the codeword data. The parity check matrix is selected according to the codeword width (see appendix). The size of the unencoded word is also selected according to the codeword width. Address offset 0x08; default 0



1. Code width register.

**Noise register:**

An additive noise in the noise channel. Used in full channel operation only (this register is ignored in other operations). The value in this register is XORed with the codeword (output from encoder). When the codeword width is shorted than Amba\_word, the noise is located at the LSB (MSB is ignored). Address offset 0x0C; default 0.



1. Noise register.

## References

AMBA APB spec from the moodle  
The lectures from the course