4-bit ALSU with 40 Operations in Modular Design



Ahmed Hisham Abdelfattah ECE Department

Table Of Contents:

I.Introduction

2. Design Architecture.

- 2.1 Design Overview of ALSU.
- 2.2 Arithmetic& Arithmetical Shift Unit Architecture.
- 2.3 Logic& Logical Shift Unit Architecture.

3.ALSU Operations – Selectors Table.

4.Detailed Design Architecture for Arithmetic& Arithmetical Shift Unit.

- 4.1 Adder Top Module Implementation & Testing.
- 4.2 Multiplier Top Module Implementation & Testing.
- 4.3 Divider Top Module Implementation & Testing.
- 4.4 Parity Checker Top Module Implementation & Testing.
- 4.5 Arithmetical Shifter Top Module Implementation & Testing.
- 4.6 Reverser Top Module Implementation & Testing.
- 4.7 Incrementer Top Module Implementation & Testing.
- 4.8 Decrementer Top Module Implementation & Testing.
- 4.9 Arithmetic& Arithmetical Shift Unit Top Module Schematic.

5. Detailed Design Architecture for Logic & Logical Shift Unit.

- 5.1 And, OR, XOR, XNOR Operations Block Top Module Implementation & Testing.
- 5.2 NAND, NOR, Complement Ops. Block Top Module Implementation & Testing.
- 5.3 Bypass, Equality, SLT Operations Block Top Module Implementation & Testing.
- 5.4 Logical Shift & Rotate Operations Block Top Module Implementation & Testing.
- 5.5 Logic& Logical Shift Unit Top Module Schematic.

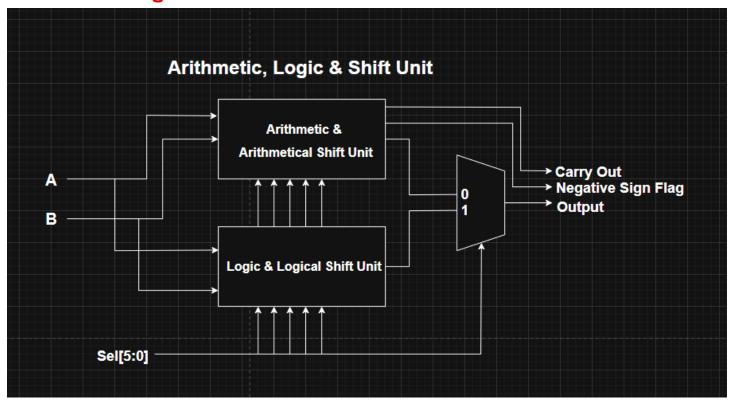
6. Simulation & Testing of ALSU Top Module.

1.Introduction:

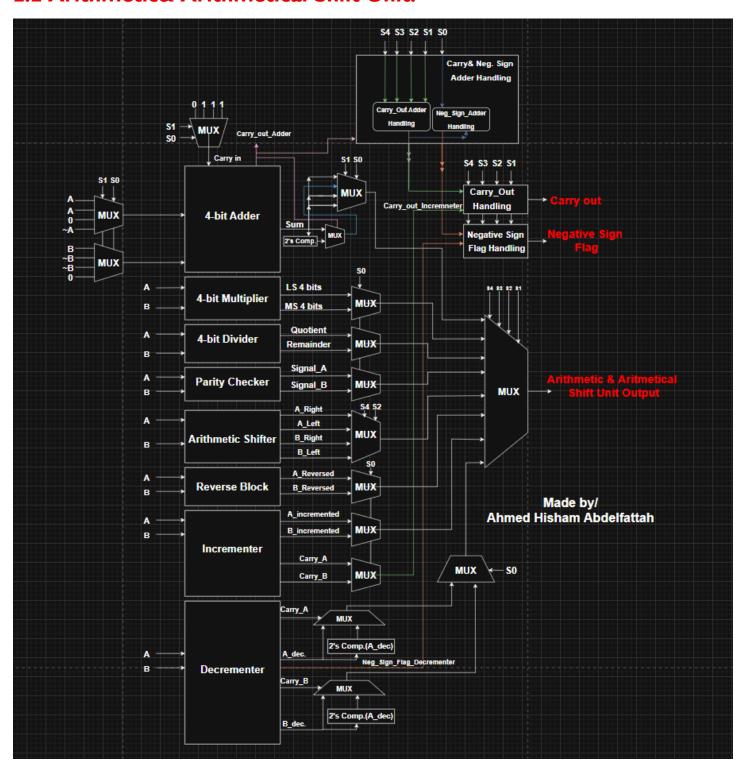
- ➤ This Document Presents the Design and Implementation of 4-bit ALSU with 40 Operations in Modular Design in Verilog HDL
- ➤ This Project is made in modular Design as I made the Architecture of Arithmetic & Arithmetical Shift Unit and the Architecture of Logical and Logic Shift unit and All the Blocks in the system.
- And this Project includes also like 4-bit Divider and its Architecture, 4-bit Multiplier, Incrementer, Decrementer taking in consideration the Negative Sign in Operations Like Subtractor and Decrementer by making an output (Negative Sign Flag).
- > This Project also Contains Logic Operations like AND, OR, XOR, XNOR and a lot of other logical operations.
- ➤ This project is mainly designed in around 3000 lines of Code using Verilog HDL as I made a design and Test Bench for each module individually and for the whole system (you Can See them from my Repo.)
- > Repo: Repo ALSU 40-Operations
- > LinkedIn Account: LinkedIn Ahmed Hisham Abdelfattah
- For any Questions, feel free to reach out at: ahmed.hesham2005.bu@gmail.com

2.Design Architecture:

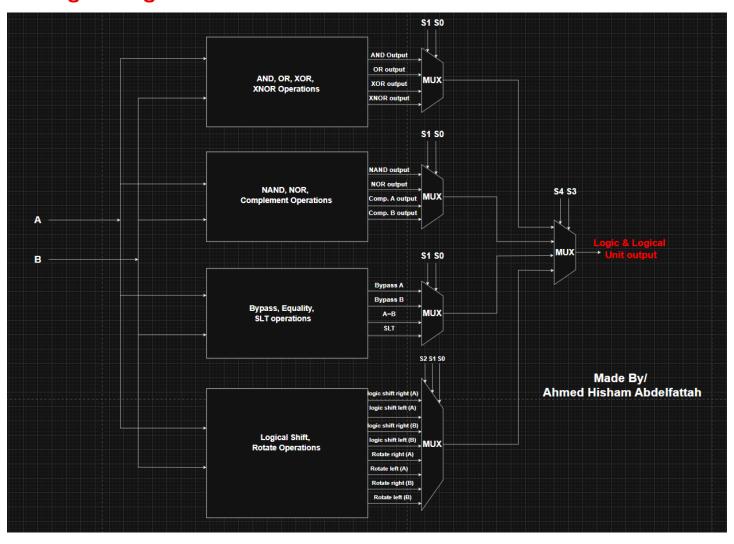
2.1 ALSU Design Overview:



2.2 Arithmetic& Arithmetical Shift Unit:



2.3 Logic& Logical Shift Unit:



3.ALSU Operations – Selectors Table:

3.1 Logic& Logical Shift Unit:

S5	S4	S3	S2	SI	S0	Function
I	0	0	0	0	0	AND
I	0	0	0	0	I	OR
I	0	0	0	I	0	XOR
I	0	0	0	I	I	XNOR
I	0	I	1	0	0	NAND
I	0	I	I	0	I	NOR
I	0	I	I	I	0	Complement (A)
I	0	I	I	I	I	Complement (B)
I	1	0	0	0	0	Bypass (A)
I	I	0	0	0	I	Bypass (B)
I	I	0	0	I	0	Equality (A = B)
I	I	0	0	I	I	SLT (A < B)
I	I	I	0	0	0	Logic Shift Right (A)
I	I	1	0	0	I	Logic Shift Left (A)
I	I	1	0	I	0	Logic Shift Right (B)
I	I	I	0	I	I	Logic Shift Left (B)
I	I	I	I	0	0	Rotate Right (A)
I	I	1	I	0	I	Rotate Left (A)
I	I	1	I	I	0	Rotate Right (B)
I	I	1	I	I	I	Rotate Left (B)

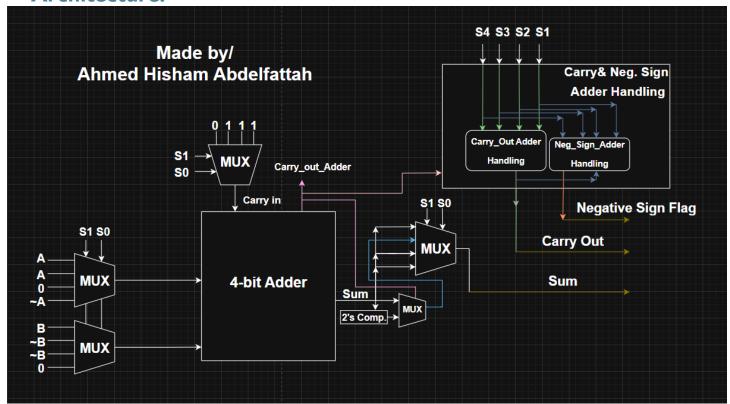
3.2 Arithmetic& Arithmetical Shift Unit:

S5	S4	S3	S2	SI	SO	Function
0	0	0	0	0	0	A + B
0	0	0	0	0	1	A - B
0	0	0	0	I	0	2's Comp. (B)
0	0	0	0	I	1	2's Comp. (A)
0	0	0	I	0	0	A × B (least significant 4 bits)
0	0	0	I	0	I	A × B (most significant 4 bits)
0	0	I	I	0	0	A / B (Quotient)
0	0	I	I	0	I	A / B (Remainder)
0	I	0	1	0	0	Parity Checker (A)
0	I	0	1	0	I	Parity Checker (B)
0	0	I	0	I	0	Arithmetic Shift Right (A)
0	0	I	I	I	I	Arithmetic Shift Left (A)
0	I	I	0	I	I	Arithmetic Shift Right (B)
0	I	I	I	I	1	Arithmetic Shift Left (B)
0	I	0	0	0	0	Reverse (A)
0	I	0	0	0	1	Reverse (B)
0	1	1	I	0	0	Increment (A)
0	1	I	I	0	I	Increment (B)
0	I	I	0	0	0	Decrement (A)
0	I	1	0	0	I	Decrement (B)

4. Detailed Architecture Description for Arithmetic& Arithmetical Shift Unit:

4.1 Adder Top Module:

• Architecture:



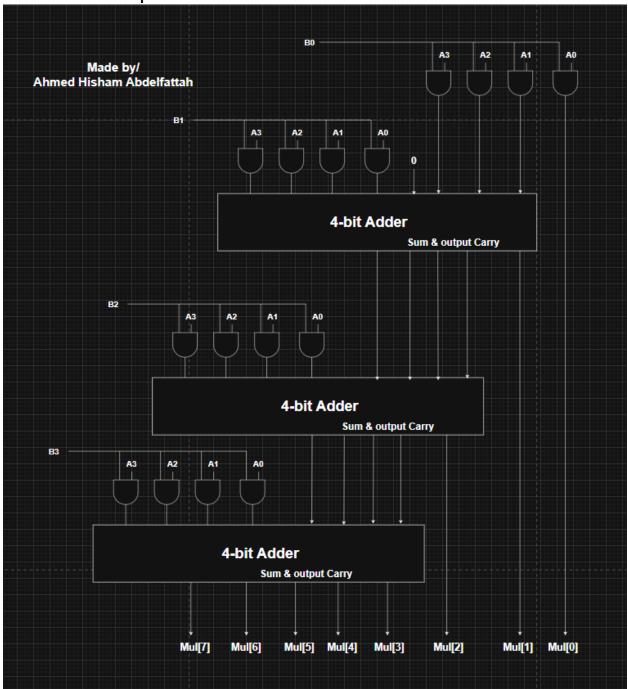
This Module works according to the Selectors
 If Sel = 00 → (A+B), Sel = 01 → (A-B), Sel = 10 → (2's Comp.(B))
 , If Sel = 10 → (2's Comp.(A))

• Test Bench Simulation:

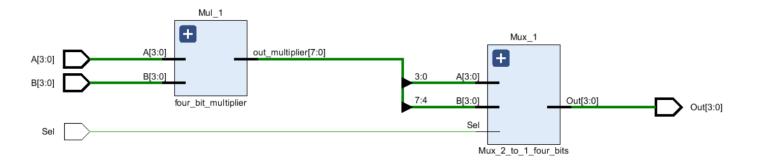


4.2 Multiplier Top Module:

4.2.1 4-bit Multiplier Block:



4.2.2 Multiplier Top Module Implementation:

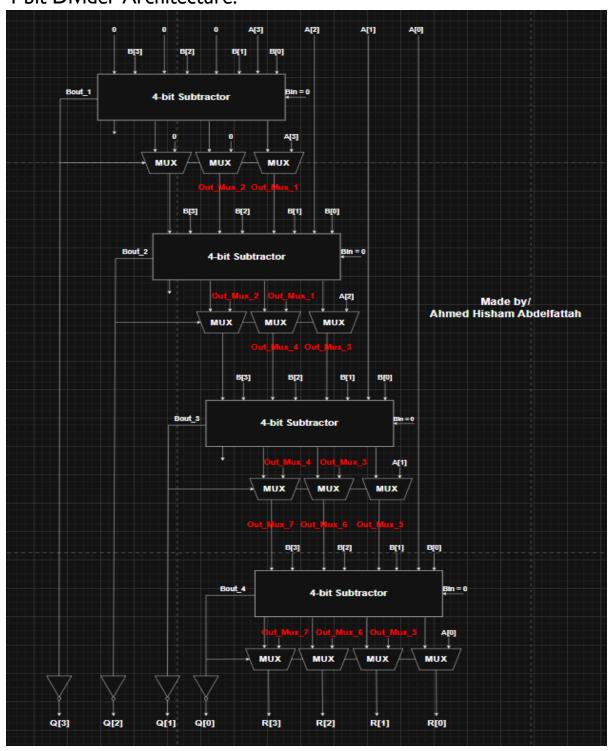


- This Module works according to the Selector:
 If Sel = 0 → Out = Least Significant 4 bits and If Sel = I → Out = Most Significant 4 bits.
- Test Bench Simulation:

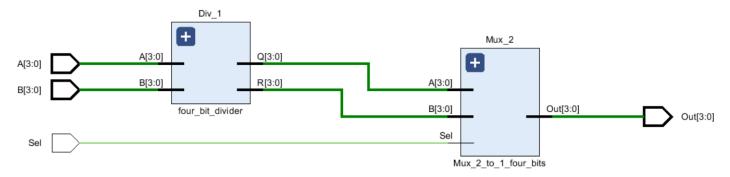
\$ 1 →	Msgs										
■- 〈 A ■- 〈 B	4'd0	0	5		0	15	3	10	12	15	0
 → B	4'd0	0	4	3	0	3	5	1	13	15	1
+ -◆ Out	8'd0	0	20	15	0	45	15	10	156	225	0
- → error_count	32'd0	0									

4.3 Divider Top Module:

4.3.1 4-bit Divider Architecture:



4.3.2 Divider Top Module Implementation:



• When Sel = 0, the output will be the Quotient and when Sel = 1, the output will be the Remainder.

4.3.3 Test Bench Simulation:

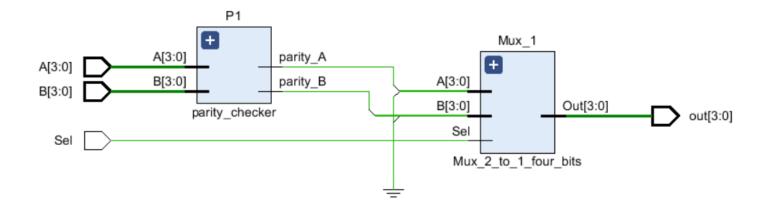
\$ 1 →	Msgs										
⊕ - ♦ A	4'd10	5		0	15	3	10	12	15	10	
♦ B	4'd5	4	3	1	3	5	1	13	15	5	
⊞- Q	4'd2	1	1	0	5	0	10	0	1	2	
-	4'd0	1	2	0	0	3	0	12	0	0	
error_count	32'd0	0									

4.4 Parity Checker Top Module:

4.4.1 4-bit Parity Checker:



4.4.2 Parity Checker Top Module Implementation:



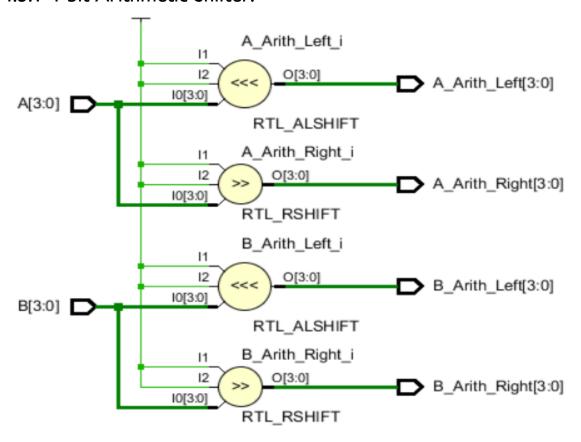
This Module when Sel = 0, the output will equal Parity_Checker_A
 And when Sel = 1, the output will equal Parity checker B

4.4.3 Test Bench Simulation:

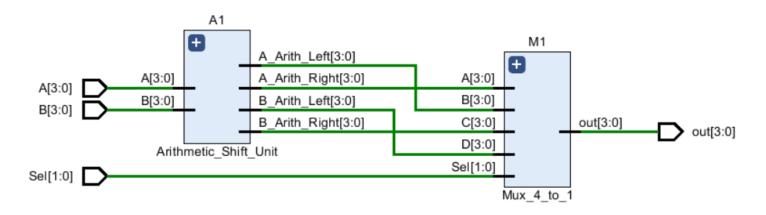
Msgs																
4'b0000	0000 (0001	0010 (0011	0100 0101	0110 0111	1000 (1001	1010 (1011	1100 (1101	1110 (1111	0000							
4b1111	0000								(0001	0010 (0011	0100 (0101	0110 (0111	1000 (1001	1010 (1011	1100 1101	1110 (1111
1'b1																
4'b0000	0000 (0001	(0000	0001 0000	(0001	(0000	(0001	0000 (0001	(0000	(0001	(0000	0001 0000	(0001	(0000	(0001	0000 0001	(0000
32b0000000000	000000000000000000000000000000000000000	000000000000	00000000													
	4b0000 4b1111 1b1 4b0000	4b0000 0000 0001 4b1111 0000 1b1 4b0000 0000 00001	4b0000 0000 0001 0010 0011 4b1111 0000 1b1 4b0000 0000 0001 0000 0000	450000 (0000)0001 (0010)0011 (0100)0101 451111 (0000)0001 (10000	#50000 0000 10010 10010 1010 1010 10110 10111 1011 1011 1011 10111 1011	450000 0000 (0001 0010 0011 0100 (0101 0110 0111 1000 11001 1101 1111 1000 11001 1101	#50000 0000 70001 0010 70011 0100 70101 0110 70111 1000 71001 1010 71011 710111 71011 71011 71011 71011 7101	450000 0000 (0001 0010 1010 1010 1010 10	450000 0000 10010 10010 1010 1010 1010 1	#50000 0000 70001 0010 70011 0100 70111 0100 71011 1000 71011 1100 71111 1100 71111 1110 71111 0000 10111 1111	450000 0000 0010 0010 0011 0100 0111 1000 1001 1010 1011 1100 1101 1110 1111 1000 1001 0010 0011 0110 0111 1000 1001	450000 0000 (0001 0010 0011 0100 (0101 0110)0111 1000 (1001 1110)1111 1110 (1111 0000 1001 0110)0101 1010 1010 1010 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111 1110 1111	#50000	450000 0000 0001 0010 0011 0100 0111 1000 1001 1010 1011 1100 1101 1110 1111 0000	#50000 0000 0001 0010 0011 0110 0111 1100 1101 1101 1100 1101 1110 1111 11000 1001 0110	450000 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1111 1000 1011 1110 1111 1000 1011 1010

4.5 Arithmetic Shifter Top Module:

4.5.1 4-bit Arithmetic Shifter:



4.5.2 Arithmetic Shifter Top Module Implementation:



. 4

- This Module when Sel = 00, the output will equal Arithmetic Shift Right(A), when Sel = 01, the output will equal Arithmetic Shift Left(A), when Sel = 10, the output will equal Arithmetic Shift Right(B)
 And when Sel = 11, the output will equal Arithmetic Shift Left(A).
- Notes about Arithmetic Shift:

```
// Arithmetic right shift (>>>) replicates the sign bit
reg signed [3:0] A = 4'b0101;
result = A >>> 1;
A = 0101 \rightarrow >>> 1 \rightarrow 0010
reg signed [3:0] A = 4'b1100;
result = A >>> 1;
A = 1100 // -4
A >>> 1 = 1110 // -2
reg signed [3:0] A = 4'b0101;
result = A >>> 2;
A >>> 2 = 4'b0001 // which is +1 in 4-bit signed
reg signed [3:0] A = 4'b1100;
result = A >>> 2;
A >>> 2 = 4'b1111 // which is -1 in 4-bit signed
// Arithmetic right shift (>>>) works as Logical Shift Right
reg [3:0] A = 4'b1100;
      = 1100 // 12
A >>> 1 = 0110 // 6
      = 1100 // 12
A >>> 2 = 0011 // 3
reg [3:0] A = 4'b0101;
     = 0101 // 5
A >>> 1 = 0010 // 2
       = 0101 // 5
A >>> 2 = 0001 // 1
```

```
// in Case of Unsigned & Signed Numbers: For Arithmetic Shift Left:
// Arithmetic Left Shift (<<<) behaves the same as Logical Left Shift (<<).
// This is true for both signed and unsigned types.

reg [3:0] A = 4'b0101;

A = 0101 (5)
A <<< 1 = 1010 (10)

A = 0101 (5)
A <<< 2 = 0100 (4)

reg signed [3:0] A = 4'b0101;

A = 0101 (+5)
A <<< 1 = 0101 <<< 1 = 1010 = (-6) in 4-bit signed

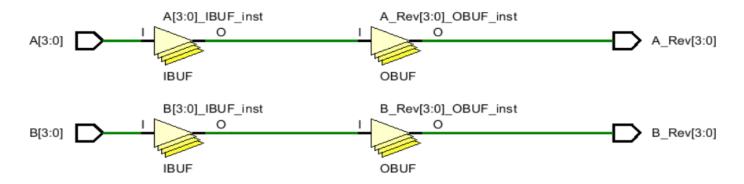
A = 0101 (+5)
A <<< 2 = 0101 <<< 2 = 0100 = (+4) in 4-bit signed
```

4.5.3 Test Bench Simulation:

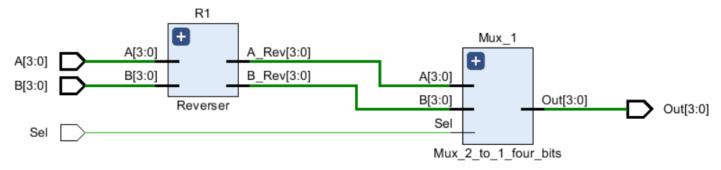


4.6 Reverser Top Module:

4.6.1 4-bit Reverser:



4.6.2 Reverser Top Module:



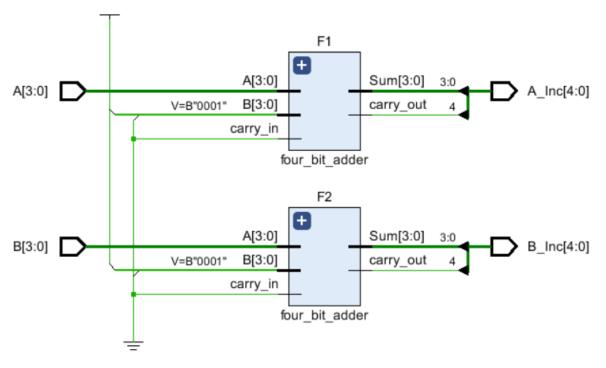
This Module when Sel = 0, the output will equal Reverse_A
 And when Sel = I, the output will equal Reverse_B.

4.6.3 Test Bench Simulation:

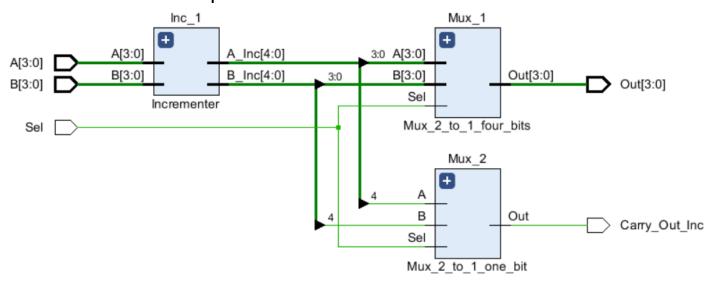


4.7 Incrementer Top Module:

4.7.1 4-bit Incrementer:

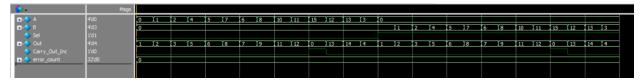


4.7.2 Incrementer Top Module:



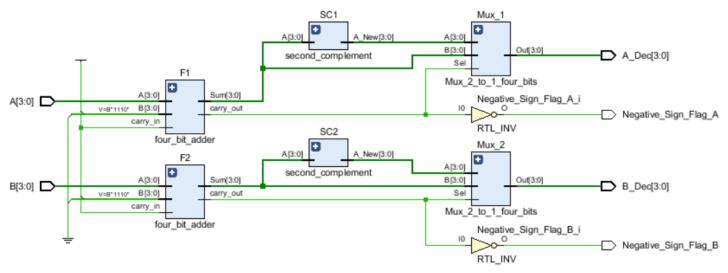
• This Module when Sel = 0, the output will equal A_Incremented and Carry_Out_A And when Sel = I, the output will equal B_Incremented and Carry_Out_B.

4.7.3 Test Bench Simulation:

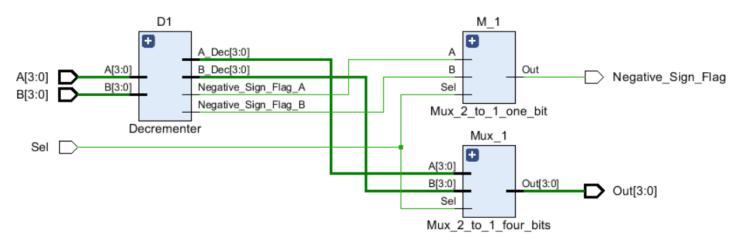


4.8 Decrementer Top Module:

4.8.1 4-bit Decrementer:



4.8.2 Decrementer Top Module:



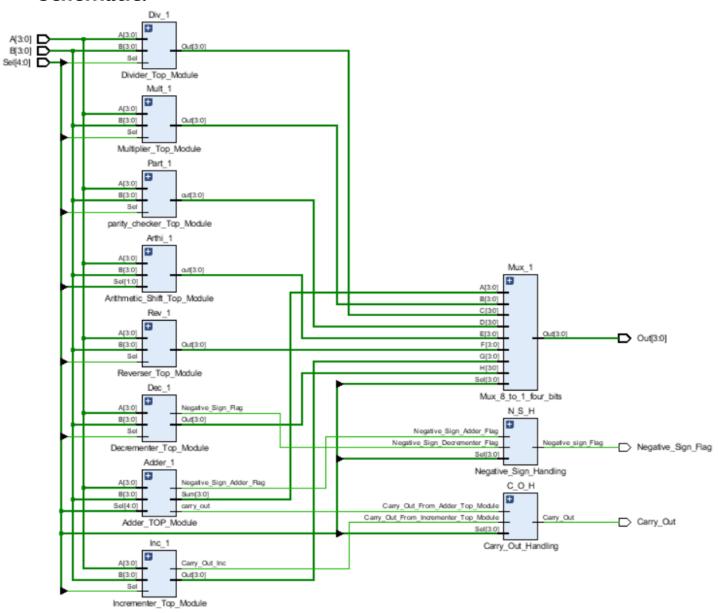
This Module when Sel = 0, the output will A_Decremented and Negative_Sign_Flag_For_A
 And when Sel = 1, the output will equal B_Decremented and Negative_Sign_Flag_For_B.

4.8.3 Test Bench Simulation:

4	Msgs																								
■ → A	4'd0	0	(1	2	(3	4	(5	6	7	8	9	10	(15	0											
 → B		0												1	0	2	(3	4	(5	6	7	8	9	10	15
♦ Sel																									
⊕ Out		1	(0	1	(2	3	(4	5	(6	7	(8	9	(14	0	1		(2	3	(4	5	(6	7	(8	9	14
Negative_Sign_Flag																									
→ derivative error_count		0																							

4.9 Arithmetic& Arithmetical Shift Unit Top Module:

Schematic:

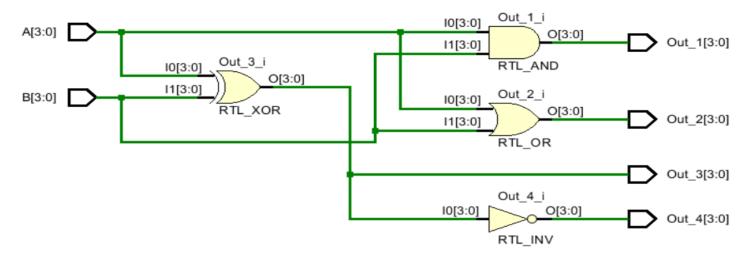


• This Module Works according to the Selectors table which is in the beginning of the report as you have five selectors here → you will enter Sel[4:0] and Sel[5] isn't found here because it is constant for all Arithmetic& Arithmetical Shift operations, so we use it to select between Arithmetic& Arithmetical Shift Unit and Logic& Logical Shift unit because Sel[5] for the first is 0 and for the second is 1.

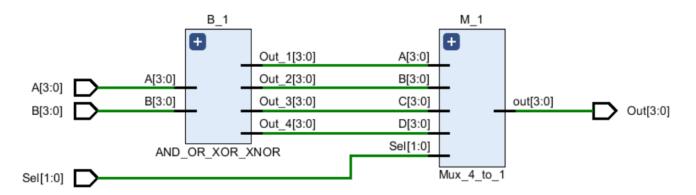
5. Detailed Design Architecture for Logic& Logical Shift Unit:

5.I AND_OR_XOR_XNOR Operations Top Module:

5.1.1 AND_OR_XOR_XNOR Operations Block:



5.1.2 AND_OR_XOR_XNOR Operations Block:



This Module When Sel = 00 → Output = A & B, and When Sel = 01 → Output = A | B
 And When Sel = 10 → Output = A ^ B, and when Sel = 11 → Output = ~(A^B)

• Test Bench Simulation:

When Sel = 00 and 01

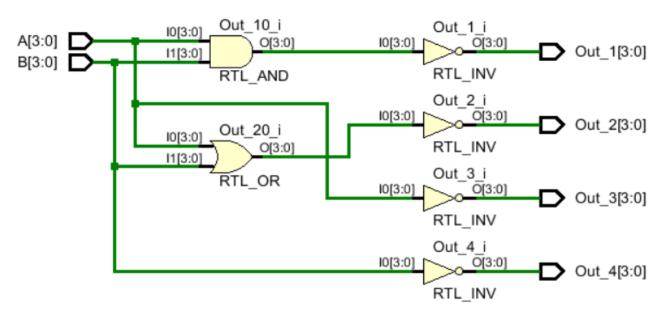
\$ 1 √	Msgs																						
₽ - ♦ A	4'b0001	0001	0010	0100	1000	1100	1010	1111	0011	0110	1001	0111	1000	0100	0010	0001	0011	0101	10000	1100	1001	0010	0100
 → B	4'b1111	1111	1110	1010	1100	1111	0110	0001	0101	1011	1001	0111	0001	0010	0100	1000	1100	1010	1111	0011	0110	1001	0111
正 - 分 Sel	2'b00	00											01										
E - 分 Out	4'b0001	0001	0010	0000	1000	1100	0010	0001		0010	1001	0111	1001	0110		1001	1111					1011	0111
error_count	32'b00000000000	0000000	0000000	0000000	00000000	00																	

When Sel = 10 and 11

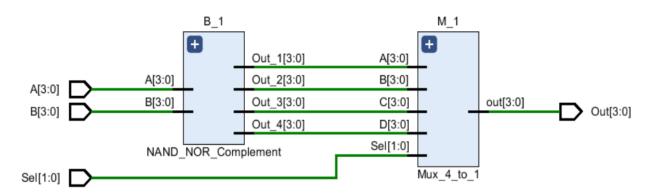
∻	Msgs																						
⊕ - ♦ A	4b0001	1000	0100	0010	0001	0011	0101	10000	1100	1001	0010	0100	1000	0100	0010	0001	0011	0101	0000	1100	1001	0010	0100
⊞- ◆ B	4b1111	0001	0010	0100	1000	1100	1010	1111	0011	0110	1001	0111	0001	0010	0100	1000	1100	1010	1111	0011	0110	1001	0111
≖ – 分 Sel	2'b00	10											11										
- → Out	4b0001	1001	0110		1001	1111					1011	0011	0110	1001		0110	0000					0100	1100
	32'b00000000000	00000000	000000000	000000000	00000																		

5.2 NAND_NOR_Complement Operations Top Module:

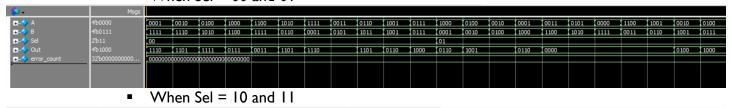
5.2.1 NAND NOR Complement Operations Block:



5.2.2 NAND_NOR_Complement Operations Top Module Block:



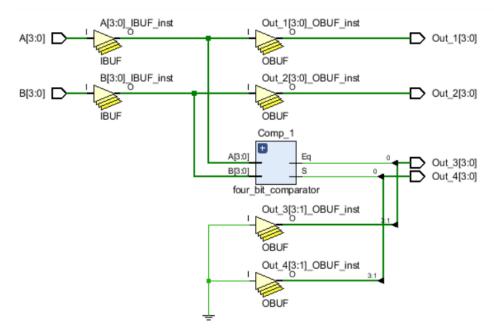
- This Module When Sel = $00 \Rightarrow$ Output = \sim (A & B), and When Sel = $01 \Rightarrow$ Output = \sim (A | B) And When Sel = $10 \Rightarrow$ Output = \sim (A), and when Sel = $11 \Rightarrow$ Output = \sim (B)
- Test Bench Simulation:
 - When Sel = 00 and 01



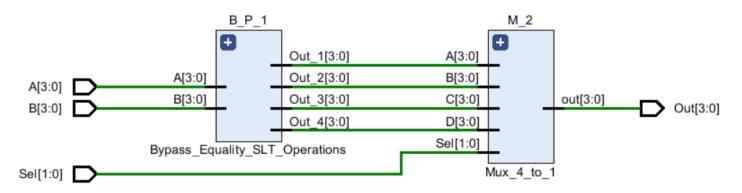
41 •	Msgs																						
B -◆ A	4'b0000	1000	0100	0010	0001	0011	0101	0000	1100	1001	0010	0100	10000										
₽- ◆ B	4b0111	0000											0001	0010	0100	1000	1100	1010	1111	0011	0110	1001	0111
⊪- Sel	2b11	10											11										
⊞ – ∜ Out	4b1000	0111	1011	1101	1110	1100	1010	1111	0011	0110	1101	1011	1110	1101	1011	0111	0011	0101	0000	1100	1001	0110	1000
error_count	32'b00000000000	00000000	000000000	000000000	00000																		

5.3 Bypass, Equality and SLT Operations Top Module:

4.3.1 Bypass, Equality and SLT Operations Block:



4.3.2 Bypass, Equality and SLT Operations Top Module Block:

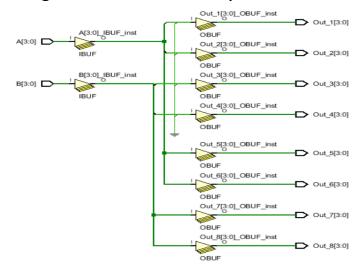


- This Module When Sel = 00, Output → A
 When Sel = 01, Output → B
 When Sel = 10, Output → I if (A==B) and 0 otherwise.
 And when Sel = 11, Output → I if (A<B) and 0 otherwise.
- Test Bench Simulation:

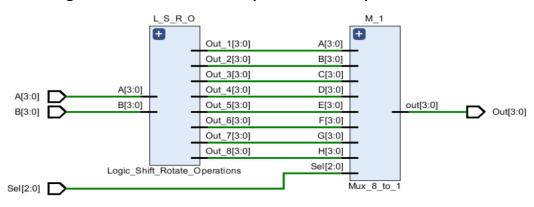


5.4 Logical Shift & Rotate Operations Top Module:

5.4.1 Logical Shift & Rotate Operations Block:



5.4.2 Logical Shift & Rotate Operations Top Module:

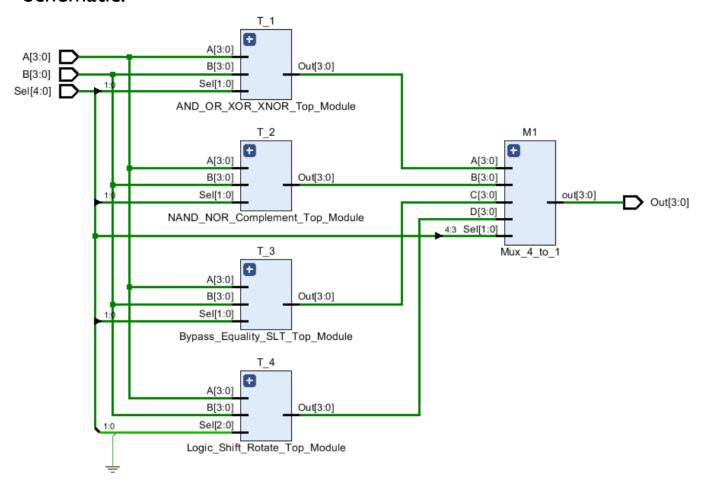


- This Module When Sel = $000 \rightarrow logic shift right(A)$, when Sel = $001 \rightarrow logic shift left(A)$, when Sel = $010 \rightarrow logic shift right(B)$, when Sel = $111 \rightarrow Rotate Right(A)$, when Sel = $101 \rightarrow Rotate Left(A)$, when Sel = $111 \rightarrow Rotate Left(B)$.
- Test Bench Simulation:

	162	ιD	en	CH	JIII	lui	auc	ЛI.																
\$ 1+	Msgs																							
B -♦ A			010 010	0 (1000	1100 (10	10 1111	1 (0011	0110 (1001 (0111	(0001 (0010 (010	1000 (1100 101	0 (1111	0011 (0	110 1001			V V	, l	V	v		
■- 小 B ■- 小 Sel		0000								Y001							100		10100	1000 111	00 1010	1111 ,00	11 10110	1001 10111
■- Out	4°b0000	0000 X0	001 1001	0 (0100	10110 101	01 1011	1 (0001	0011 X			0100 Ĭ 100	10000 Y	1000 I 010	0 11110	0110) 1	100 I 00 10			10010	0100 [01	10 (0101	0111 (00	01 [0011	0100 10011
error_count	32'b00000000000	0000000	00000000	00000000	00000000													,			'			
•																								
♦ 1 •	Msgs																							
■ - ♦ A	4'b0000	0000				j				10001 (0010 0100	1000	1100 (101	0 11111	0011 0	110 (1001	0111 (00	01 0010	(0100	1000 (11	00 1010	1111 00	11 (0110	1001 (0111
⊞ - ∲ B			0010 (010	00 1000	(1100 11	10 (111	1 0011	0110	1001 (0111	0000														
■		011								100							(10							
⊕ Out						00 111	0110	1100	0010 (1110	1000	0001 0010	0100	0110 (010	1 1111	1001 0	011 (1100	1011 (00	10 0100	1000	0001 10	01 0101	1111 (01	10 1100	0011 (1110
error_count	32'b00000000000	0000000	00000000	00000000	00000000			_						_			_		_		_			
<u> </u>	Msgs																							
⊞ - ∜ A	4'b0000	0111	2022																				=	
B-✓ A B-✓ B	4b1000	0000		0010	I 0 100	1000	1100	1010	1111	0011	[0110	1001	0111	0001	10010	[0100	1000	1100	I 1010	1111	0011	10110	1001	0111
₽-♦ Sel	3'b110	101		10010	10100	1000	11100	1010	11111	10011	10110	1001	10111	1111	10010	10100	1000	11100	1010	11111	10011	10110	1001	0111
∓ - ∕ Out	4'b0100	1110		0001	10010	0100	0110	10101	1111	1001	10011	11100	1011	0010	I 0 100	I 1000	10001	I 1001	10101	1111	10110	I 1100	I 0011	1110
→ error_count	32'b00000000000				000000000		3110	0101		1551	3311	1100	1011	3313	0100	1000	10001	1331	0101		0110	1100		
p—	1																							

5.5 Logic & Logical Shift Unit Top Module:

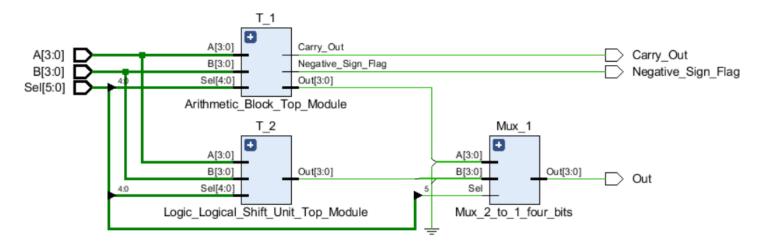
• Schematic:



• This Module Works according to the Selectors table which is in the beginning of the report as you have five selectors here → you will enter Sel[4:0] and Sel[5] isn't found here because it is constant for all Logic& Logical Shift operations, so we use it to select between Arithmetic& Arithmetical Shift Unit and Logic& Logical Shift unit because Sel[5] for the first is 0 and for the second is 1.

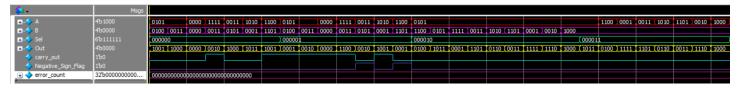
6. Simulation & Testing for the whole System:

• Schematic:



Waveform Snippets for the whole System:

 When Sel = 000000 and when 000001 and when 000010 and when 000011(Adder Top Module)



• When Sel = 000100 and when 000101 (Multiplication Top Module)



• When Sel = 001100 and when 001101(Division Top Module)



When Sel = 010100 and when 010101(Parity Checker Top Module)



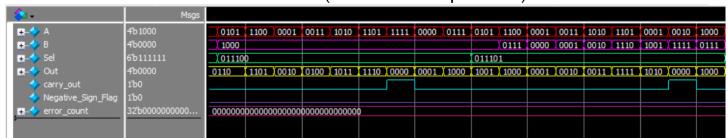
• When Sel = 001010 and when 001111 and when 011011 and when 011111(Arithmetic Shifter Top Module)



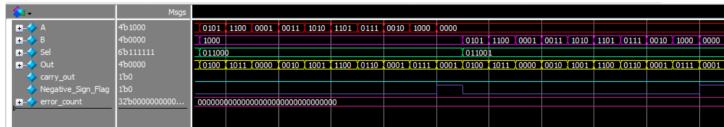
When Sel = 010000 and when 010001(Reverser Top Module)



• When Sel = 011100 and when 011101(Incrementer Top Module)



• When Sel = 011000 and when 011001 (Decrementer Top Module)



 When Sel = 100000 and when 100001 and when 100010 and when 100011(AND OR XOR XNOR operations Block Top Module)



 When Sel = 101100 and when 101101 and when 101110 and when 101111(NAND_NOR_Complement operations Block Top Module)



 When Sel = 110000 and when 110001 and when 110010 and when 110011(Bypass_Equality_SLT operations Block Top Module)





 When Sel = 111000 and when 111001 and when 111010 and when 111011 and when 111100 and when 111101 and when 111110 and when 111111 (Logic Shift and Rotate operations Block Top Module)



ALSU Project With 40 Operations in Modular Design

1– ♦ A	4'b00	000	0101 1	00 0001	0011	1010 110	0001	0010	1000	0101	1100 0001	0011 10	10 110	1 0001	0010 1000	0101 11	00 1	0001 0	0011	1010	1101	0001 0	010 10
9 - ∳ B	4'b00					1000 1010					1100 / 0001	0011 / 10	10 110	1 / 0001	0010 _/ 1000	0101 /11	00	0001 / 0	7011	1010	1101	0001 / 0	010 10
- *	6'b01			10 ,0011	¥ 0 100	1000 / 1010	1100	1110	11111							444404	=						
⊩- Sel			111011		V	ļ				111100		V			V	111101	_					V	
- Out	4'b00	001	0010 0	00 0110	1000	0000 0100	1000	1100	1110	1010	0110 1000	1001 01	01 1110	0 1000	0001 0100	1010 10	01 (0010 0	0110	0101	1011	0010 0	100 00
carry_ou									_	\rightarrow							_		\rightarrow				\rightarrow
Negative	1b1																_						
👍 error_coւ	ınt 32'b0	0000	000000000	000000000	0000000	000000																	
		_																					
	Msgs																						
	Msgs 4'50000	0101	1100	0001	0011	1010	1101	000	01	0010	1000	0101	1100	0001	0011	1010	1101	00	001	0010		1000	
♦ A ♦ B	4'b0000	0101	1100 0001	0001 0010	0011 0011	1010	1101	000		00 10 100 1	1000		1100	0001			1101 0101			0010		1000	
A B Sel	4'b0000 4'b0000																						
A B Sel Out	4'b0000 4'b0000 6'b011001 4'b0001	1000							11			1000 111111				0100			111				
A B Sel Out carry_out	4'b0000 4'b0000 6'b011001 4'b0001 1'b0	1000 111110	0001	0010	0011	0100	0101	111	11	1001	0000	1000 111111	0001	0010	0011	0100	0101	11	111	1001		0000	
A B Sel Out carry_out Negative	4'b0000 4'b0000 6'b011001 4'b0001 1'b0	1000 1111110 0100	1000	0010	1001	0100	0101	111	11	1001	0000	1000 111111	0001	0010	0011	0100	0101	11	111	1001		0000	
A A B Sel Out A carry_out A Negative	4'b0000 4'b0000 6'b011001 4'b0001 1'b0	1000 1111110 0100	0001	0010	1001	0100	0101	111	11	1001	0000	1000 111111	0001	0010	0011	0100	0101	11	111	1001		0000	