University of Tripoli Computer and Electronics Engineering department



Design and Implementation of Single Precision Floating-point Arithmetic Logic Unit for RISC Processor on FPGA

FPGA Design for Embedded Systems EC 582

By: Noor Alhuda Adela (022152760) Dania K. Alamen (022160629)

> Supvr: **Dr. Mohamed M. Eljhani**

> > March 2022

Summary

In this project report, we discuss the design of an arithmetic logic unit (ALU) and a floating-point unit (FPU) architecture that together performs all arithmetic and logical operations of computer processors and gives the flexibility of scalability up to 32-bits. The ALU and FPU were implemented in Verilog, simulated, and tested in ModelSim Altera.

Table of Contents

1. In	troduction	3
1.1.	Project Timeline	. 4
2. Aı	rithmetic Logic Unit Overview	4
2.1.	Design Block	.5
2.1.1.	Opcode Decoder	.5
2.1.2.	Arithmetic Block	. 7
2.1.3.	Comparator block.	.8
2.1.4.	Logical block.	. 9
2.1.5.	Shifter/Rotate block	. 9
3. Fl	oating-Point Unit Overview	10
3.1.	The IEEE 754 Standard	10
3.2.	FPU Instructions.	11
3.2.1.	Floating-Point Adder/Subtractor.	11
3.2.2.	Floating-Point Multiplication	12
4. Al	LU & FPU Synthesis Results	13
5. Co	onclusion	22

1. Introduction

This paper focuses on the design of the arithmetic logic unit (ALU) and floating-point unit (FPU), which perform integer mathematical and logic operations, and decimal mathematical operations, respectively. The end goal of this thesis is the design, Verilog implementation, and synthesis of the arithmetic logic unit and floating-point unit of a 32-bit reduced instruction set computer (RISC) processor.

The input consists of an instruction that contains an operation code (opcode), one or more operands. The operation code tells the ALU what operation to perform and the operands are used in the operation. (For example, two operands might be added together or compared logically.) The format may be combined with the opcode and tells, for example, whether this is a fixed-point or a floating-point instruction. The output consists of a result that is placed in a storage register and settings indicate whether the operation was performed successfully.

The remainder of the paper is divided into two parts - one each for the ALU and FPU. The first part describes the implementation of the functions of the arithmetic logic unit. Integer adders, multipliers, and dividers are discussed in detail, with the basic logic operations - AND, OR, XOR, etc. The second part of the paper describes the floating-point unit. Implementations of single-precision adder/subtractors, multipliers, and dividers that conform to the IEEE 754 standard are presented.



Figure 1. The general architecture of hybrid ALU



Figure 2: Flowchart

1.1 Project Timeline

We planned to follow a Bottom to up approach while designing this ALU. Firstly, we implemented simple logic blocks such as Comparator, Adder, Shifter, etc. in Verilog. Then after verifying their performance in the ModelSim simulator, with respect to generated signal patterns as inputs, we implemented the whole design.

Part I

2. Arithmetic Logic Unit Overview

An arithmetic logic unit (ALU) is an integral essential combinational circuit of any central processing unit (CPU). The processor uses the ALU to perform arithmetic operations such as addition, subtraction, multiplication, and division - as well as logical operations - OR, AND, XOR, inversion, and transformation operations. Additional operations such as data block comparisons. The ability to perform all these results of operations in the ALU is one of the most complex circuits in the CPU.

2.1. Design Block

Our design comprises 5 basic sections. We describe each one below

- Opcode Decoder
- Arithmetic Block
- Logical Block
- Comparator Block
- Shifter Block



Figure 3 . ALU Block Diagram

2.1.1. Opcode Decoder:

The proposed design consists of a primary opcode decoder unit that activates the respective function block based on the instruction opcode performed by the processor and forwarded to the ALU block. The respective output lines will feed the enable lines of respective function blocks.

The decoder is a 6:43 unit i.e., it takes 6 bits opcode as input and in turn activates the respective function block to perform the desired operation.

Selected Instructions

The designed ALU takes a clock signal, two 32-bit operands, and a 6-bit opcode as inputs. A complete list of the implemented instructions can be seen in Table 1.

SELECT			ACTIVE-HIGH DA	ТА	
OPCODE	ARITH	IMETIC	COMPARRATOR	LOGIC	SHIFTING
	S4 S5	5 =00	S4 S5 = 01	S4 S5 =10	S4 S5 =11
S0 S1 S2 S3	Cin=0	Cin=1			
0 0 0 0	OUT=A	OUT=A+1	OUT=A>B	OUT=~A	OUT= shl A
0 0 0 1	OUT=A+B	OUT=A+B+1	OUT=A < B	OUT=~(A^B)	OUT=shl B
0 0 1 0	OUT=A+`B	OUT=A+`B+1	OUT=A≠B	OUT=~A^B	OUT=shr A
0 0 1 1	OUT=-1	OUT=0		OUT=0	OUT=shr B
0 1 0 0	OUT=B	OUT=B+1		OUT=~(A∨B)	OUT=rol A
0 1 0 1	OUT=`B	OUT=`B+1		OUT=~B	OUT=rol B
0 1 1 0	OUT=A-B	OUT=A-B+1		OUT=A	OUT=ror A
0 1 1 1	OUT=B-A	OUT=B-A+1		OUT=A^~B	OUT=ror B
1 0 0 0	OUT=1	OUT=2		OUT=A [∨] B	
1 0 0 1	OUT=0	OUT=1		OUT=~A^B	
1 0 1 0	OUT=A-1	OUT=A		OUT=B	
1 0 1 1	OUT=A+A	OUT=A+A+1		OUT=A^B	
1 1 0 0	OUT=B-1	OUT=B		OUT=1	
1 1 0 1				OUT=A ^{∨~} B	

Table 1. ALU Operations

Using a Combination of These operations, any logic operation can be implemented.

2.1.2. Arithmetic Block:

This block is used to implement arithmetic operations such as Addition, Subtraction, Multiplication, and division. The Accumulator and the auxiliary b register feed as inputs to this block.



Figure 4. Arithmetic Operation

Integer Addition/Subtraction

Fast addition is extremely important in many digital systems. As an elementary school child knows, addition and subtraction are the same operations. Subtraction merely inverts the sign of the second operand. Using this knowledge, it becomes simple to implement integer subtraction using any type of integer adder.

To alter the adder to be able to handle both integer addition and subtraction, logic must be established to perform two's complement conversion on the B operand if subtraction is selected, and to leave B untouched if addition is selected. This can be done in one of two ways: using an inverter and a multiplexer on each bit, or an XOR gate with one bit tied to control, and the other tied to the corresponding bit of B.

Integer Multiplication

Multiplication, like addition, is a heavily used operation that figures prominently in many types of operations. Among many other uses, multiplication is used in signal processing and scientific applications. It is also a common basis for division.

Integer multipliers can be implemented in a variety of ways. Typical implementations are a shift and add. The multiplication operation produces a 64-bit output that utilizes both registers.



Figure 5. Shift and Add Multiplier Circuit



Figure 6: Unsigned Binary Multiplication Flowchart

Integer Division

The division is the least used of the four basic arithmetic operations. As such, it has been the least researched of the four operations and remains the most difficult operation to implement efficiently.



Figure 7: Unsigned Binary Division Flowchart

2.1.3. Comparator block:

This block consists of combinational circuit HDL code which performs bit matching and comparisons. Respective outputs may be used to branch instructions based on the comparison. Similarly, numerically larger, or smaller indications on respective operands may be used.

2.1.4. Logical block:

This block comprises basic logic gate units such as AND, OR, NOT, XOR, etc. Such operations on data operands are served by this block. Outputs are stored in respective latches. The logical operations implement a series of standard logic operations on the operands at the bit level. The AND operation produces a 1 at output bit *i* only if A_i and B_i are both equal to 1. The OR operation produces a 1 at output bit *i* if A_i or B_i is B_i is equal to 1, but not both. The NOR operation is the opposite of the OR operation. A 0 is inserted at the bit position if the operating conditions are not met. The ENV operation inverts each bit of both A and B.

2.1.5. Shifter/Rotate block:

This block consists of basic shifters such as Barrel Shifters and mechanisms for bit rotation. The right shift is implemented both arithmetically - where the operation is sign-extended as it is shifted – and logically - where 0's are inserted into the bit positions that are shifted out.



Figure 8. An example for a 32 bit scaled shifter block

Part II

3. Floating-Point Unit Overview

Floating-Point Units (FPU) are the hardware components that handle decimal mathematical operations in the CPU. Like the ALU, the FPU implements the four basic mathematical operations - addition, subtraction, multiplication, and division - the difference being the number representation scheme utilized. An ALU handles integer values, represented in binary numbers. This means that the entire 32-bits of the bit vector represents the portion of a number to the left of the decimal point. An FPU deals with both the integer and fraction portions of numbers. As there is no way to slide a decimal point into the bit vectors to tell the computer what is the integer and what is the fraction, the operands must be divided into sections representing the sign, exponent, and mantissa of the number.

3.1. The IEEE 754 Standard

The standard supports single-precision 32-bit numbers, and doubleprecision 64-bit numbers. As would be expected, double-precision offers a larger range (11 exponent bits compared to 8) and greater accuracy (52 fraction bits compared to 23) than single-precision. As they operate in the same manner and the focus of the work presented in this paper is on 32-bit inputs, only the single-precision format will be explored.



Figure 9: Single-Precision IEEE 754 Format

Table 2.	Floating-Point:	Special	Cases
----------	-----------------	---------	-------

Number	Sign	Exponent (e)	Fraction (f)
0	Х	00000000	000000000000000000000000000000000000000
8	0	11111111	000000000000000000000000000000000000000
-∞	1	11111111	000000000000000000000000000000000000000
NAN	Х	11111111	nonzero

3.2. FPU Instructions

The instructions implemented for the floating-point unit are added, subtract, multiply and divide.

Operation	OP Code
Addition	00
Subtraction	01
Multiplication	10

Table 3. FPU O	perations
----------------	-----------

3.2.1.Floating-Point Adder/Subtractor

Just as integer addition/subtraction is the most used ALU operation, FP addition/subtraction is the most utilized floating-point operation.



Figure 10. Block Diagram of Floating-Point Adder/Subtractor

The difference between the exponents is used to determine the amount of right shifting necessary to align the smaller operand with the larger operand.



Figure 11: FP Addition & Subtraction Flowchart

3.2.2.Floating-Point Multiplication

Floating-point multiplication has nearly as many far-reaching applications as floating-point addition/subtraction. As a result, it is important to implement an efficient multiplier design.



Figure 12. Generic FP Multiplier Block Diagram



Figure 13: Floating Point Multiplication Flowchart

4. ALU & FPU Synthesis Results

Each of the arithmetic blocks and floating-point described previously were implemented in the Verilog hardware descriptive language. Test benches were developed in ModelSim, and executed in order to ensure that the functions performed as expected.



Figure 14: A – 1



Figure 15: A



Figure 16: A - B



Figure 17: A - B + 1

🎓 🗸	Msgs																			
Operands																				
	001100000111110010111101110101010	01000001000	10111110	1001100	001100000	111110010	11101110	01010												10110111011
· ⊥ → /main_tb/B	101101110111111000100001010101010	00111111000	01000011	1111101	101101110	111111000	100001010	11010												
Cin	·																			
🔶tb/Cin	0								1				l							<u>ک محمد او</u>
Opcode																				ک کے ا
opcode	0010000	1010000	1100000		0000110				0001010				0000101		001	0000		0010001		0010010
Result																				ككعك
њ/о∪т	000000000000000000000000000000000000000	01000001000	1110000110	001011	011110001	1111111	01111000	.1111111	001100000)1111110	001100000	1111110	010010001	1000000	000	00000	0000000	10000000	0000000	000000000000000
Status																				ک کک کے
◆ …ut/ALU	St1 St0																			
	St0															_				
→ut/CMP	510																			
→ut/LOG	511																			
	sto																			
🧄ut/Add	sto																			
👍ut/Sub	St0																			
🧄uut/Mul	St0				-															
b/uut/A	00110000011111001011101110101010	01000001000	10111110	1001100	001100000	111110010	11101110	01010												10110111011
b/uut/B	10110111011111100010000101011010	00111111000	01000011	1111101	101101110	111111000	100001010	11010												
=opcode	0010000	1010000	1100000		0000110				0001010				0000101		001	0000		0010001		0010010
🁍 …uut/Cin	St0																			
ut/OUT	000000000000000000000000000000000000000	01000001000	11000011	001011	011110001	111111	01111000	.1111111	001100000)111110	001100000	111110	010010001	1000000	000	00000	000000	10000000	0000000	000000000000000
U_ALU	000000000000000000000000000000000000000	000000000000000000000000000000000000000							001100000)111110	001100000	111110	010010001	1000000	. 000	00000	000000	10000000	0000000	00000000000
U_FPU		01000001000																		ک کک ک
🖽 🎝/alu1/a		01000001000	1		0															10110111011
🖽 🎝/alu1/b		00111111000	01000011	1111101	101101110	111111000	100001010	11010												و کسی
	0010000	1010000	11100000		0000110				0001010				0000101		1001	0000		100 1000 1		0010010
A I I I I I I I I I I I I I I I I I I I	1900 ns	30	0 ns	40) ns	500) ns	60) ns	700) ns	800) ns	90	0 ns		100	00 ns	110	10 ns
🗟 🖉 🤤 Cursor 1	917 ns														917	'ns				

Figure 18: A > *B*

🔹 -	Msgs	1															
Operands																	
🚛	001100000111110010111101110101010	01000001000	101111101001100	0011000001	111100101	1101110101010											10110111011
🖃	101101110111111000100001010101010	00111111000	010000111111101	1011011101	111110001	0000101011010											
Cin		•															
🧇tb/Cin	0						1										
Opcode																	
H	0010001	1010000	1100000	0000110			0001010				0000101		0010000		00 <mark>1</mark> 0001		0010010
Result																	
tb/OUT	100000000000000000000000000000000000000	01000001000	[110000110001011	0111100011	.11111	111100011111111	00110000	<u>01111110</u>	001100000	1111110	01001000	1000000	000000000	000000	[<u>1000000</u>	<u>00000000</u>	
Status																	
	St1 St0																
ARITH	sto																
ut/CMP	510																
ut/LOG	Sto																
SHIFT	sto																
🔶ut/Add	Sto																
🔥ut/Sub	St0																
🧄uut/Mul	St0																
🗐 🥠b/uut/A	00110000011111001011101110101010	01000001000	101111101001100	0011000001	111100101	1101110101010											10110111011
🖃 🍫b/uut/B	101101110111111000100001010101010	00111111000	010000111111101	1011011101	111110001	0000101011010											کر کے ا
🖃 🌧 opcode	0010001	1010000	1100000	0000110			0001010				0000101		0010000		0010001		0010010
iuut/Cin	St0						1				1						
ut/OUT	100000000000000000000000000000000000000		110000110001011														
U_ALU	100000000000000000000000000000000000000		000000000000000000000000000000000000000				00110000	0111110	001100000	111110	01001000	000000	000000000	000000	[<u>100</u> 0000	00000000	
U_FPU	000000000000000000000000000000000000000		110000110001011														
/alu1/a	00110000011111001011101110101010		101111101001100	î													10110111011
- /alu1/b	101101110111111000100001010101010	00111111000	010000111111101	1011011101	111110001	0000101011010									0010001		0010010
Alex Now	1900 ns																dama and and
		30	0 ns 40	0 ns	500 (ns 60	0 ns	70	0 ns	80	0 ns	90	0 ns	100	0 ns	11	00 ns
🗟 🎸 🤤 Cursor 1	1015 ns														1015 ns		

Figure 19: A < *B*

\$ 1•	Msgs																				
Operands																					
🚛	101101110111111000100001010101010	001100000	11111001	011101110	101010											1011	01110	11111100	010000101	011010	
	1011011101111110001000010101011010	101101110	11111100	010000101	011010																
Cin																					
🔶tb/Cin	0					1				1											
Opcode	·																				
	0010010	0000110				0001010				0000101		0010000		0010001		0010	010		0100001		0101001
Result																					<u> </u>
tb/OUT	000000000000000000000000000000000000000	011110001		01111000	1111111	100110000	01111110	001100000	111110	010010001	1000000	00000000	000000	110000000	0000000	10000	000000	000000	01001000	1000000	00000
	St1																				
ut/FPU	StD																				
ARITH	Sto											-									1
ut/CMP	St1																				
ut/LOG	St0																				
stift	St0																				
👍ut/Add	St0																				
🧄ut/Sub	St0																				
	St0																				
🖅 🕁b/uut/A	10110111011111100010000101011010	001100000	11111001	011101110	101010											1011	01110	11111100	010000101	011010	
🖅 🎝b/uut/B	10110111011111100010000101011010	101101110	11111100	010000101	011010																
🖅 🚽 😳 🖅	0010010	0000110				0001010				0000101		0010000		0010001		0010	010	_	0100001		0101001
🔶uut/Cin	Sto																				
H	000000000000000000000000000000000000000	011110001																			
U_ALU	000000000000000000000000000000000000000	011110001				00110000	JIIII0	0001100000	111110	010010001	1000000	00000000		10000000	0000000	10000	000000	000000	01001000	1000000	00000
	00000000000000000000000000000000000000	000000000														10.1.1	01110	1111100	010000101	011010	
+	1011011101111110001000010101011010	1001100000														101.	01110		10000101		
	0010010	0000110	11111100	010000101	011010	0001010				0000101		0010000		10010001		00.10	010		0100001		0101001
🛎 📰 💿 🛛 Now	1900 ns	00 pc		0 ns		0 ns		1	20) ns	00	0 ns	100	0 ns	110		u u u b		1 10 ns	120	uluuuuu 00 ns
G Cursor 1	1123 ns	00115	50	0.115		ons	70	0.115		7115	90	0.115	100	00 HS	110	1123	Ins	120	0115	150	0.115
Curbor 1	1120110															112.	5 110				

Figure 20: $A \neq B$

2	Msgs																		
Operands															T				_
	1011011101111110001000010101011010	0011000001111	10010111011101	01010							10110111	11111100	10000101	011010	┼- <mark>┤</mark> ───				1001100
+ /main_tb/B															; ; ; ; ; ;				00110011
Cin																			
tb/Cin	0																		
- Opcode																			
	0101001	0001010			0000101		0010000		0010001		0010010		0100001		0101001		0100111		0110000
- Result					Î				Ï				1		1		1		
🕁њ/оит	000000000000000000000000000000000000000	001100000111	110 00110000	0111110	01001000		000000000	000000	10000000	000000	00000000	000000	01001000	1000000	0000000	000000000000000000000000000000000000000	000000000	000000	011000
Status																			
Iut/ALU	St1																		
Iut/FPU	St0																		
III	St0																		
strate the second secon	St0												1						
stut/LOG	St1																		i <u>Ees</u>
shift 🔶 🔶	St0																		
Iut/Add	St0														↓ 				
	St0														⊥				
strategy 🔶 🔶 🔶	St0														↓				
🖃 🎝b/uut/A		0011000001111									10110111	11111100	010000101	011010					001100
🖅 🎝b/uut/B		1011011101111	11000100001010																
🖅 🎝 opcode	0101001	0001010			0000101		0010000		0010001		0010010		0100001		0101001		0100111		0110000
iuut/Cin	St0				ļ														
🖅 🕂ut/OUT		001100000111			1		1		î.		1		1						011000
U_ALU		001100000111			01001000	000000	000000000000000000000000000000000000000	0000000	10000000	0000000	00000000	0000000	01001000	1000000	10000000	000000000000000000000000000000000000000	0000000000	000000	011000
U_FPU		000000000000000000000000000000000000000																	
/alu1/a		0011000001111									10110111	011111100	010000101	011010					001100
···/alu1/b		1011011101111	11000100001010	11010											10101001				
All Now	<u>0101001</u> 1900 ns																		
		00 ns	700 ns	800) ns	90) ns	100	0 ns	110	0 ns	120	0 ns	130	<u> </u>	14)0 ns	15	00 ns
🗟 🍆 🤤 Cursor 1	1313 ns														1313 ns				

Figure 21: ~*A*&*B*

*	Msgs																				
Operands																					
- 	101101110111111000100001010101010	0011000001	111100101	111011101	01010							110110111	0111111000	1000	01010	11010					1001100
+/main_tb/B	101101110111111000100001010101010																				
Cin																					
tb/Cin	0	-																			
- Opcode																					
	0100001	10001010				10000101		100 10000		0010001		10010010		0100	001		10101001		0100111		10110000
- Result						1		1									1				
tь/о∪т	01001000100000011101111010100101	001100000	111110	00110000	0111110	01001000	1000000	00000000		10000000	0000000	. 00000000	000000	0100	10001	000000	00000000	0000000000	00000000	000000	011000
- Status																					
🔶ut/ALU	St1																				
🔶ut/FPU	St0																				
🔶/ARITH	St0																				
	St0																				
	St1																				
I/SHIFT	St0																				
	St0																				
	St0																				
	St0																				
🖅 🎝b/uut/A	10110111011111100010000101011010	0011000001	111100101	111011101	01010							10110111	01111111000	1000	01010	11010					001100
🖃 🍲b/uut/B	101101110111111000100001010101010	1011011101	111110001	100001010	11010																
🖅 🎝 opcode	0100001	0001010				0000101		0010000		0010001		0010010		0100	001		0101001		0100111		0110000
iuut/Cin	St0	1				1															
🖅 🕂ut/OUT	01001000100000011101111010100101	001100000	111110	00110000	0111110	01001000	1000000	00000000	000000	10000000	0000000	. 00000000	000000	0100	10001	000000	00000000	000000000	000000000	00000	011000
U_ALU	01001000100000011101111010100101	001100000	111110	00110000	01111110	01001000	1000000	00000000	000000	10000000	0000000	. 00000000	000000	0100	10001	000000	00000000	000000000	000000000	00000	011000
U_FPU	000000000000000000000000000000000000000	0000000000	000000000	00000000	00000																
🖅 과 😳 🖪 🖪	10110111011111100010000101011010	0011000001	111100101	111011101	01010							10110111	0111111000	1000	01010	11010					001100
/alu1/b/alu1/b		1011011101	111110001	100001010	11010																
🗖 📥 loncode		10001010				0000101		10010000		0010001		0010010		0100	_		0101001		0100111		0110000
Area Now	1900 ns)0 ns		0 ns		0 ns		0 ns		l0 ns		00 ns	120)0 ns	140			00 ns
🚊 🌽 😑 Cursor 1	1224 ns													122	4 ns						

Figure 22: ~(*A*&*B*)

<u>)</u> -	Msgs																				
																					—
	00110000011111001011101110101010	0011000	001111100	101110	10110111	011111100	010000101	011010					00110000	011111001	011101110	101010			⊢		
+	101101110111111000100001010111010					011111100	010000101	011010					00110000	511111001	011101110	101010			Ħ	==	
— Cin — — — — — — — — — — — — — — — — — — —			10111111	000100001	1011010														t t	27	
 tb/Cin	0																				
- Opcode																					
	0110111	0010 1	0010001		0010010		0100001		0101001		0100111		0110000		0110011		0110100		01101	11	
- Result																					
	01011011101111110001000010101101	0000]	10000000	0000000	00000000	0000000	01001000	1000000	000000000	00000000	000000000	000000	01100000	1111100	01011011	1011111	01100000	1111100	01011	011101	1111
- Status	l																				
🔶ut/ALU	St1																			۲۲	
🔶ut/FPU	St0																				
🔶/ARITH	St0																				
🔶ut/CMP	St0																				
🔶ut/LOG	St0												1								
🔶/SHIFT	St1																				
🔶ut/Add	St0																				
🔶ut/Sub	St0																			<u>و ک</u>	
🔶uut/Mul	St0																				
⊢�b/uut/A				0101110	1	011111100	010000101	011010					00110000	011111001	011101110	101010					
⊢�b/uut/B				00100001															-		
⊨� …opcode		0010)	0010001		0010010		0100001		0101001		0100111		0110000		0110011		0110100		01101	11	
💠uut/Cin	StO																				
⊢🔩ut/OUT									000000000000000000000000000000000000000								01100000				
⊢					1	0000000	01001000	1000000	000000000000000000000000000000000000000	00000000	000000000	00000	01100000	1111100	01011011	1011111	01100000	1111100	0101	011101	1111
⊨				000000000																و ک	
🗠/alu1/a				0101110		0111111100	010000101	011010					00110000	011111001	011101110	101010					
🗠/alu1/b			10111111	000100001	01011010																
<u>A loncode</u> Now	0110111 1900 ns	mmm		humun	huuun												0110100		0110		
		1000) ns	110	0 ns	120	0 ns	130)0 ns	140	0 ns	150	0 ns	160	l0 ns	170	0 ns	180	0 ns		1
🖉 🗧 Cursor 1	1825 ns																		1825	ns	

Figure 23: ror B

<u> </u>	Msgs	Î																		
- Operands -																				
THE IMAGE IN THE IMAGE INTERNAL INTE	00110000011111001011101110101010	0011000	00111110	0101110	10110111	011111100	010000101	011010					001100	0001111100	1011101110	0101010				
	101101110111111000100001010111010																			
Cin	L																			
🔶tb/Cin	0																			
Opcode																				
💶 - 🍫 opcode	0110000	0010	0010001		0010010		0100001		0101001		0100111		011000	0	0110011		0110100		0110111	
Result																				
🗉 🔶 ф/ОИТ	01100000111110010111011101010100	0000	10000000	0000000	00000000	0000000	01001000	000000	000000000	0000000000	0000000000	00000	011000	001111100	.0101101	110111111	01100000	1111100	010110111	011111
Status																				
strut/ALU	St1																			
	St0																			
	Sto						-													
ut/CMP	Sto																			
↓ut/LOG	500																			
ut/Add	St0												-							
 ut/Adu ut/Sub 	StD												+							
🧄uut/Mul	sto																			
=	00110000011111001011101110101010	0011000	00111110	0101110	10110111	011111100	0 10000 10 1	011010					001100	0001111100	1011101110	0101010				
				000100001	1															
	0110000		0010001		0010010		0100001		0101001		0100111		011000	0	0110011		0110100		0110111	
🤙uut/Cin	St0																			
🖃 📥ut/OUT	01100000111110010111011101010100	0000	10000000	0000000	00000000	0000000	01001000	000000	000000000	00000000	000000000	00000	011000	001111100	.0101101	1011111	01100000	1111100	010110111	011111
U_ALU	01100000111110010111011101010100	0000)	10000000	0000000	00000000	0000000	01001000	1000000	000000000	0000000000	0000000000	000000	011000	001111100	.0101101	11011111	01100000	1111100	010110111	011111
U_FPU				000000000	_															
🖪 🛷/alu1/a				0101110		011111100	010000101	011010					001100	00011111100	1011101110	0101010				کسے
🖽 🍫/alu1/b			10111111	000100001	01011010															اصعد
	0110000	0010	0010001		10010010		10100001		0101001		0100111		011000		0110011		10110100		0110111	mmmin
AR Now	1900 ns	1000) ns	110)0 ns	120	I0 ns	130)0 ns	140	0 ns	15	00 ns		00 ns	170)0 ns	180) ns	1900 n
📄 🦾 🤤 Cursor 1	1537 ns												153	7 ns						

Figure 24: shl A

1 •	Msgs															
— Operands —																
🛶 /main_tb/A	010000001111110000000000000000000000000	000000000	010000	01	010000010	101111101	00110000011	11001011101	110101010					10110111011	111100010000	101011010
⊢� /main_tb/B	001111100100000000000000000000000000000	000000000	001111	100	001111110	010000111	10110111011	11100010000	101011010							
🔶tb/Cin	0															
Opcode	-															
-🔶opcode	1000000	0000000	100000	0	1010000	1100000	0000110		0001010		0000101	0010000	0010001	0010010	0100001	0101001
Result	-															
🔷њ/оит	010000010000000100000000000000000000000	000000000	010000	10	0 100000 10	110000110	011110001	011110001	001100000	001100000	010010001	000000000	. 100000000	. 000000000	010010001	000000000
Status																
🔶ut/ALU	St0		1													
🔶ut/FPU	St1 🔶															
🔶/ARITH	St0		1													
🔶ut/CMP	St0															
🔶ut/LOG	St0															
🔶/SHIFT	St0															
🔶ut/Add	St1 ←			1												
🔶uut/Sub	St0															
🔶uut/Mul	St0															
🧄b/uut/A	010000001111110000000000000000000000000	000000000	010000	01	0 100000 10	101111101	00110000011	11001011101	110101010					10110111011	111100010000	101011010
🤣b/uut/B	001111100100000000000000000000000000000	00000000	001111	100	001111110	010000111	10110111011	111100010000	101011010							
🧄opcode	1000000	0000000	100000	0	1010000	1100000	0000110		0001010		0000101	0010000	0010001	0010010	0100001	0101001
🤙uut/Cin	St0								1		1					
👍ut/OUT	010000010000001000000000000000000000000	000000000	010000	10	010000010	110000110	011110001	011110001	001100000	001100000	010010001	000000000	. 100000000	. 000000000	010010001	000000000
🔶U_ALU	000000000000000000000000000000000000000	00000000000	000000	00000	00000000		011110001	011110001	001100000	001100000	010010001	000000000	. 100000000	. 000000000	010010001	000000000
🧇U_FPU	0 100000 10000000 100000000000000000000	000000000	010000	10	010000010	110000110	000000000000000000000000000000000000000	000000000000000000000000000000000000000	00000000							
🧽/alu1/a	010000001111110000000000000000000000000	000000000	010000	01	010000010	101111101	00110000011	11001011101	110101010					10110111011	111100010000	101011010
🥠/alu1/b	001111100100000000000000000000000000000	000000000	001111	100	001111110	010000111	10110111011	11100010000	101011010							
📥 oncoda	100000	0000000	1100000		1010000	1100000	0000110		0001010		0000101	0010000	0010001	0010010	0100001	0101001
🕤 Now		ns		200) ns	40	Dins	60			10 ns	10	00 ns	120	00 ns	111111111
Cursor 1	157 ns		15	7 ns												

Figure 25: FPU Adder

Floating-point numbers

	0 10000001	111 1100 0000 0000 0000 0000
	0 01111100	100 0000 0000 0000 0000 0000
	Exponent	Fraction
	1000001	111 1100 0000 0000 0000 0000
Step 1	01111100	100 0000 0000 0000 0000 0000
	10000001	1.111 1100 0000 0000 0000 0000
Step 2	01111100	1.100 0000 0000 0000 0000 0000
	10000001	1.111 1100 0000 0000 0000 0000
Step 3	- 01111100	1.100 0000 0000 0000 0000 0000
		ift amount)
	10000001	1.111 1100 0000 0000 0000 0000
Step 4	1000001	0.000 0110 0000 0000 0000 0000 0000
	10000001	1.111 1100 0000 0000 0000 0000
Step 5	10000001 +	
		10.000 0010 0000 0000 0000 0000
Step 6	10000001	10.000 0010 0000 0000 0000 >> 1
	+ 1 10000010	1.000 0001 0000 0000 0000 0000
Step 7	(No rounding r	necessary)
Step 8	0 10000010	000 0001 0000 0000 0000 0000

Figure 26: Floating-point Addition

- \$ 1 •	Msgs															
Operands																
	010000010001110000000000000000000000000	000000000	010000001	01000	0010	101111101	00110000011	111001011101	10101010					10110111011	111100010000	101011010
+	001111110001000000000000000000000000000	000000000	001111100	00111	1110	010000111	10110111011	111100010000	01011010							
Cin																
🔶tb/Cin	0															
Opcode																
🖅 🔶opcode	1010000	000000	1000000	10100	00	1100000	0000110		0001010		0000101	0010000	0010001	0010010	0100001	0101001
Result																
🗉 🔷tb/ОUТ	010000010001001100000000000000000000000	000000000	010000010	01000	0010	110000110	011110001	011110001	001100000	001100000	010010001	000000000	100000000	000000000	010010001	0000000000000
Status	l															
Iut/ALU	St0															
Iut/FPU	St1 🔶						l									
I/ARITH	St0		l													
struct/CMP	St0														<u> </u>	
structure 🔶 🔶	St0				_											
SHIFT	St0				_											
strut/Add	St0															
uut/Sub	St1															
uut/Mul	Sto				_											
b/uut/A	010000010001110000000000000000000000000													10110111011	111100010000	101011010
b/uut/B			001111100	т — т			1	111100010000								
• opcode	1010000 St0	000000	1000000	10100	00	1100000	0000110		0001010		0000101	0010000	0010001	0010010	0100001	0101001
uut/Cin ut/OUT		000000000	010000010	01000	0010	110000110	011110001	011110001	001100000	001100000	010010001	000000000	100000000	000000000	010010001	000000000000
ut/OUT	000000000000000000000000000000000000000		010000010													00000000000000
	010000010001000100000000000000000000000		0100000010				î .	î .		001100000	010010001		.100000000		,010010001	
=/alu1/a	010000010001110000000000000000000000000		0100000010											10110111011	111100010000	101011010
	001111110001000000000000000000000000000		0011111100			1	1							10110111011	11100010000	101011010
	1010000	0000000000	1000000	10100	00	1100000	0000110	11100010000	0001010		0000101	001000	0010001	0010010	0100001	0101001
Ales Now	1900 ns		200			400		600			Unininini Ons	100	liiiiiiiiii 0 ns	111111111	liiiiiiiii 10 ns	1400 ns
Gel Qursor 1	242 ns	15	200	247	ns	400	/115	000	/115		UTIS	100	10 115	120	0.115	1400 ms
Cursor 1	212113			212	113											

Figure 27: FPU Subtraction



Floating-point numbers

Figure 28: Floating-point Subtraction

Operands																
/main_tb/A														[10110111011	111100010000	101011010
/main_tb/B	0100001111111010001000000000000	000000000	001111100	001111110	0100	0111	10110111011	111100010000	101011010							
Cin																
>tb/Cin	0									_						
Opcode —																
opcode	1100000	0000000	1000000	1010000	[1100	00	0000110		0001010		0000101	0010000	0010001	0010010	0100001	0101001
Result ———																
tb/OUT	11000011000101100001001100110011	000000000	010000010	010000010	1100	0110	011110001	011110001	001100000	001100000	010010001	000000000	100000000	000000000	010010001	<u>0000000000000000000000000000000000000</u>
status —																
ut/ALU	St0															
ut/FPU	St1															
/ARITH	St0															
ut/CMP	St0															
ut/LOG	St0					_										
/SHIFT	St0					_										
ut/Add	St0															
uut/Sub	St0															
uut/Mul	St1 🔶															
b/uut/A	101111101001100110011001100110010	000000000	010000001	010000010	1011	11101	00110000011	111001011101	110101010					10110111011	111100010000	101011010
b/uut/B	010000111111101000100000000000000000000	000000000	001111100	001111110	0100	0111	10110111011	111100010000	101011010							
opcode	1100000	0000000	1000000	1010000	1100	00	0000110		0001010		0000101	0010000	0010001	0010010	0100001	0101001
uut/Cin	St0															
ut/OUT	11000011000101100001001100110011	000000000	010000010	010000010	1100	0110	011110001	011110001	001100000	001100000	010010001	000000000	100000000	000000000	010010001	0000000000
U_ALU	000000000000000000000000000000000000000	000000000000000000	00000000000000000	00000000			011110001	011110001	001100000	001100000	010010001	000000000	100000000	000000000	010010001	000000000000000000000000000000000000000
U_FPU	11000011000101100001001100110011	000000000	010000010	010000010	1100	0110	000000000000000000000000000000000000000	000000000000000000000000000000000000000	00000000							
•/alu1/a	101111101001100110011001100110010	000000000	010000001	010000010	1011	11101	00110000011	111001011101	110101010					10110111011	111100010000	01011010
/alu1/b	010000111111101000100000000000000000000	000000000	001111100	001111110	0100	0111	10110111011	111100010000	101011010							
oncode	1100000	000000	1000000	1010000	11100	00	0000110		0001010		0000101	0010000	0010001	0010010	0100001	0101001
	1900 ns	ns	200) ns		400) ns		0 ns	80	0 ns	100	10 ns	120	10 ns	
Cursor 1	338 ns				33											

Figure 28: FPU Multiplication



Figure 30: Floating-point multiplicat

5. Conclusion

While working on this project we sought to gain practical experience and knowledge in the field of computer engineering and computer organization. Tools such as ModelSim Simulator, used in this project offer a unique way of design testing and verification by enabling signal.

REFERENCES

- 1) Stallings, William, *Computer Organization & Architecture: Designing for Performance,* 7th *ed.* Upper Saddle River, NJ: Prentice-Hall, 2006.
- 2) Computer Architecture ALU Design: Division and Floating Point, <u>http://www.ann.ece.ufl.edu/courses/eel4713_12spr/slides/Lec8-division.pdf</u>