Error Correction Encoder & Decoder

Digital Design and Logical Synthesis for Electrical Computer Engineering (36113611)
Course Project

Digital High Level Design

Version 0.1

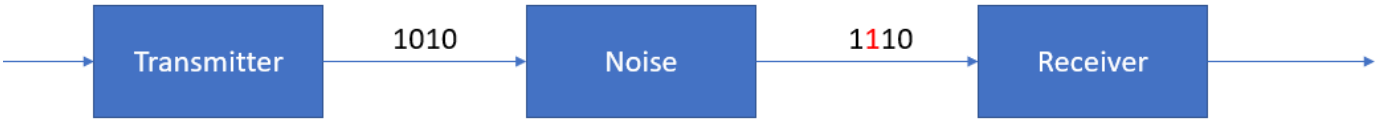
Revision History

Rev	Description	Done By	Date
0.1	Initial document	Yuval Yoskovits, Roy Shor	28.10.2021
0.2	Fixed typos in chapter 2.3 Interface	Yuval Yoskovits	3.11.2021
0.3			

Contents

1. INTRODUCTION

We want to transmit data though a noisy channel. The noise can, with some low probability, flip bits.



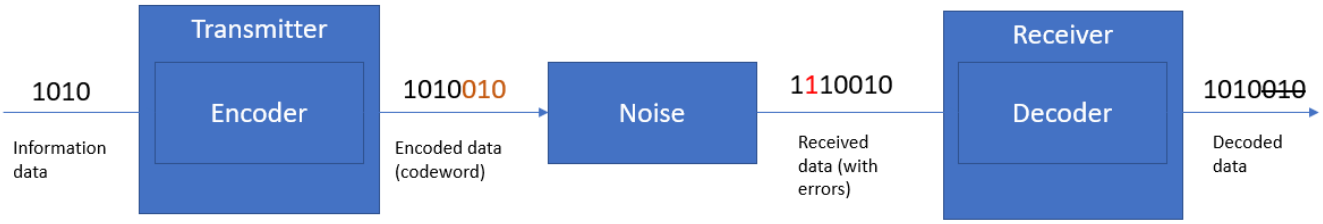
We want to protect our information data from flipping bits. To do that, we add extra bits to the data – those bits are referred to as "parity bits" (or "redundancy").

An encoder generates the parity bits to create a legal "codeword". The transmitter will transmit the codeword. The noisy channel can with some probability flip some bits (cause errors in the codeword). The receiver takes the erroneous codeword, and checks if there are errors in the data.

The decoder can:

1. Check if the received data is a legal codeword (no errors).
2. Find if there is a **single bit** error in the data and correct it.
3. Find if there are **two-bit errors** in the data, but it is not able to know which two bits are errors, and it cannot correct the data in that case.

The decoder cannot correct or detect more than two errors in the data. We assume the noisy channel cannot flip more than two bits.



Algorithm:

Codewords:

A legal codeword **c** is a row vector that applies:

$$Hc^T = 0$$

Where **H** is the "parity check matrix" of the code. **0** is the zero vector.

When multiplying the matrix with a vector, all operations are "module 2" operations, i.e. addition operation and multiplication operation are performed according to:

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

The parity check matrices for this project are defined in the appendix of this document.

Example:

$$H = (1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,0\,1\,0\,0\,1\,1\,0\,1\,0\,0\,1\,0\,1\,0\,1\,1\,0\,0\,0\,1)$$

$$y = [1\,0\,1\,0\,1\,0\,1\,0]$$

Is **y** a legal codeword?

We can see that $Hy^T = 0$ (check that!), and therefore y is a legal codeword.

Encoding:

We have a 4-bit word **w**, and we want to add parity bits to create a legal codeword **c**.

Luckily, we can see from the structure of **H**, that its right part is an upper triangular sub-matrix.

$$\boldsymbol{H} = \left(\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

Which translates the equation $Hc^T = 0$ to a simple system of equations (check that!):

$$c_8 = c_1 + c_3 + c_4$$

$$c_7 = c_1 + c_2 + c_4$$

$$c_6 = c_1 + c_2 + c_3$$

$$c_5 = c_1 + c_2 + c_3 + c_4 + c_6 + c_7 + c_8$$

Example:

w = [1 0 1 0]

And we want to complete the parity bits in **c** = [1 0 1 0 ? ? ? ?].

Then, according to the equations above:

$$c_8 = 1 + 1 + 0 = 0$$

$$c_7 = 1 + 0 + 0 = 1$$

$$c_6 = 1 + 0 + 1 = 0$$

$$c_5 = 1 + 0 + 1 + 0 + 0 + 1 + 0 = 1$$

The encoded data is c = [1 0 1 0 1 0 1 0]

In this project we will use only simple parity check matrices with similar structure.

Decoding:

We received the corrupted codeword **y** from the noisy channel.

$$y = c + e$$

Where **c** is the legal codeword that was transmitted. **e** is the "error vector", and we assume that **e** has as much as two bits that are 1. Let $s = Hy^T$

Thus,

$$s = Hy^T = H(c + e)^T = Hc^T + He^T = 0 + He^T = He^T$$

- If there are no errors, then:

$$s = 0$$

- If there is one bit error, then **s** is the column of the matrix **H**, in the same position where the error is located in **e**.

Example:

c = [1 0 1 0 1 0 1 0]

e = [0 0 1 0 0 0 0 0]

y = [1 0 0 0 1 0 1 0]

$$s = (1\ 1\ 0\ 1)$$

Which is the third column of the matrix **H**, and therefore the decoder knows that the third bit is flipped.

- If there are two bits with errors, then the is no column in the matrix **H** that equals to **s** (since **H** is built in a way that every three columns are linearly independent)

Example:

c = [1 0 1 0 1 0 1 0]

e = [0 0 1 0 0 0 0 1]

y = [1 0 0 0 1 0 1 1]

$$s = (0\ 1\ 0\ 0)$$

Is there a column in **H** that equals to that **s**?

The decoder can tell that there are two errors in the data, but cannot tell where the errors are located since there are many combinations that can create that **s**.

Hardware:

Our goal is to create a hardware accelerator that can perform both encoding and decoding.

The encoder and decoder will be located on the same module (one top level module) and may share some resources.

Our design is a peripheral part of an ARM Processor and uses APB bus protocols to communicate with it (Figure 2). The CPU can read and write a register bank inside the design (marked in red).

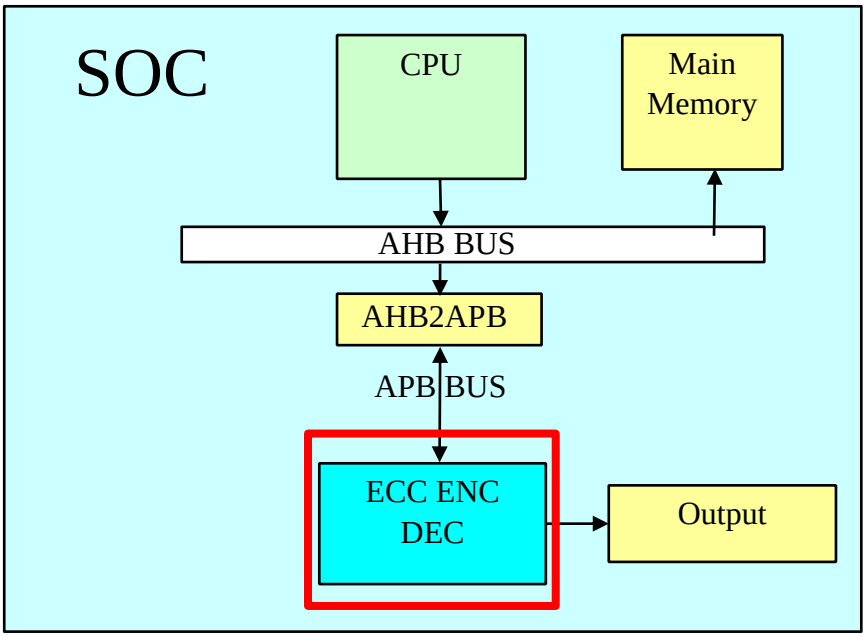


Figure 2: Top view of the SOC environment around the design

The design, named ECC_ENC_DEC (Error Correction Coding Encoder & Decoder), is controlled by the CPU which configures the design via APB bus and a register bank inside the design. CPU can write to the register bank and read its content.

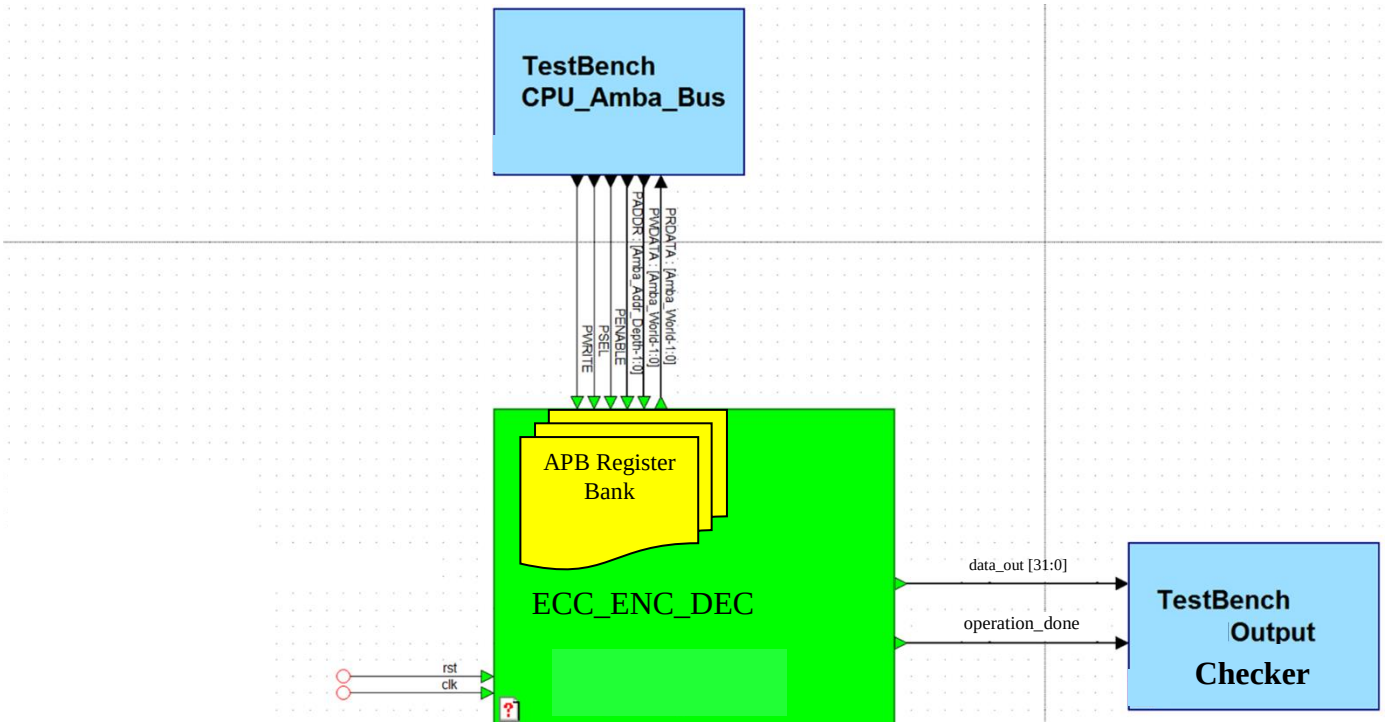


Figure 3: Top view of the test environment around the design

2. SPECIFICATION

2.1 Functional Description

The design is a hardware accelerator that is controlled by a firmware program. The CPU translates the firmware into registers read/write operations, which are translated into APB transactions. The ECC_ENC_DEC receives requests to read or write to the register from the APB Bus. The APB Slave (implemented inside the ECC_ENC_DEC module) translates the APB transactions into registers operations.

Please refer to the AMBA APB Protocol specification sheet (on moodle).

2.2 Operations

The design supports three operations

2.2.1 Encode

In the encode operation, the input data (DATA_IN) contains the unencoded word (information data). The module will encode it and put the codeword result on the output.

2.2.2 Decode

In the decode operation, the input data (DATA_IN) contains a codeword with errors. The module will decode the data, and notify the number of errors via the output. If there are less than two errors, the module will also

put the decoded (fixed) word on the output and discard the parity. If there are two errors in the data, the module only notifies that there are two errors in the data, and the output data is not valid in that case.

2.2.3 Full channel

In the full channel operation, we simulate the entire process of encoding and decoding data. The module will read an unencoded word from the DATA_IN register. Then, it will encode it to create a legal codeword. Then it will add noise to the data (xor with the value that is configured in the NOISE register). Then, it will decode the data. The output is the number of errors that were detected. If there are less than two errors it will also output the decoded data (similar to decode operation).

2.3 Interface

.

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 1 : Block interface.

2.4 Parameters

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 2 : Parameters.

2.5 Register block and programming model:

The controller has a set of several registers at size Amba_Word bit registers. The registers are accessible via the APB interface. You should implement this interface according to the AMBA standard (APB section).

The registers are:

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 3 : Registers in the register bank.

2.5.1 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

2.5.2 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

2.5.3 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

2.5.4 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

2.6 Design Objectives

Set **a priority and** plan your design accordingly:

- € Smallest power consumption and smallest design area.
- € Best performance.

Explain in details how the tradeoff affected your design!

2.7 Project Assumptions, Constraints and Work flow

2.7.1 Constraints

1) Write/read process only when APB protocol is correct, otherwise unexpected results may happen. ECC_ENC_DEC starts operation when CPU writes to the control register.

2) CPU and design can read the contents of the register bank simultaneously.

3) For each register, in the bank, only CPU has write permission.

4) CPU does not change the registers values while the ECC_ENC_DEC is busy performing any operation (i.e. between CPU writes to the control register and until the operation_done goes to '1').

2.7.2 Workflow

1) CPU writes input data and other required configuration values to the register bank via APB bus.

2) CPU writes a legal opcode value to the control register.

3) The ECC_ENC_DEC performs the requested operation

4) When operation is finished, the operation_done signal is asserted for one cycle, together with the output data, and number of bit errors that were detected (in case of decode/full channel operations)

5) Only after an operation is done, the CPU may change the configuration registers or start a new operation (assume the CPU is informed of when the operation_done goes to '1').

3. SUBMISSION REQUIREMENTS

For the first (design) part of the project you have to submit the following:

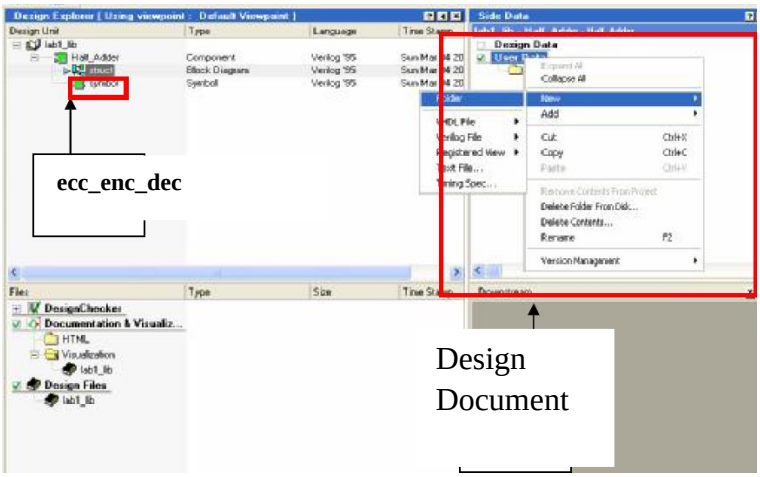
1) HDL Designer library including project file, hdl and hds libraries. Top level module should be named **ecc_enc_dec**, and the file should be named **ecc_enc_dec.v**. Verilog 2005 syntax

2) **Short Design Guide** (about 5-10 pages) containing the following (in pdf format):

   - Top level Block Diagram
   - Functionality description of your design.
   - Flow chart\State machine diagrams.
   - Explain which rules you waved in design checker.

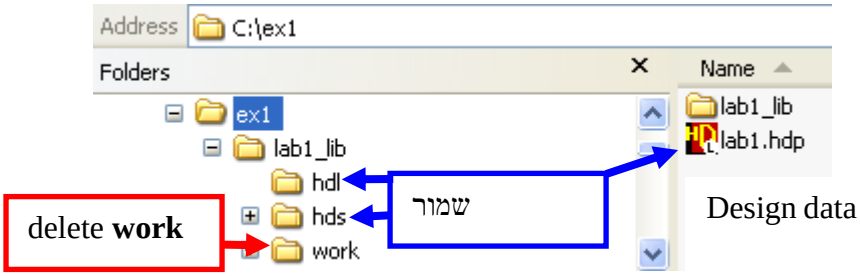
Use the template provided in the moodle□מעבדות□part1 template: **Digital Block_02_Definition_Template.docx**

1) Document should be attached in the **ecc_enc_dec** design in HDL designer using side data library



2) **Delete** the **work** library

3) Compress **Project file**, **hds** and **hdl** libraries into a ZIP file named ***<ID1>_<ID2>.zip***, where ID1 and ID2 are the ID (*teudat zeut*) numbers of the submitters.



4) All the library files should be zipped to a file named **id1_id2.zip** and submitted to moodle.

5) Both Students Should Submit their work to moodle.

SUBMISSION DATE: 25/11 AT 23:59

4. PROJECT GRADE EVALUATION

This part is **30 points** **from the project grade.** Preliminary checks must pass before your work is checked. **Failure in the preliminary checks means grade of 0 for this part of the project.**

Task	Description
Code compiles	Code compiles without compilation error in QuestaSim and Design Compiler

Table 4 : Perliminary tests

After passing the preliminary tests the system's performance will be checked with additional tests.

grade	Task	Description

40%	RTL code quality	**Grade given by Design Checker on the customized course policy (DO-254 +RMM).** All rules considered except the ones use chose to exclude and explained them well in your document.
30%	Document	Block diagrams, flow diagram, FSM machine diagrams **easy to read and understand your design** (use **Digital Block_02_Definition_Template**).
30%	Reuse	Design works and compiles under all the ranges of the parameters from Table 2.

Table 5 : Grading policy

Late submission will degrade the project score by 5*days

5. APPENDIX

5.1 Terminology

5.2 References

[1] Amba standard moodle Amba Specifications

5.3 Codes

The following table describes the number of information bits, parity bits, and codeword width for each code, and their related matrix:

#	information	parity	codeword	matrix
1	4	4	8	H_1
2	11	5	16	H_2
3	26	6	32	H_3

H_1

1	1	1	1	1	1	1	1
1	1	1	0	0	1	0	0
1	1	0	1	0	0	1	0
1	0	1	1	0	0	0	1

H_2

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0
1	1	1	1	0	0	0	1	1	1	0	0	0	1	0	0
1	1	0	0	1	1	0	1	1	0	1	0	0	0	1	0
1	0	1	0	1	0	1	1	0	1	1	0	0	0	0	1

H_3

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
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	1	0	0	0	0
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	1	0	0	0
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	1	0	0
1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1	1	0	0	1	1	0	1	1	0	1	0	0	0	0	1	0
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	1