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	First Design of Encoder Decoder		Refael Ben Maor Tal Kapelnik	2.12.2021
0.2					
0.3					

Table of Content

LIST OF FIGURES ... 3

LIST OF TABLES ... 4

1. BLOCKS FUNCTIONAL DESCRIPTIONS ... 5

1.1 Top Level – Encoder Decoder & Error Fixer ... 5

1.1.1 Interface ... 6
1.1.2 Block Diagram ... 7
1.1.3 Hierarchy ... 8

1.2 Register Selector ... 8

1.2.1 Interface ... 8
1.2.2 Block Diagram ... 9
1.2.3 Flow Diagram ... 10

1.3 Encoder ... 11

1.3.1 Interface ... 11
1.3.2 Block Diagram ... 12
1.3.3 Flow Diagram ... 13
1.3.4 Main XOR gates for the module ... 14

1.4 Num_Of_Errors ... 16

1.4.1 Interface ... 16
1.4.2 Block Diagram ... 17
1.4.3 Flow Diagram ... 18

1.5 Error_Fix ... 19

1.5.1 Interface ... 19
1.5.2 Block Diagram ... 20
1.5.3 Flow Diagram ... 21

2. RULES IN DESIGN CHECKER ... 22

2.1 Errors ... 22

2.1.1 Rst error ... 22
2.1.2 Size error ... 22

2.2 Warning ... 23

3. APPENDIX ... 24

3.1 Terminology ... 24

3.2 References ... 27

LIST OF FIGURES

Figure 1: ECC_ENC_DEC Interface. 6
Figure 2: ECC_ENC_DEC Diagram. 7
Figure 3: ECC_ENC_DEC Hierarchy. 8
Figure 4: Register_selector Interface. 8
Figure 5: Register_selector Diagram. 9
Figure 6: Register_selector flow diagram. 10
Figure 7: Encoder Interface. 11
Figure 8: Encoder Diagram. 12
Figure 9: Encoder flow diagram. 13
Figure 10: Xor logic. 14
Figure 11: Num_Of_Errors Interface. 16
Figure 12: Num_Of_Errors Diagram. 17
Figure 13: Num_Of_Errors flow diagram. 18
Figure 14: Error_fix Interface. 19
Figure 15: Error_fix Diagram. 20
Figure 16: Error_fix flow diagram. 21
Figure 17: Rst error. 22
Figure 18: Size error. 22
Figure 19: warning. 23

LIST OF TABLES

Table 1: ECC_ENC_DEC. 7
Table 2: Register_selector. 9
Table 3: Encoder. 12
Table 4: How we made the xor logic. 15
Table 5: Num_Of_errors. 17
Table 6: Error_fix. 20
Table 7: Assignment Interface. 24
Table 8: Parameters. 24
Table 9: Register file description. 25
Table 10: CTRL register. 25
Table 11: Data In register. 26
Table 12: Code width register. 26
Table 13: Noise register. 27

1. BLOCKS FUNCTIONAL DESCRIPTIONS

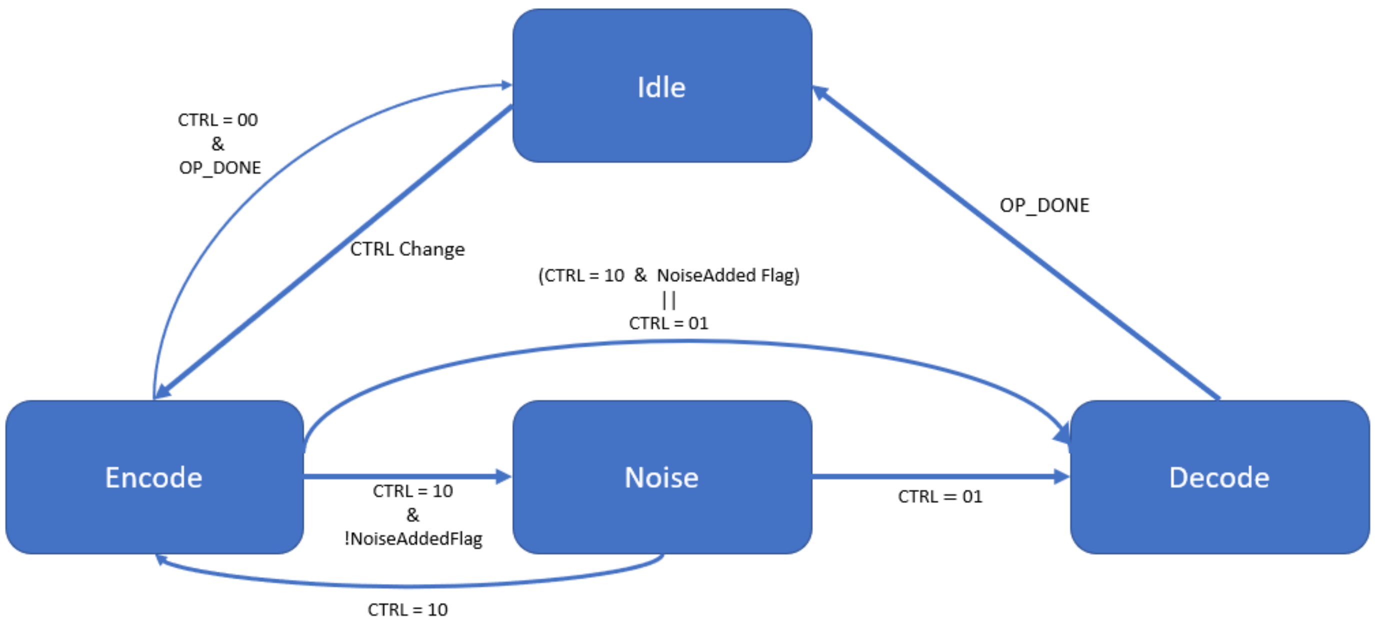
1.1 Top Level – Encoder Decoder & Error Fixer

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:



1.1.1 Interface

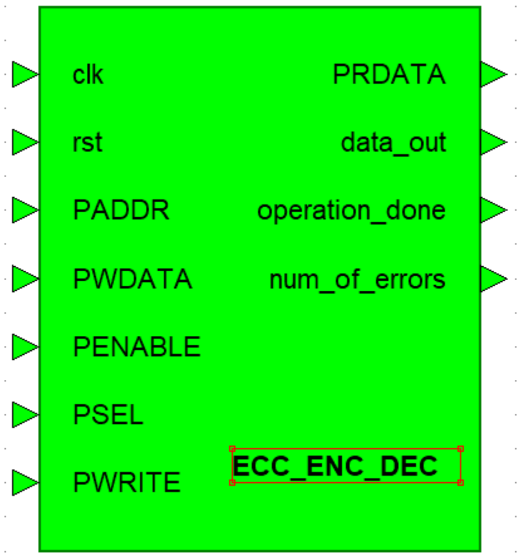
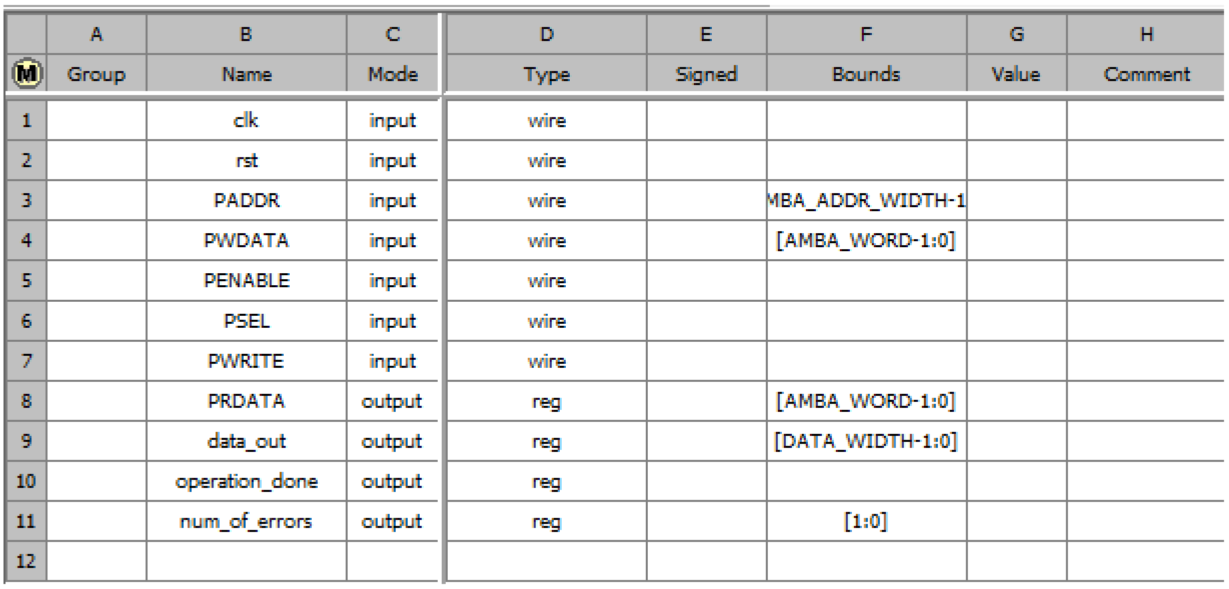


Figure 1: ECC_ENC_DEC Interface.



	A	B	C	D	E	F	G	H
M	Group	Name	Mode	Type	Signed	Bounds	Value	Comment
1		clk	input	wire				
2		rst	input	wire				
3		PADDR	input	wire		MBA_ADDR_WIDTH-1		
4		PWDATA	input	wire		[AMBA_WORD-1:0]		
5		PENABLE	input	wire				
6		PSEL	input	wire				
7		PWRITE	input	wire				
8		PRDATA	output	reg		[AMBA_WORD-1:0]		
9		data_out	output	reg		[DATA_WIDTH-1:0]		
10		operation_done	output	reg				
11		num_of_errors	output	reg		[1:0]		
12								

Table 1: ECC_ENC_DEC.

1.1.2 Block Diagram

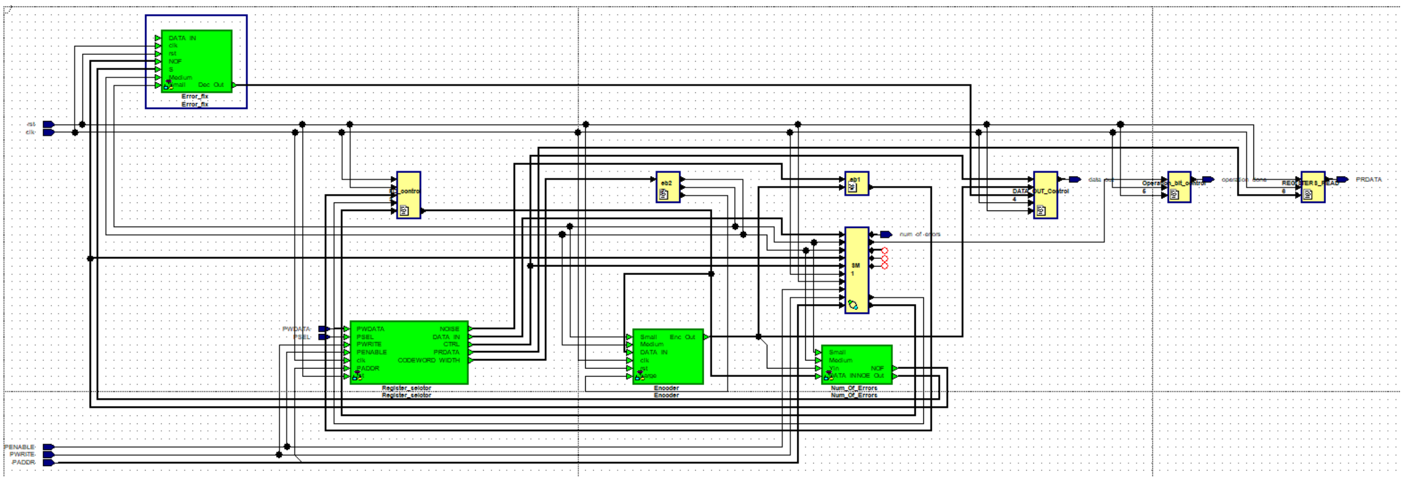
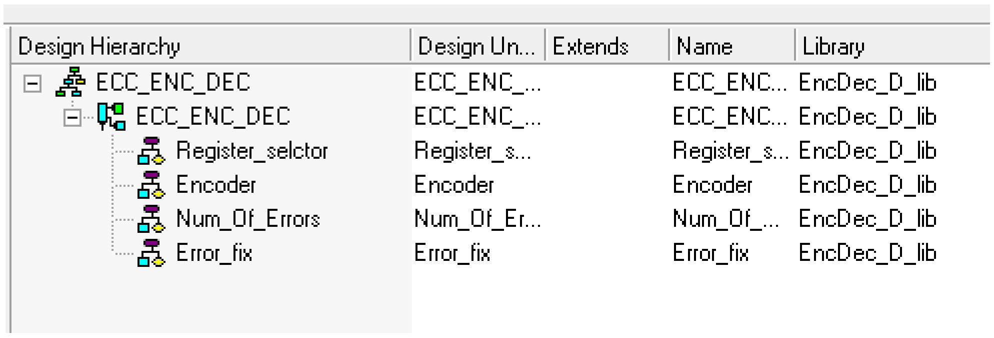


Figure 2: ECC_ENC_DEC Diagram.

1.1.3 Hierarchy

As we can see the ECC_ENC_DEC is the Top-Level of the design



Design Hierarchy	Design Un...	Extends	Name	Library
⊟ ECC_ENC_DEC	ECC_ENC_...		ECC_ENC...	EncDec_D_lib
⊟ ECC_ENC_DEC	ECC_ENC_...		ECC_ENC...	EncDec_D_lib
Register_selctor	Register_s...		Register_s...	EncDec_D_lib
Encoder	Encoder		Encoder	EncDec_D_lib
Num_Of_Errors	Num_Of_Er...		Num_Of_...	EncDec_D_lib
Error_fix	Error_fix		Error_fix	EncDec_D_lib

Figure 3: ECC_ENC_DEC Hierarchy.

1.2 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.

1.2.1 Interface

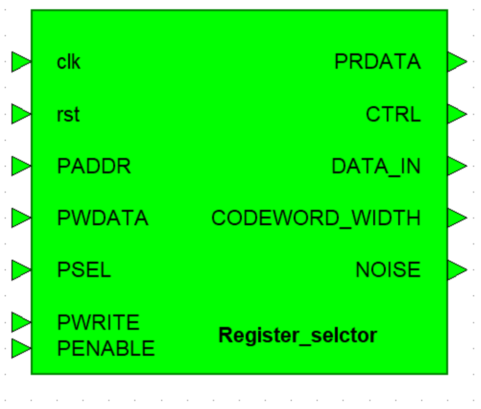
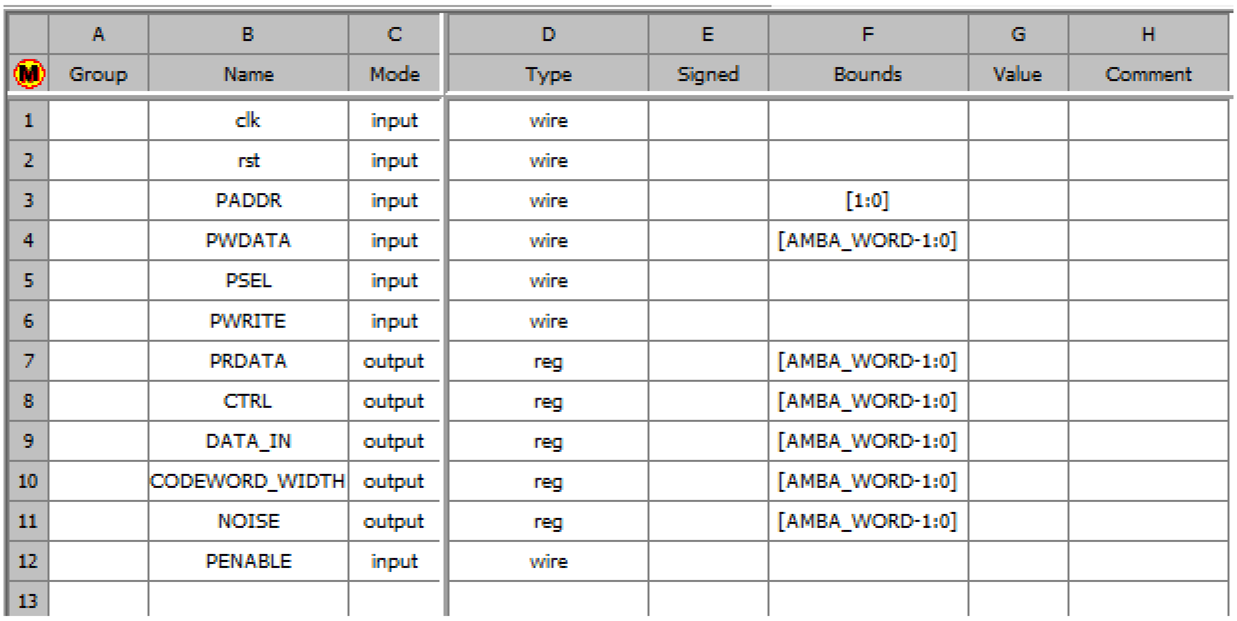


Figure 4: Register_selector Interface.

Classification:	**Template Title:**	Owner	Creation Date	Page
Logic Design course	Block High Level Design	Amir Kolaman	3, November, 2015	8 of 27



	A	B	C	D	E	F	G	H
M	Group	Name	Mode	Type	Signed	Bounds	Value	Comment
1		clk	input	wire				
2		rst	input	wire				
3		PADDR	input	wire		[1:0]		
4		PWDATA	input	wire		[AMBA_WORD-1:0]		
5		PSEL	input	wire				
6		PWRITE	input	wire				
7		PRDATA	output	reg		[AMBA_WORD-1:0]		
8		CTRL	output	reg		[AMBA_WORD-1:0]		
9		DATA_IN	output	reg		[AMBA_WORD-1:0]		
10		CODEWORD_WIDTH	output	reg		[AMBA_WORD-1:0]		
11		NOISE	output	reg		[AMBA_WORD-1:0]		
12		PENABLE	input	wire				
13								

Table 2: Register_selector.

1.2.2 Block Diagram

Eb1 telling to the block Register_Selction to save the value that come from the CPU.

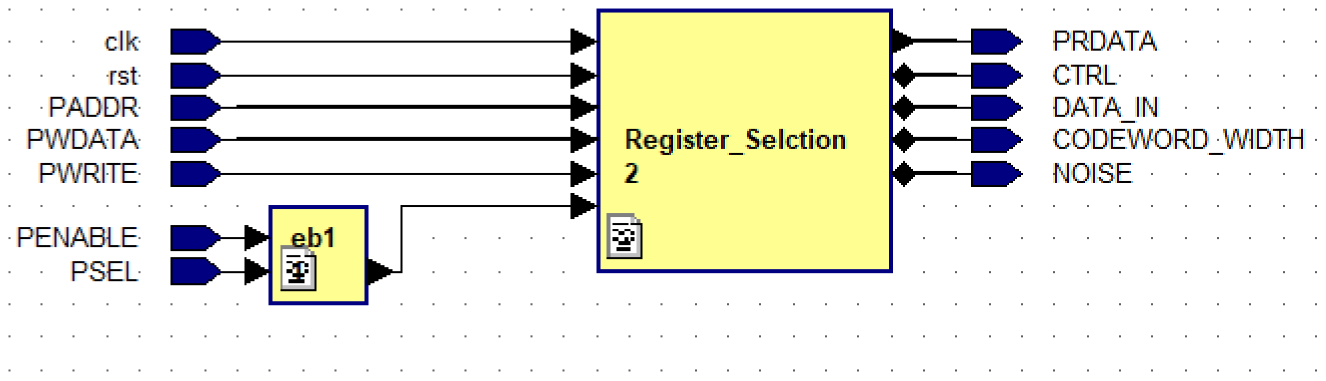


Figure 5: Register_selector Diagram.

Classification:	**Template Title:**	Owner	Creation Date	Page
Logic Design course	Block High Level Design	Amir Kolaman	3, November, 2015	9 of 27

1.2.3 Flow Diagram

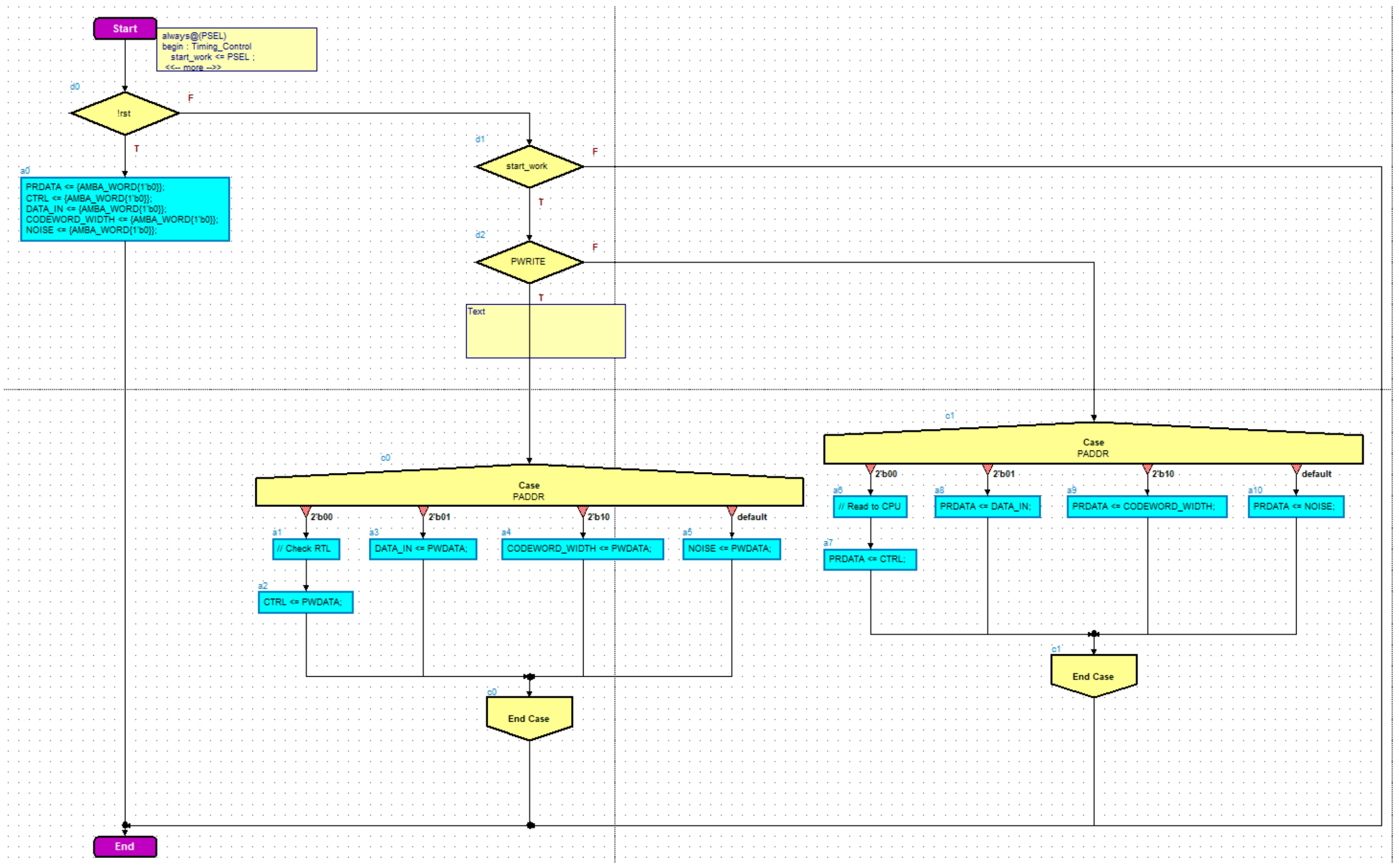


Figure 6: Register_selector flow diagram.

Classification:	**Template Title:**	Owner	Creation Date	Page
Logic Design course	Block High Level Design	Amir Kolaman	3, November, 2015	10 of 27

1.3 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.

1.3.1 Interface

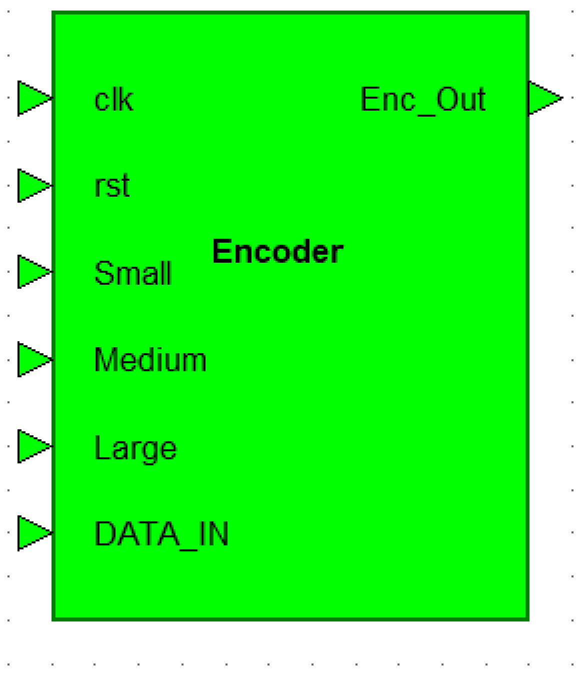
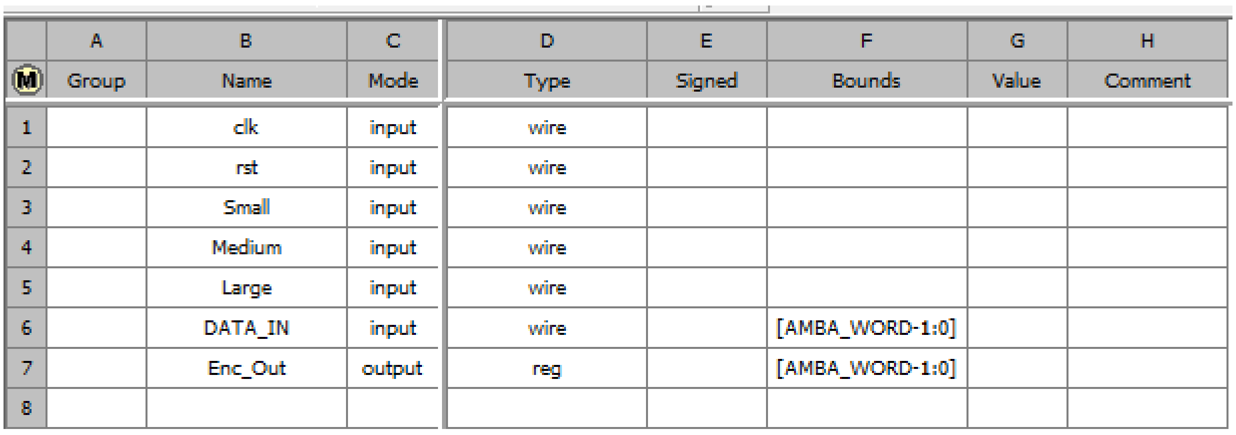


Figure 7: Encoder Interface.



	A	B	C	D	E	F	G	H
Ⓜ	Group	Name	Mode	Type	Signed	Bounds	Value	Comment
1		clk	input	wire				
2		rst	input	wire				
3		Small	input	wire				
4		Medium	input	wire				
5		Large	input	wire				
6		DATA_IN	input	wire		[AMBA_WORD-1:0]		
7		Enc_Out	output	reg		[AMBA_WORD-1:0]		
8								

Table 3: Encoder.

1.3.2 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

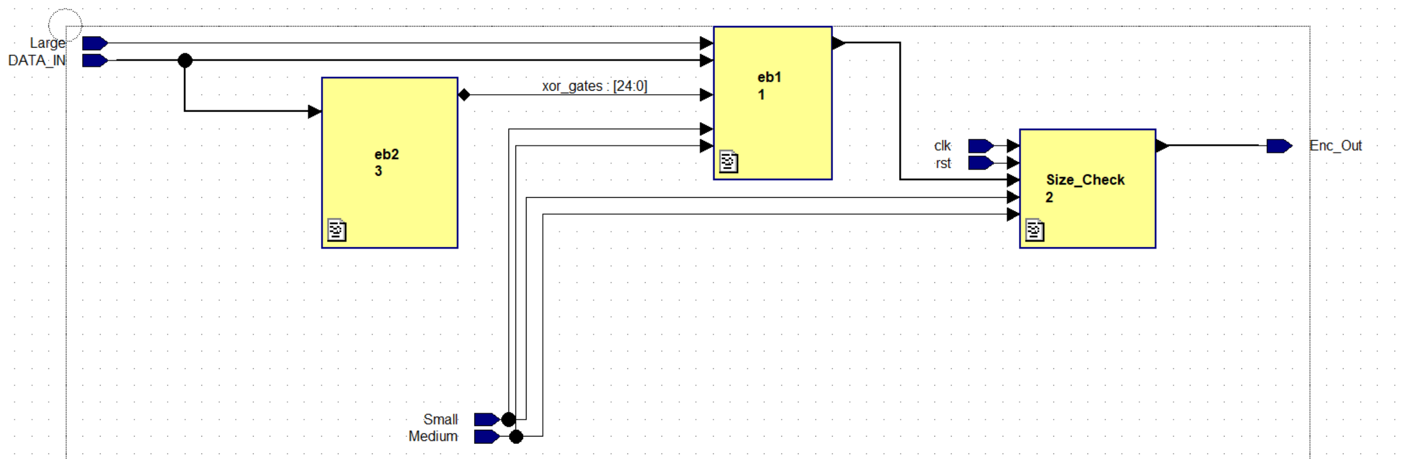


Figure 8: Encoder Diagram.

1.3.3 Flow Diagram

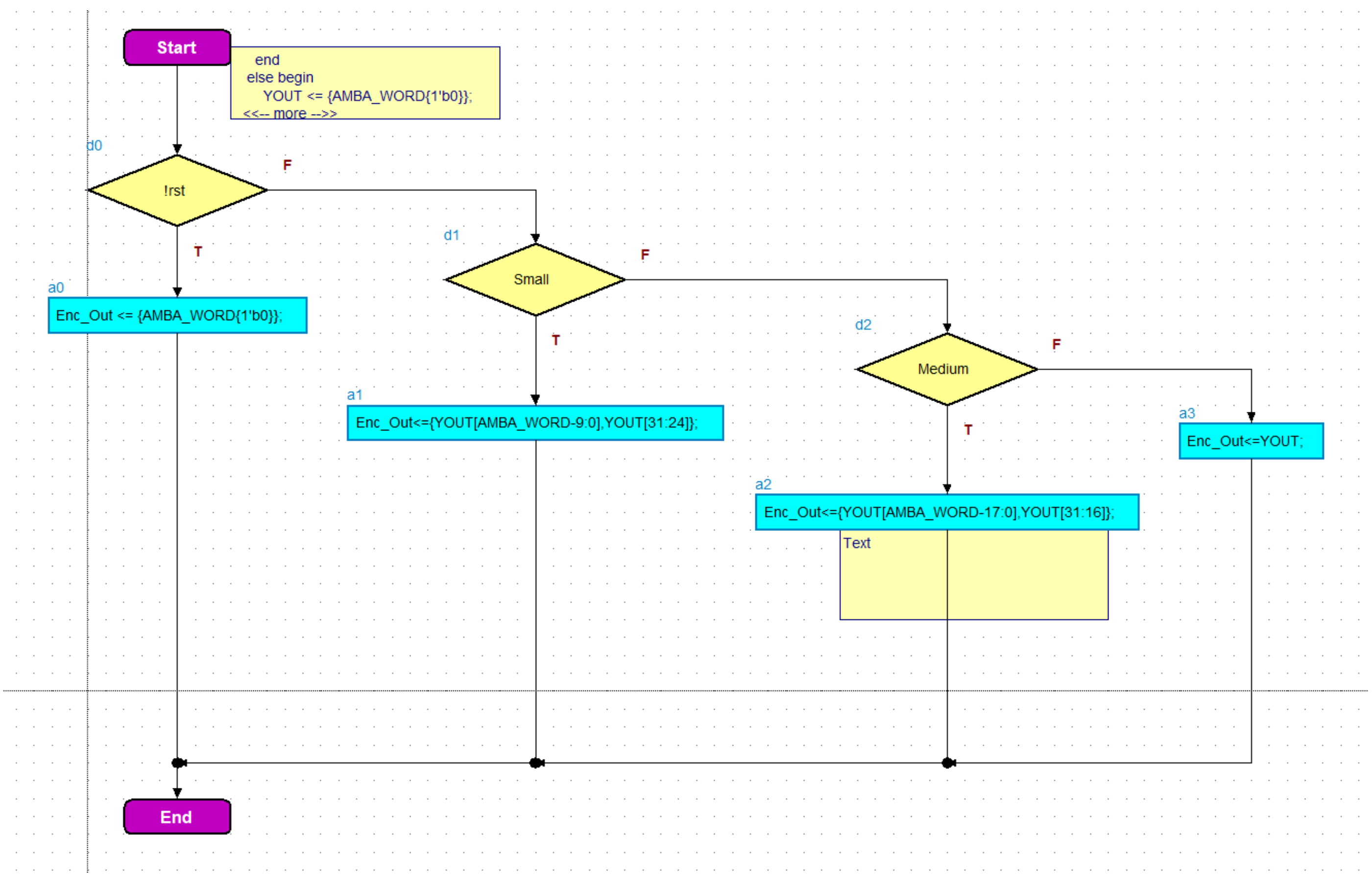

Start
end
else begin
YOUT <= {AMBA_WORD{1'b0}};
<<-- more -->>
d0
!rst
F
T
a0
Enc_Out <= {AMBA_WORD{1'b0}};
d1
Small
F
T
a1
Enc_Out<={YOUT[AMBA_WORD-9:0],YOUT[31:24]};
d2
Medium
F
T
a3
Enc_Out<=YOUT;
a2
Enc_Out<={YOUT[AMBA_WORD-17:0],YOUT[31:16]};
Text
End

Figure 9: Encoder flow diagram.

1.3.4 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

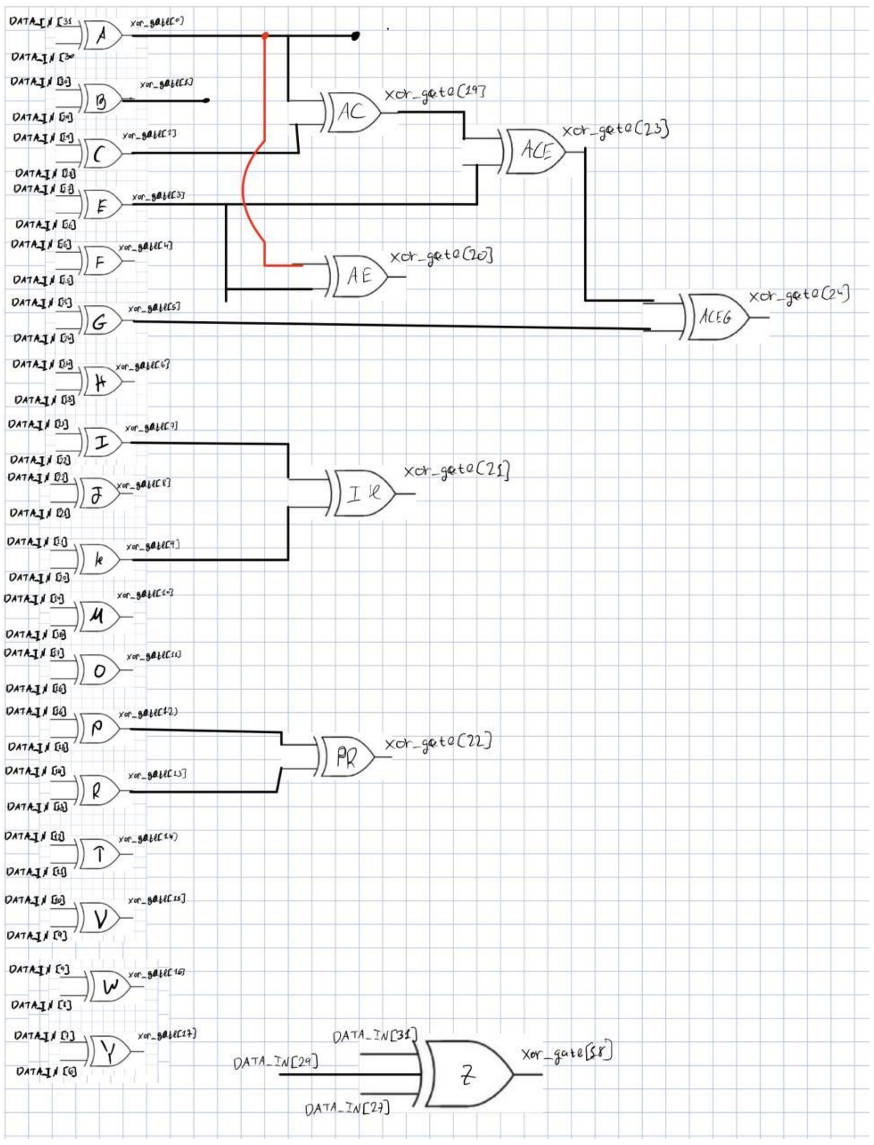


Figure 10: Xor logic.

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

Table 4: How we made the xor logic.

1.4 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.

1.4.1 Interface

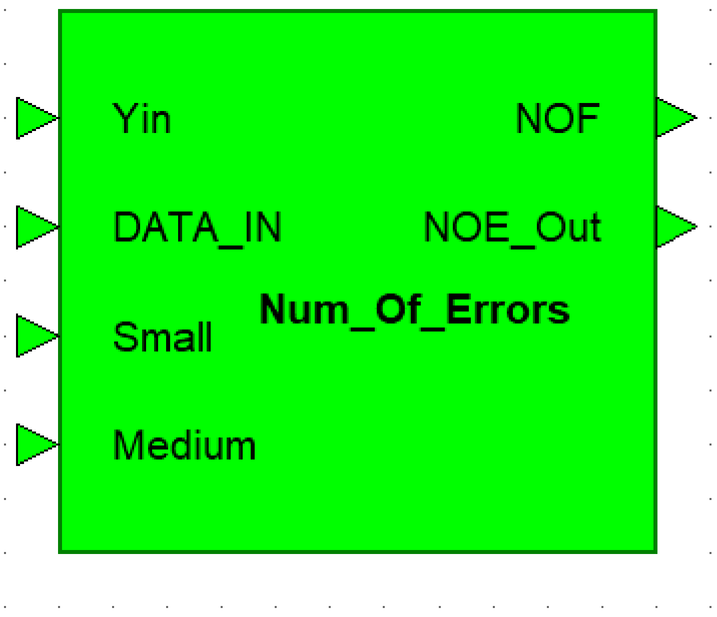
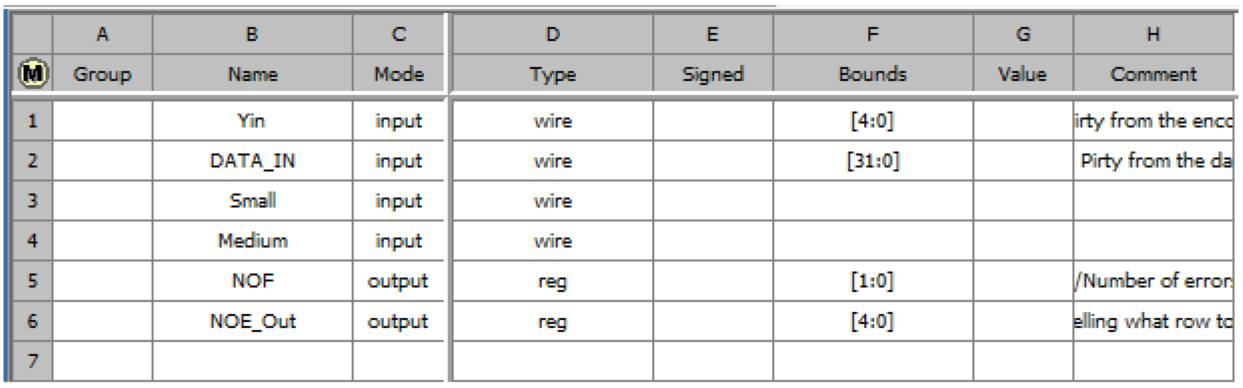


Figure 11: Num_Of_Errors Interface.



A	B	C	D	E	F	G	H
Group	Name	Mode	Type	Signed	Bounds	Value	Comment
1	Yin	input	wire		[4:0]		irty from the enco
2	DATA_IN	input	wire		[31:0]		Pirty from the da
3	Small	input	wire				
4	Medium	input	wire				
5	NOF	output	reg		[1:0]		/Number of error
6	NOE_Out	output	reg		[4:0]		elling what row to
7							

Table 5: Num_Of_errors.

1.4.2 Block Diagram

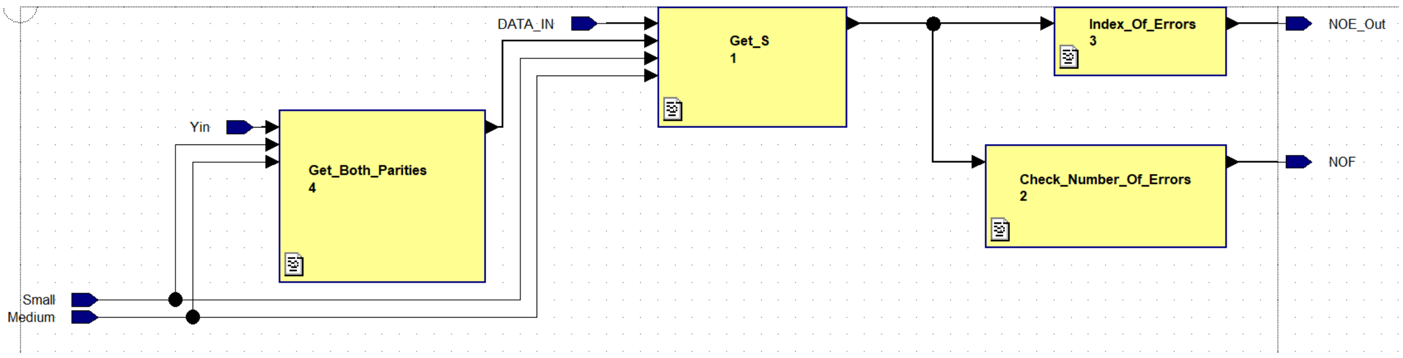
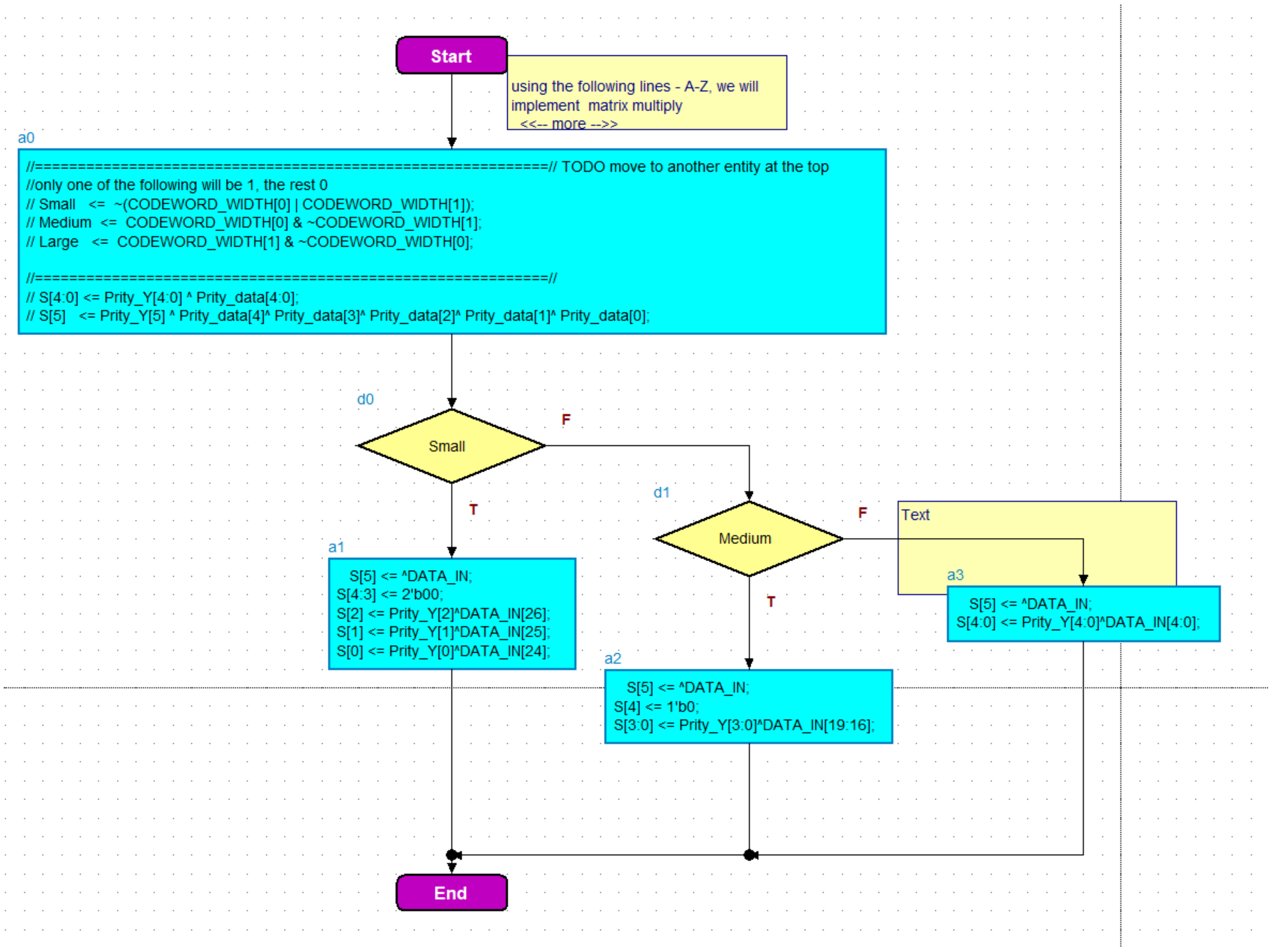


Figure 12: Num_Of_Errors Diagram.

Classification:	**Template Title:**	Owner	Creation Date	Page
Logic Design course	Block High Level Design	Amir Kolaman	3, November, 2015	17 of 27

1.4.3 Flow Diagram


Start
using the following lines - A-Z, we will implement matrix multiply
<<-- more -->>
a0
d0
Small
F
T
a1
d1
Medium
F
Text
a3
T
a2
End

```
//============================================================// TODO move to another entity at the top
//only one of the following will be 1, the rest 0
// Small   <=  ~(CODEWORD_WIDTH[0] | CODEWORD_WIDTH[1]);
// Medium  <=  CODEWORD_WIDTH[0] & ~CODEWORD_WIDTH[1];
// Large   <=  CODEWORD_WIDTH[1] & ~CODEWORD_WIDTH[0];

//============================================================//
// S[4:0] <= Prity_Y[4:0] ^ Prity_data[4:0];
// S[5]   <= Prity_Y[5] ^ Prity_data[4]^ Prity_data[3]^ Prity_data[2]^ Prity_data[1]^ Prity_data[0];
```

```
   S[5] <= ^DATA_IN;
S[4:3] <= 2'b00;
S[2] <= Prity_Y[2]^DATA_IN[26];
S[1] <= Prity_Y[1]^DATA_IN[25];
S[0] <= Prity_Y[0]^DATA_IN[24];
```

```
   S[5] <= ^DATA_IN;
S[4] <= 1'b0;
S[3:0] <= Prity_Y[3:0]^DATA_IN[19:16];
```

```
   S[5] <= ^DATA_IN;
S[4:0] <= Prity_Y[4:0]^DATA_IN[4:0];
```

Figure 13: Num_Of_Errors flow diagram.

1.5 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.

1.5.1 Interface

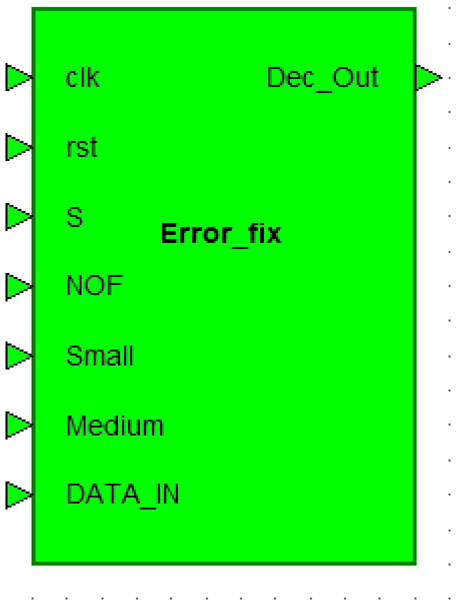
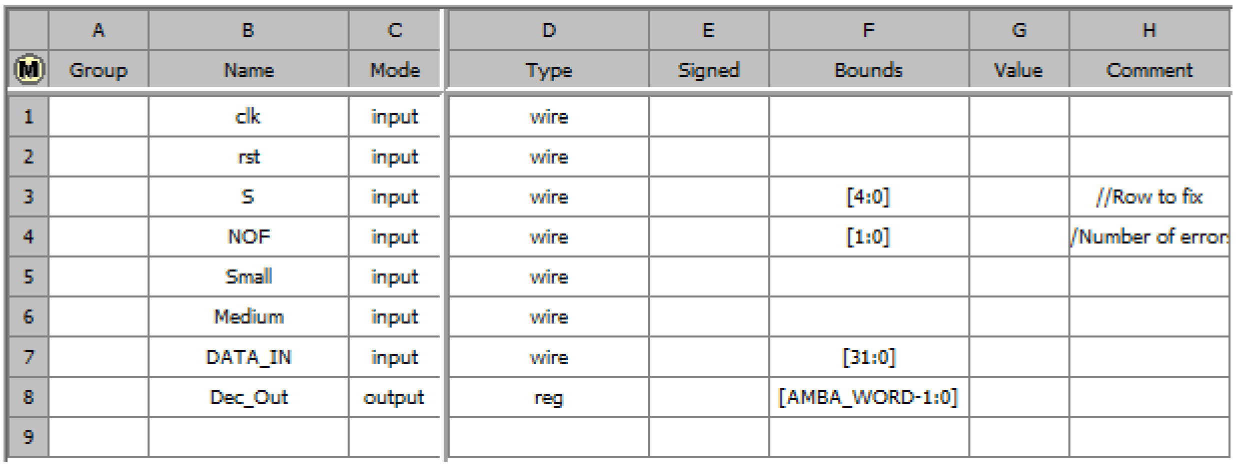


Figure 14: Error_fix Interface.



	A	B	C	D	E	F	G	H
M	Group	Name	Mode	Type	Signed	Bounds	Value	Comment
1		clk	input	wire				
2		rst	input	wire				
3		S	input	wire		[4:0]		//Row to fix
4		NOF	input	wire		[1:0]		/Number of error
5		Small	input	wire				
6		Medium	input	wire				
7		DATA_IN	input	wire		[31:0]		
8		Dec_Out	output	reg		[AMBA_WORD-1:0]		
9								

Table 6: Error_fix.

1.5.2 Block Diagram

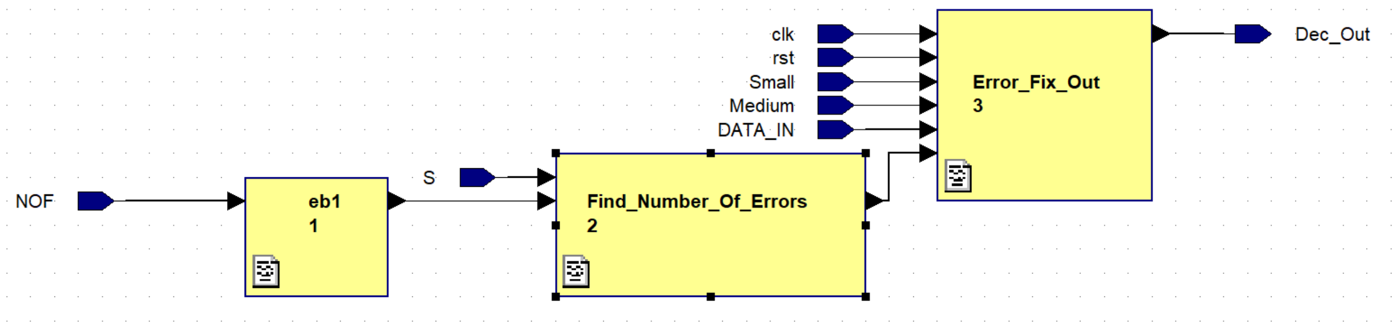
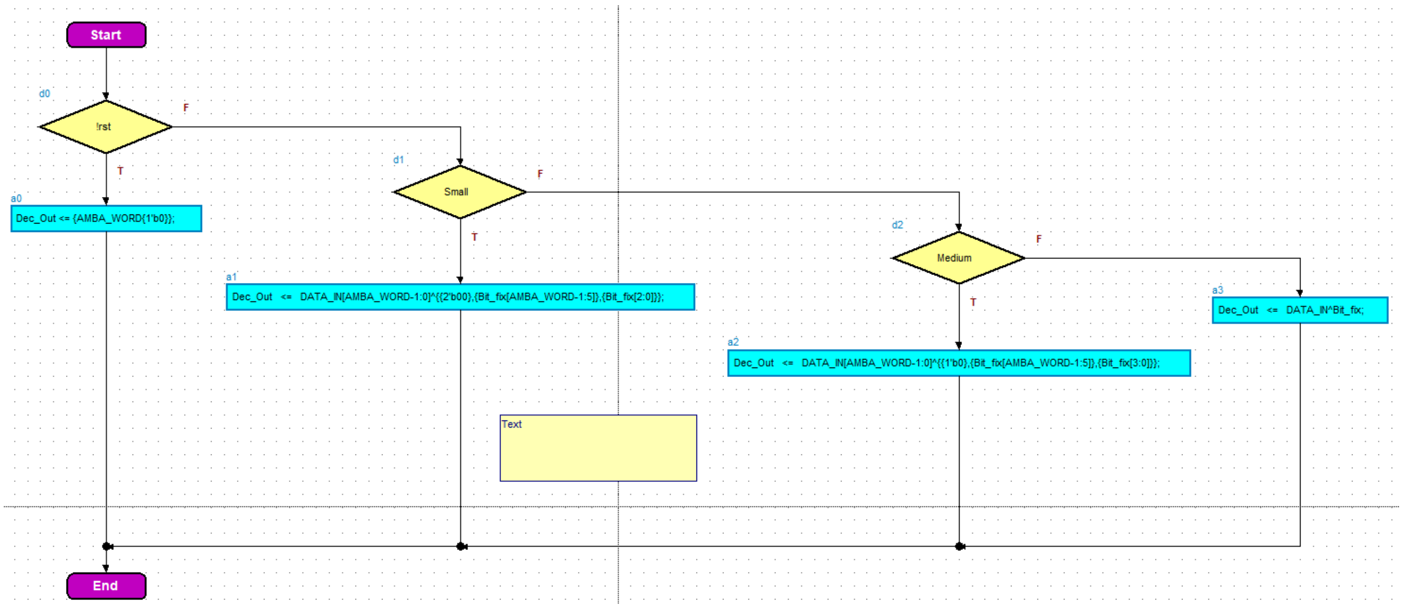


Figure 15: Error_fix Diagram.

1.5.3 Flow Diagram



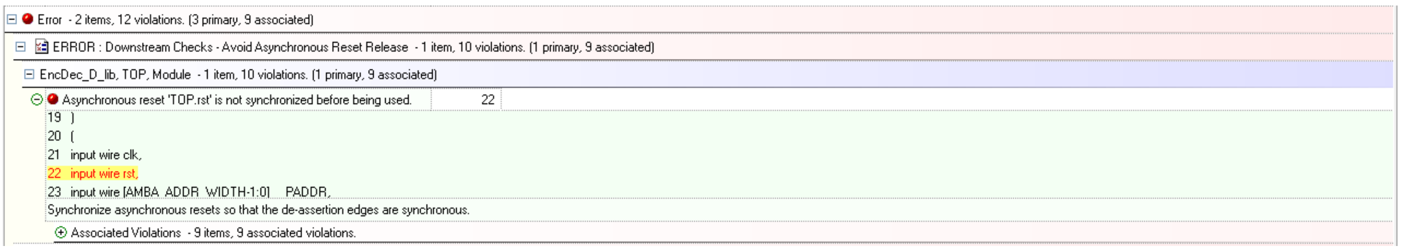
Start
d0
!rst
F
T
a0
Dec_Out <= {AMBA_WORD{1'b0}};
d1
Small
F
T
a1
Dec_Out <= DATA_IN[AMBA_WORD-1:0]^{{2'b00},{Bit_fix[AMBA_WORD-1:5]},{Bit_fix[2:0]}};
d2
Medium
F
T
a2
Dec_Out <= DATA_IN[AMBA_WORD-1:0]^{{1'b0},{Bit_fix[AMBA_WORD-1:5]},{Bit_fix[3:0]}};
a3
Dec_Out <= DATA_IN^Bit_fix;
Text
End

Figure 16: Error_fix flow diagram.

2. RULES IN DESIGN CHECKER

2.1 Errors

2.1.1 Rst error

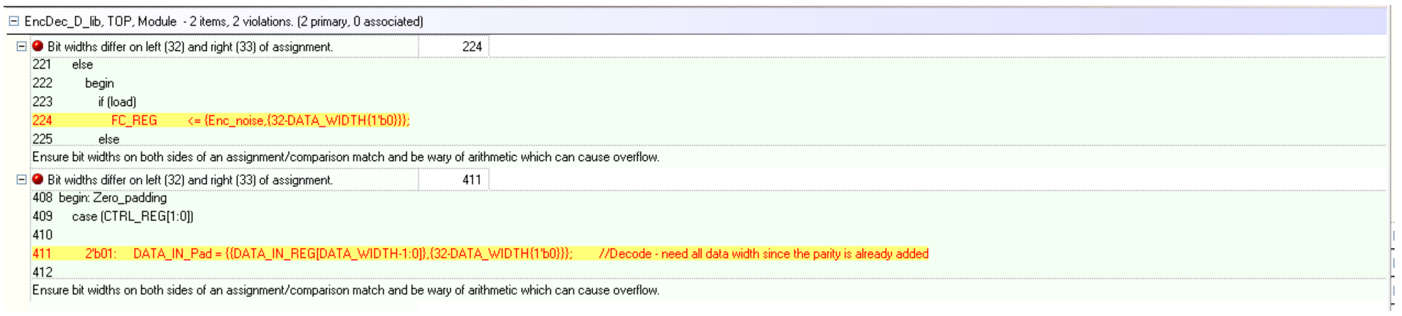


```
⊟ Error  - 2 items, 12 violations. (3 primary, 9 associated)
  ⊟ ERROR : Downstream Checks - Avoid Asynchronous Reset Release  - 1 item, 10 violations. (1 primary, 9 associated)
    ⊟ EncDec_D_lib, TOP, Module  - 1 item, 10 violations. (1 primary, 9 associated)
      ⊖ Asynchronous reset 'TOP.rst' is not synchronized before being used.        22
        19  )
        20  (
        21  input wire clk,
        22  input wire rst,
        23  input wire [AMBA_ADDR_WIDTH-1:0]    PADDR,
        Synchronize asynchronous resets so that the de-assertion edges are synchronous.
          ⊕ Associated Violations  - 9 items, 9 associated violations.
```

Figure 17: Rst error.

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

2.1.2 Size error



```
⊟ EncDec_D_lib, TOP, Module  - 2 items, 2 violations. (2 primary, 0 associated)
  ⊟ Bit widths differ on left (32) and right (33) of assignment.        224
    221     else
    222         begin
    223             if (load)
    224                 FC_REG       <= {Enc_noise,{32-DATA_WIDTH{1'b0}}};
    225             else
    Ensure bit widths on both sides of an assignment/comparison match and be wary of arithmetic which can cause overflow.
  ⊟ Bit widths differ on left (32) and right (33) of assignment.        411
    408  begin: Zero_padding
    409      case (CTRL_REG[1:0])
    410
    411          2'b01:    DATA_IN_Pad = {{DATA_IN_REG[DATA_WIDTH-1:0]},{32-DATA_WIDTH{1'b0}}};      //Decode - need all data width since the parity is already added
    412
    Ensure bit widths on both sides of an assignment/comparison match and be wary of arithmetic which can cause overflow.
```

Figure 18: Size error.

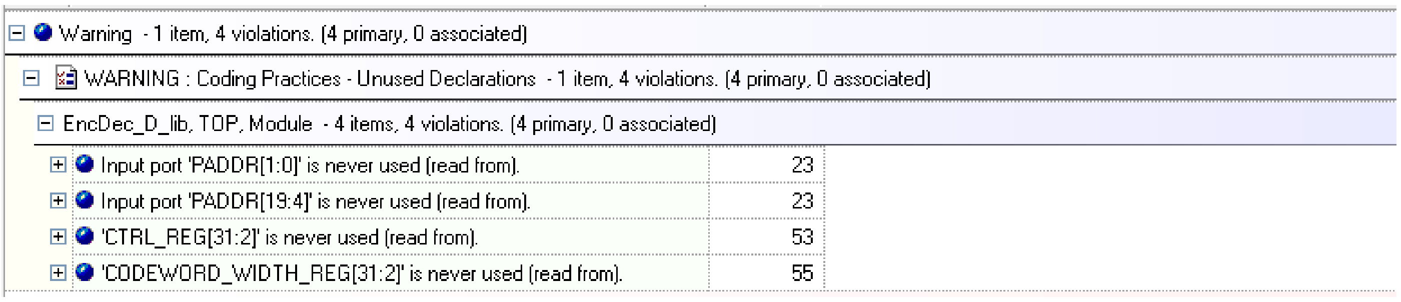
As we can see even though that say we have wrong bit amount we can se ethe even on both sides

[DATA_WIDTH-1:0] = DATA_WIDTH amount of bits

Data in pad is 32 bit

so 32 = DATA_WIDTH – (32 – DATA_WIDTH) = 32 bits

2.2 Warning



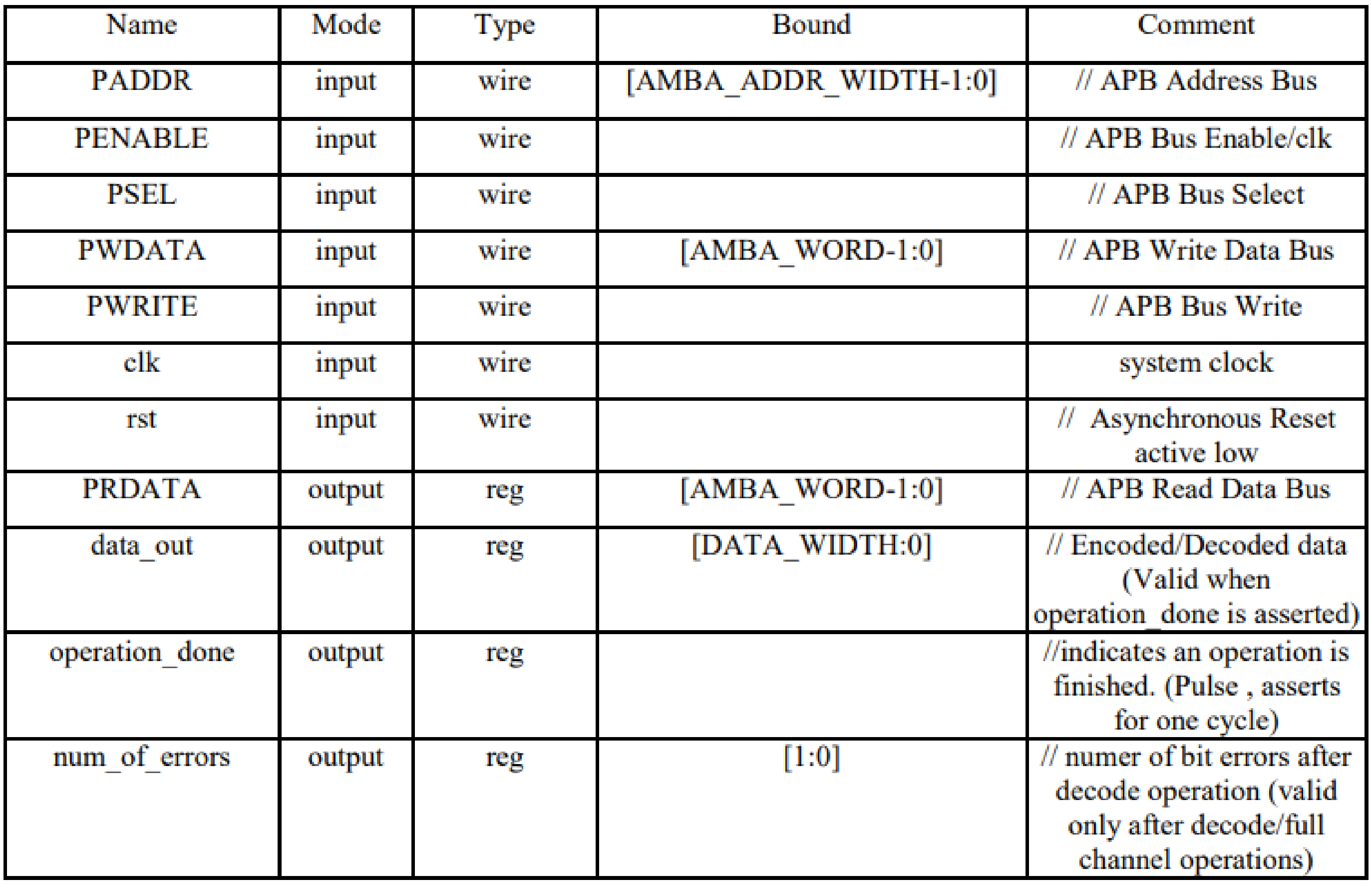
```
⊟ Warning  - 1 item, 4 violations. (4 primary, 0 associated)
  ⊟ WARNING : Coding Practices - Unused Declarations  - 1 item, 4 violations. (4 primary, 0 associated)
    ⊟ EncDec_D_lib, TOP, Module  - 4 items, 4 violations. (4 primary, 0 associated)
      ⊞ Input port 'PADDR[1:0]' is never used (read from).            23
      ⊞ Input port 'PADDR[19:4]' is never used (read from).           23
      ⊞ 'CTRL_REG[31:2]' is never used (read from).                   53
      ⊞ 'CODEWORD_WIDTH_REG[31:2]' is never used (read from).         55
```

Figure 19: warning.

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

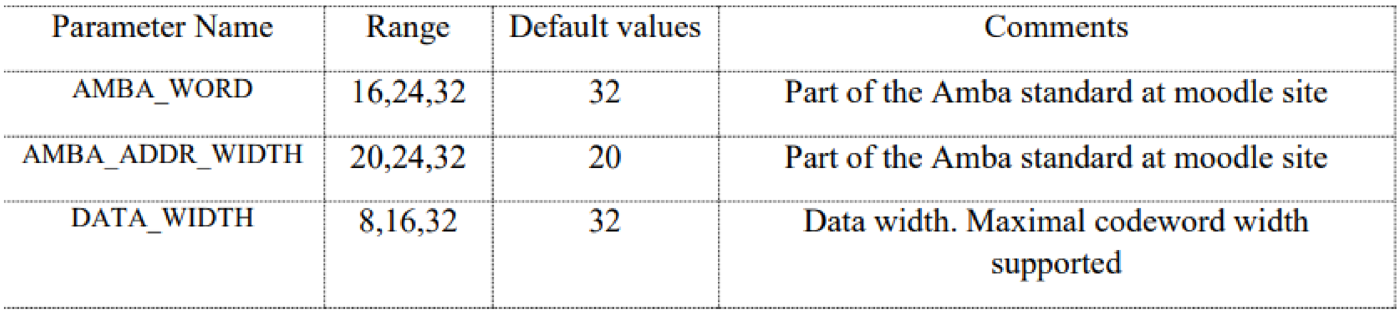
3. APPENDIX

3.1 Terminology



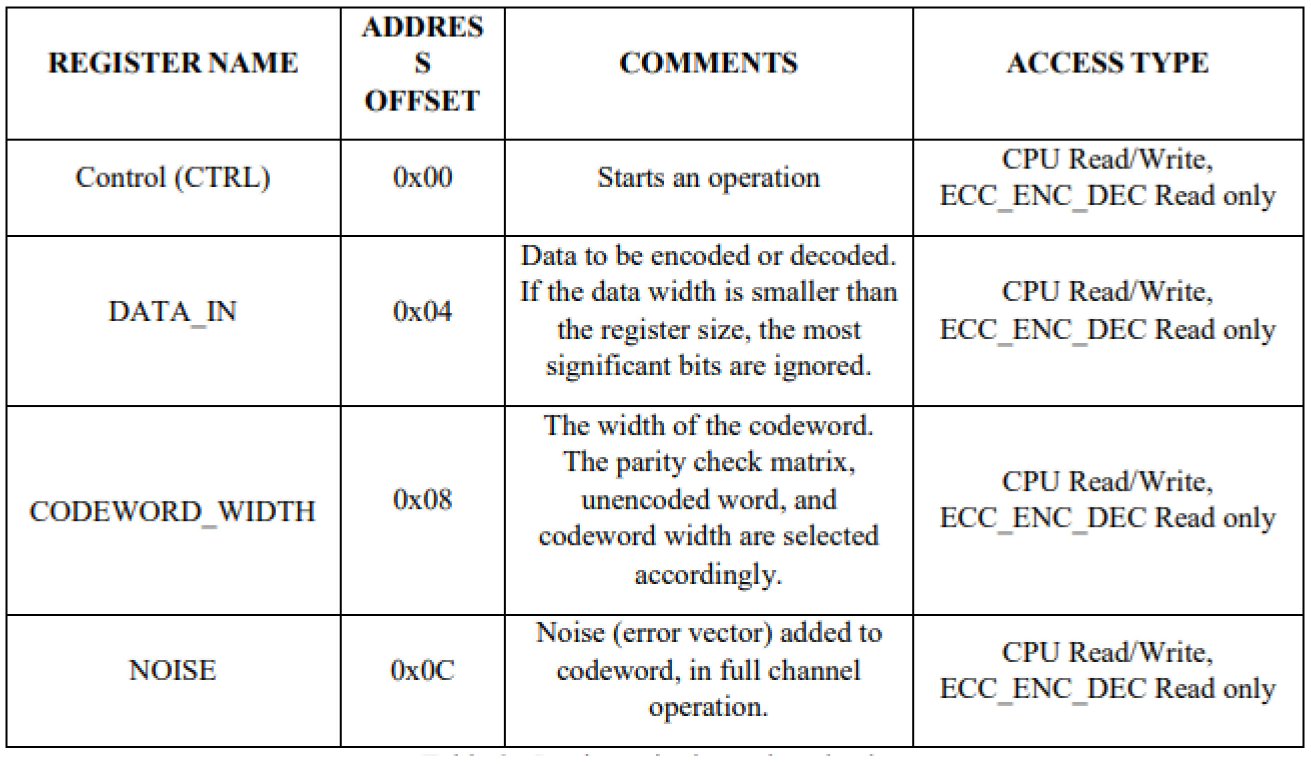
Name	Mode	Type	Bound	Comment
PADDR	input	wire	[AMBA_ADDR_WIDTH-1:0]	// APB Address Bus
PENABLE	input	wire		// APB Bus Enable/clk
PSEL	input	wire		// APB Bus Select
PWDATA	input	wire	[AMBA_WORD-1:0]	// APB Write Data Bus
PWRITE	input	wire		// APB Bus Write
clk	input	wire		system clock
rst	input	wire		// Asynchronous Reset active low
PRDATA	output	reg	[AMBA_WORD-1:0]	// APB Read Data Bus
data_out	output	reg	[DATA_WIDTH:0]	// Encoded/Decoded data (Valid when operation_done is asserted)
operation_done	output	reg		//indicates an operation is finished. (Pulse , asserts for one cycle)
num_of_errors	output	reg	[1:0]	// numer of bit errors after decode operation (valid only after decode/full channel operations)

Table 7: Assignment Interface.



Parameter Name	Range	Default values	Comments
AMBA_WORD	16,24,32	32	Part of the Amba standard at moodle site
AMBA_ADDR_WIDTH	20,24,32	20	Part of the Amba standard at moodle site
DATA_WIDTH	8,16,32	32	Data width. Maximal codeword width supported

Table 8: Parameters.

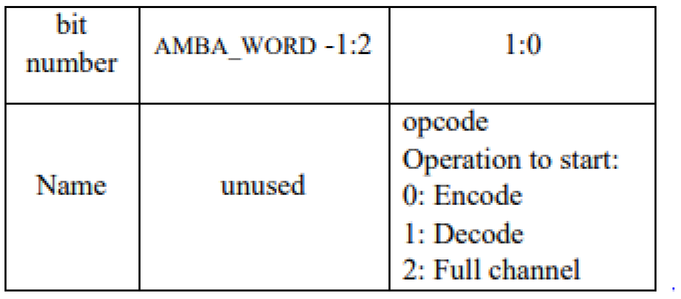


REGISTER NAME	ADDRESS OFFSET	COMMENTS	ACCESS TYPE
Control (CTRL)	0x00	Starts an operation	CPU Read/Write, ECC_ENC_DEC Read only
DATA_IN	0x04	Data to be encoded or decoded. If the data width is smaller than the register size, the most significant bits are ignored.	CPU Read/Write, ECC_ENC_DEC Read only
CODEWORD_WIDTH	0x08	The width of the codeword. The parity check matrix, unencoded word, and codeword width are selected accordingly.	CPU Read/Write, ECC_ENC_DEC Read only
NOISE	0x0C	Noise (error vector) added to codeword, in full channel operation.	CPU Read/Write, ECC_ENC_DEC Read only

Table 9: 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.

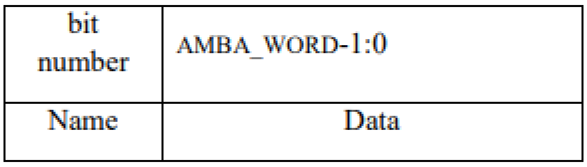


bit number	AMBA_WORD -1:2	1:0
Name	unused	opcode Operation to start: 0: Encode 1: Decode 2: Full channel

Table 10: 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

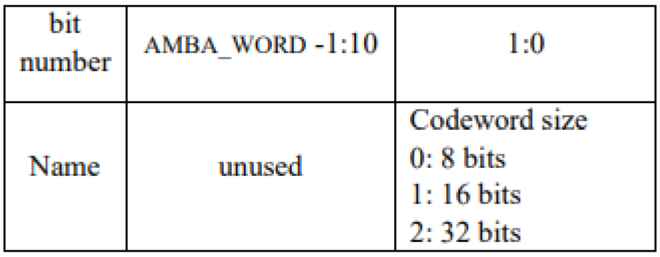


bit number	AMBA_WORD-1:0
Name	Data

Table 11: 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

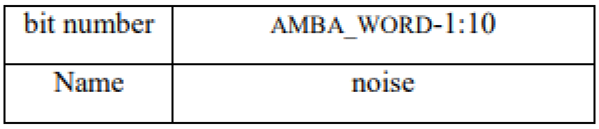


bit number	AMBA_WORD -1:10	1:0
Name	unused	Codeword size 0: 8 bits 1: 16 bits 2: 32 bits

Table 12: 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.



bit number	AMBA_WORD-1:10
Name	noise

Table 13: Noise register.

3.2 References

AMBA APB spec from the moodle

The lectures from the course