Error Correction Encoder & Decoder

Digital Design and Logical Synthesis for Electrical Computer Engineering

Digital High Level Design

Version 0.1

By: Aharon lugassi

Saar Avraham

Revision Log

Rev	Change	Description	Reason for change	Done By	Date
0.1	Initial document				
0.2					
0.3					

Table of Content

TABLES AND FIGURES

1.1 LIST OF FIGURES:

1.2 LIST OF TABLES

1. BLOCKS FUNCTIONAL DESCRIPTIONS

In this part we will describe the functioning components/modules of our project. For ease of understanding, we will list the modules in the order which we built them, working by down-top method.

2.1 Encoder

This module objective is to receive a word (string of 1/0 bits) as input, calculate the appropriate parity bits needed to be added, concatenate them to the right of the code word, and send out the coded word as output. The calculation of the parity bits is done by matrix multiplication as described in the project assignment. These multiplications can be easily translated to linear equations, allowing us to calculate each of the parity bits by adding together the appropriate bits from the input word. Some parity bits are calculated based on other parity bits, and so we chose to use blocking assignments for calculation. A temporary signal receive the correct bits, and then appointed to the output in the synchronous part of the module. Other input signals used are codeword_width, used to decide how many parity bits needs to be assigned, and an enable signal to be used for changing the output in the synchronous part. The encoder is a sub module which inputs and outputs are connected to the Controller.

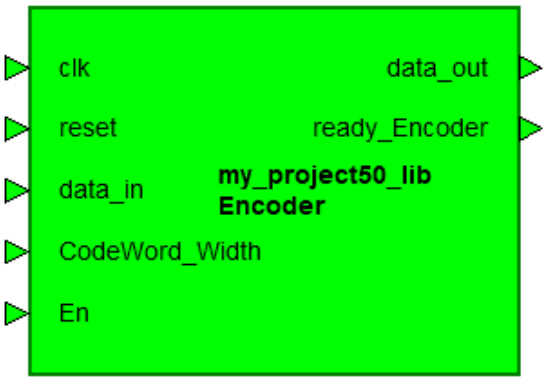


Figure 1: view of Encoder block.

	Name	Mode	Type	Bounds	Comments
1	clk	input	wire		system clock
2	reset	input	wire		reset
3	data_in	input	wire	[DATA_WIDTH-1:0]	Data to be encoded
4	CodeWord_Width	input	wire	[AMBA_WORD-1:0]	length of word
5	En	input	wire		enable signal
6	data_out	output	reg	[DATA_WIDTH-1:0]	coded word output
7	ready_Encoder	output	reg		Encoder finished indicator

Table 1: Block interface of Encoder.

2.2 Decoder

This module purpose is to receive a coded word, and output a correct word, assuming no more than 1 bit was flipped before the word entered this module. If 2 bits were flipped, the module will know to recognize this and alert so in the output. After receiving codeword_width, the module works in these two stages, following logic from the project assignment: 1- Setting bits of signal ‘s’ based on the linear equations derived from the matrix multiplication. 2- Comparing the vector to the columns of matrix H and acting by it. If ‘s’ match one of the columns, we know the number of the column is the number of the flipped bit, and so we assign to a temporary signal the input codeword without the parity bits, but with the correct bit flipped, and also assign 1 to a signal indicating number of errors. If ‘s’ is 0, we understand no errors were made and output the original input signals to the output. And if ‘s’ matches no column of H, we understand 2 or more mistakes were made, and output that in the number of errors signal. In the synchronous part we assign the temporary signals to the output, pending enable signal is turned on. The Decoder is also a sub module which inputs and outputs are connected to the Controller.

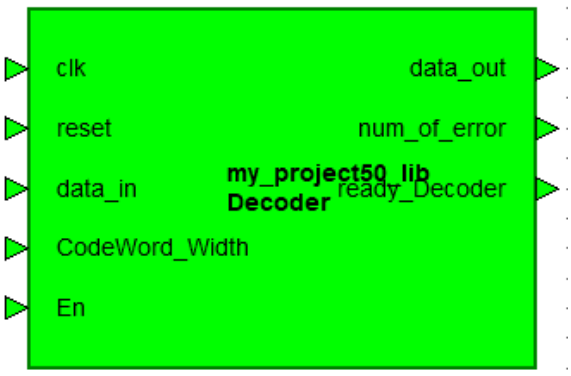


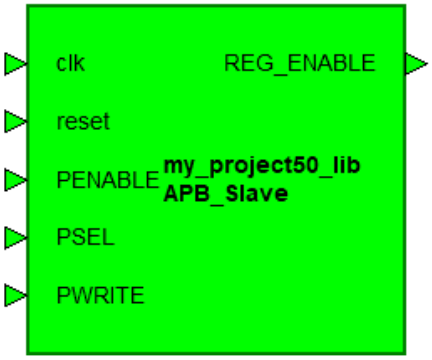
Figure 2: view of Decoder block

	Name	Mode	Type	Bounds	Comments
1	clk	input	wire		system clock
2	reset	input	wire		reset
3	data_in	input	wire	[DATA_WIDTH-1:0]	data to be decoded
4	CodeWord_Width	input	wire	[AMBA_WORD-1:0]	length of word
5	En	input	wire		enable signal
6	data_out	output	reg	[DATA_WIDTH-1:0]	decoded word output
7	num_of_error	output	reg	[1:0]	number of errors
8	ready_Decoder	output	reg		Decoder finished indicator

Table 2: Block interface of Decoder

2.3 APB_Slave

This module ensures the writing and reading between the CPU and the ecc_enc_dec is done by the correct APB protocol, as described in the project assignment. This module implement an FSM machine, acting by the inputs of PENABLE, PSEL and PWRITE, to switch states of the FSM maching. The output of the module i s the signal REG_ENABLE, which enables writing and reading in the Register_Bank module. As guided in the APB protocol, the FSM machine have 3 states: IDLE, SETUP, and ACCESS. We stay in IDLE until PSEL goes to 1, meaning we intend to use the ecc_enc_dec peripheral module, and go to state SETUP. From there we wait upon PENABLE to equal 1, and then go to ACCESS. Only once in ACCESS, we apply a value to the REG_ENABLE, indicating all parameters are stable and it is clear to write or read to/from the registers.



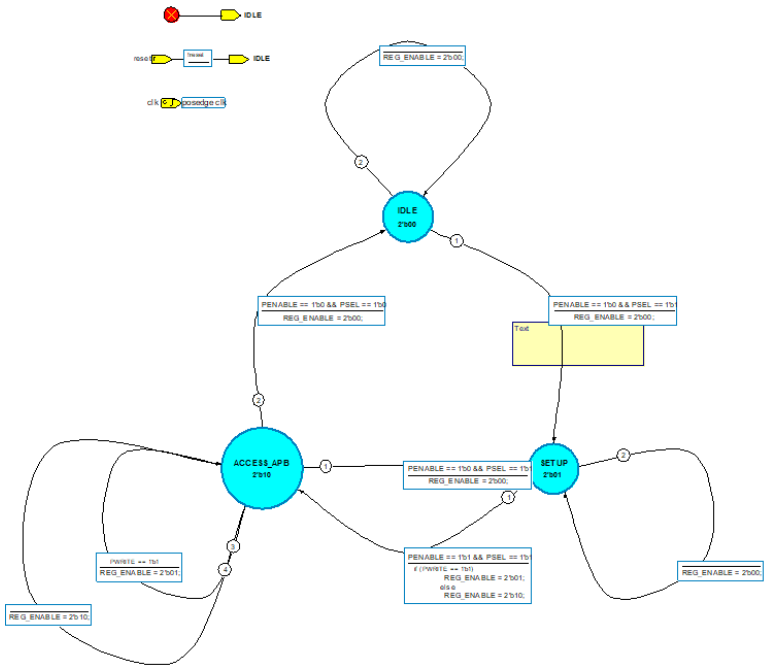


Figure 3: view of APB_Slave block and FSM machine

	Name	Mode	Type	Bounds	Comments
1	clk	input	wire		system clock
2	reset	input	wire		reset
3	PENABLE	input	wire		APB enable
4	PSEL	input	wire		APB select
5	PWRITE	input	wire		APB write/read
6	REG_ENABLE	output	reg	[1:0]	Register enabling signal

Table 3: Block interface of APB_Slave

2.4 Register_Bank

This module represent the registers CRTL, DATA_IN, CODEWORD_WIDTH and NOISE as requested in the project assignment. We implement those as reg outputs, and use PADDR and PWDATA as inputs. This module implement a basic synchronous assignment, using REG_ENABLE value to decide if we want to read or write, and a case statement using PADDR to decide which register to interact with. If REG_ENABLE holds the value which correlate to 'read', we use the PADDR case statement to decide which register value we transfer to the output PRDATA. Last output we apply in this module is CRTL_ready, which we use as indicator that CTRL register has received a writing command, which is the catalyst for using the values we assigned to the registers and calculate one of the operations required

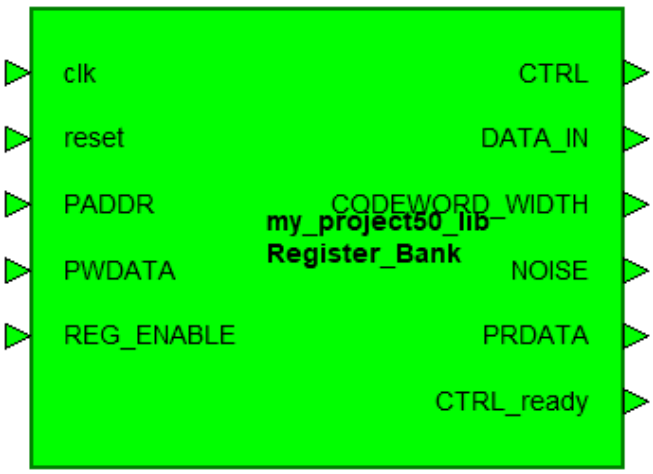


Figure 4: view of Register Bank block and FSM machine

	Name	Mode	Type	Bounds	Comments
1	clk	input	wire		system clock
2	reset	input	wire		reset
3	PADDR	input	wire	[AMBA_ADDR_WIDTH-1:0]	APB register address
4	PWDATA	input	wire	[AMBA_WORD-1:0]	APB data to write
5	REG_ENABLE	input	wire	[1:0]	signal to enable write/read to registers
6	CTRL	output	reg	[AMBA_WORD-1:0]	CTRL register
7	DATA_IN	output	reg	[AMBA_WORD-1:0]	DATA_IN register
8	CODEWORD_WIDTH	output	reg	[AMBA_WORD-1:0]	CODEWORD_WIDTH register
9	NOISE	output	reg	[AMBA_WORD-1:0]	NOISE register
10	PRDATA	output	reg	[AMBA_WORD-1:0]	APB data read to CPU signal
11	CTRL_ready	output	reg		Signal to begin operations

Table 4: Block interface of Register Bank

2.5 Controller

This is the main module in our project, and is the one who connect and use the Encoder and Decoder we built previously. We use the registers from the Register_bank as inputs, which holds all the values we need to encode/decode, and the CTRL_Ready input as the jump signal to start the operation. We use an FSM machine for each of the operations, to avoid latches and errors. For encode and decode operation we have 2 states for each, and for full operation we have 3. This allows us to have a 1 cycle delay between the encode and decode actions, and ensure a correct output of full operation mode. We connect all the inputs of the encoder and decoder to the correct register signals so they would get the correct input word, word length and enable signal, and start the FSM machine once we get 1 on CRTL_ready signal, which means the CRTL register has been written to and an operation need to take action. We choose what FSM to enable base on the value that is held in the CTRL register, which is also available to us.

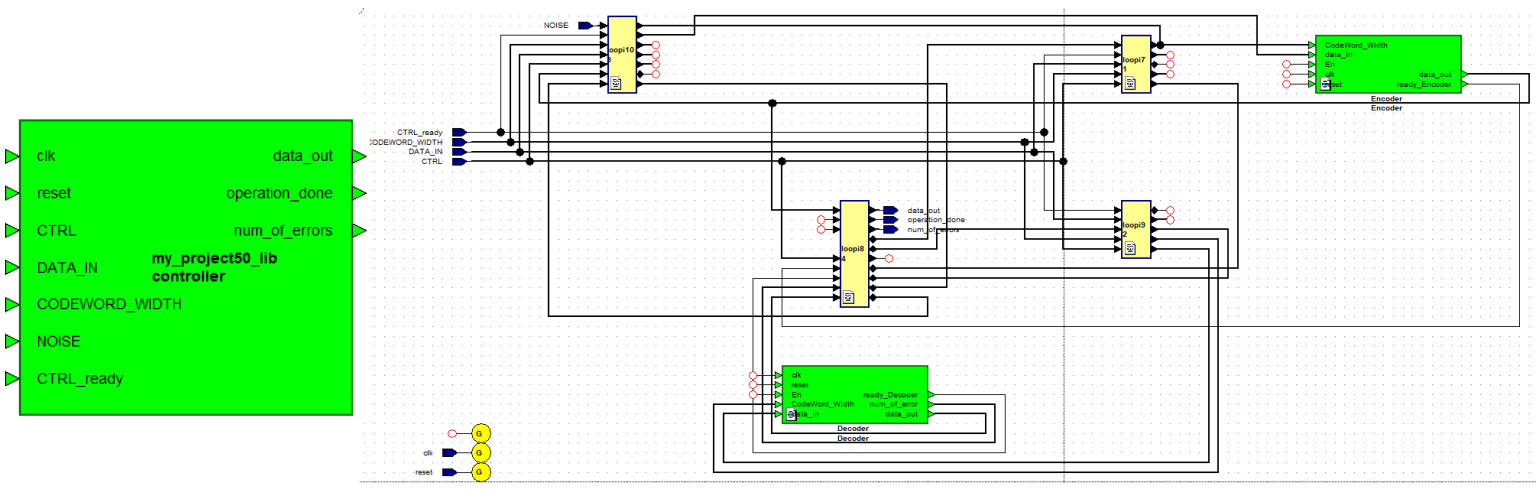


Figure 5: view of Controller block and flow chart

	Name	Mode	Type	Bounds	Comments
1	data_in_enc	local	reg	[DATA_WIDTH-1:0]	data input to encoder
2	CodeWord_Width_enc	local	reg	[AMBA_WORD-1:0]	codeword length
3	data_out_enc	local	wire	[DATA_WIDTH-1:0]	data output of encoder
4	ready_Encoder_enc	local	wire		encoder finished indicator
5	data_in_dec	local	reg	[DATA_WIDTH-1:0]	data input to decoder
6	CodeWord_Width_dec	local	reg	[AMBA_WORD-1:0]	codeword length
7	data_out_dec	local	wire	[DATA_WIDTH-1:0]	data output of decoder
8	num_of_error_dec	local	wire	[1:0]	number of errors found in codeword
9	ready_Decoder_dec	local	wire		decoder finished indicator
10	En_Enc_Dec	local	reg	[1:0]	encoder/decoder enable vector
11	clk	input	wire		system clock
12	reset	input	wire		reset
13	CTRL	input	wire	[AMBA_WORD-1:0]	CTRL register
14	DATA_IN	input	wire	[AMBA_WORD-1:0]	DATA_IN register
15	CODEWORD_WIDTH	input	wire	[AMBA_WORD-1:0]	CODEWORD _WIDTH register
16	NOISE	input	wire	[AMBA_WORD-1:0]	NOISE register
17	CTRL_ready	input	wire		Operetion begin signal
18	data_out	output	reg	[DATA_WIDTH-1:0]	Data output of ecc_enc_dec
19	operation_done	output	reg		operation_done of ecc_enc_dec
20	num_of_errors	output	reg	[1:0]	number of errors of ecc_enc_dec

Table 5: Block interface of Controller

2.6 ecc_enc_dec

This is our top module. It doesn't do anything by itself, but connects all the other modules in the appropriate way. Its inputs and outputs are exactly as described in the project assignment. Once the input changes the modules react automatically.

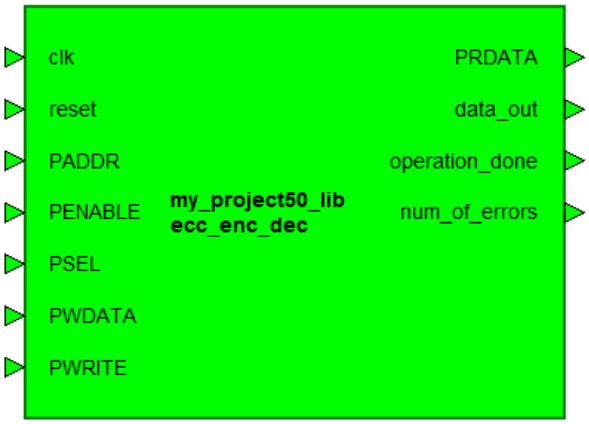


Figure 6: view of ecc_enc_dec block and flow chart

	Name	Mode	Type	Bounds	Comments
1	REG_ENABLE_APB_Slave	local	wire	[1:0]	APB_Slave to register bank
2	CTRL_Register_Bank	local	wire	[AMBA_WORD-1:0]	register bank to controller
3	DATA_IN_Register_Bank	local	wire	[AMBA_WORD-1:0]	register bank to controller
4	CODEWORD_WIDTH_Register_Bank	local	wire	[AMBA_WORD-1:0]	register bank to controller
5	NOISE_Register_Bank	local	wire	[AMBA_WORD-1:0]	register bank to controller
6	CTRL_ready_Register_Bank	local	wire		register bank to controller
7	clk	input	wire		system clock
8	reset	input	wire		reset
9	PADDR	input	wire	[AMBA_ADDR_WIDTH-1:0]	APB register adress
10	PENABLE	input	wire		APB enable
11	PSEL	input	wire		APB select
12	PWDATA	input	wire	[AMBA_WORD-1:0]	APB data to write
13	PWRITE	input	wire		APB write/read
14	PRDATA	output	wire	[AMBA_WORD-1:0]	APB date read to CPU
15	data_out	output	wire	[DATA_WIDTH-1:0]	ecc_enc_dec output
16	operation_done	output	wire		operation_done output
17	num_of_errors	output	wire	[1:0]	number of errors output

Table 6: Block interface of ecc_enc_dec

3 APPENDIX

3.1Terminology

LSB - Least Significant Bit

TBR - To Be Reviewed

TBD - To Be Defined

3.2References