

19ECE383
VLSI Design Lab
System Level Design Report

Project Title : Design a 4-bit ALU with ripple-carry adder-based arithmetic and logic unit for XOR, NOT.

Name of the Students:

- 1) Abburi Sai Keerthi –BL.EN.U4ECE22002
- 2) B. Asritha Venkata Naga Hasini-BL.EN.U4ECE22010
- 3) Bommisetty Lakshmi Sowmya-BL.EN.U4ECE22011
- 4) Megha Elango-BL.EN.U4ECE22035

ECE-A



AMRITA SCHOOL OF ENGINEERING

Faculty In-charge: Mr. Vignesh ,Mr. Swaminadhan

Date of Evaluation: 29/04/2025

Academic Year: 2024-2025 (Even Semester)

Aim:

To design a 4-bit ALU with ripple-carry adder-based arithmetic and logic unit for XOR, NOT.

1)Abstract :

The Arithmetic and Logic Unit (ALU) is a key component of digital systems, responsible for performing arithmetic and logical operations. This project involves the design and implementation of a 4-bit ALU capable of addition, subtraction, XOR, and NOT operations. The ALU accepts two 4-bit binary inputs and a 2-bit control signal to select the desired operation. Arithmetic operations are executed using a 4-bit Ripple-Carry Adder (RCA), while logic operations are performed using basic combinational logic gates. A multiplexer determines the final output based on the control signal.

The ALU design was first implemented in Verilog HDL and functionally verified through simulation in Xilinx Vivado, ensuring correct logical behavior. Subsequently, the project was realized at the transistor level using Cadence Virtuoso, employing CMOS logic gates for physical design. Transistor-level simulation verified timing, functionality, and layout accuracy. Additionally, analysis of power consumption, delay, and area was conducted to assess the design's efficiency for ASIC fabrication.

Successful implementation at both HDL and transistor levels demonstrates the ALU's correctness, modularity, and feasibility for integration into larger digital systems, such as microprocessors and embedded applications.

Tools Used:

Cadence Virtuoso , Xilinx Vivado 2020.1, Verilog and VHDL, Vivado Simulator, Power Analyzer and Timing Analyzer .

Theory:

2)Introduction

- **Importance of ALU in Digital Systems**

The Arithmetic and Logic Unit (ALU) is a vital component in every digital processing system, such as microprocessors, microcontrollers, and digital signal processors (DSPs). It is responsible for executing a variety of arithmetic operations (such as addition and subtraction) and logic operations (such as XOR and NOT), enabling complex computational tasks to be carried out efficiently.

The performance and efficiency of an ALU directly impact the speed, power consumption, and functionality of the overall system. Hence, the design and optimization of an ALU are crucial in digital system development.

- **Motivation Behind the Project**

The primary motivation behind this project is to understand and implement the fundamental principles of digital arithmetic and logic operations both at the behavioral (HDL) and physical (transistor-level) design levels. By implementing the ALU using Verilog HDL, a high-level functional model was created, enabling simulation and verification of logic at the digital abstraction level. To gain deeper insight into how digital circuits are built at the hardware level, the same ALU was also designed and implemented using CMOS logic gates in Cadence Virtuoso, representing real-world transistor-level behavior. This two-level approach helps bridge the gap between functional digital design and physical hardware realization, which is critical for advanced system design and VLSI (Very Large-Scale Integration) understanding.

- **Overview of the Two Parts: Verilog-based and CMOS-based ALU Designs**

This project consists of two major parts:

- The 4-bit ALU was described using Verilog HDL, implementing basic operations like Addition, Subtraction, XOR, and NOT. A Ripple-Carry Adder (RCA) was used for arithmetic operations, and combinational logic was used for logical functions. The design was simulated and verified using Xilinx Vivado to ensure correct functional behavior.
- To explore the hardware-level realization, the same 4-bit ALU was implemented at the transistor level using Cadence Virtuoso. CMOS logic gates were designed for the required operations, and a schematic-level simulation was performed. This approach allowed analysis of circuit parameters like propagation delay, power consumption, and area, essential for ASIC or VLSI designs.

By combining HDL-based simulation and CMOS-level hardware design, this project offers a complete understanding of digital ALU design from abstraction to physical realization.

The design uses simple combinational logic and a ripple-carry adder, making it hardware-efficient and easy to implement on FPGA or basic digital circuits.

3) Verilog HDL Based ALU Design(Vivado):

- 4-bit ALU: Supports arithmetic and logic operations with 4-bit input/output precision.
- Operations: Includes Ripple-Carry Adder (ADD), Subtractor (SUB), XOR, and NOT functions.

A ripple-carry adder consists of a series of full adders. Each full adder adds corresponding bits of two input numbers along with a carry-in from the previous stage.

- The carry output of each full adder is connected to the carry input of the next full adder.
- Because each carry must propagate through each adder, it is called a ripple-carry adder.
- 4-bit RCA: Consists of 4 full adders cascaded to add two 4-bit numbers.

- **Subtractor Implementation:**

A subtractor can be built using the same ripple-carry adder structure by inverting the subtrahend (2's complement) and setting the initial carry-in to 1.

- **4-bit Subtractor:**

For a 4-bit subtractor, the second input (B) is inverted, and the first carry-in is set to 1, effectively performing $A - B$ using 2's complement arithmetic.

Apart from arithmetic operations, an ALU must perform logic operations:

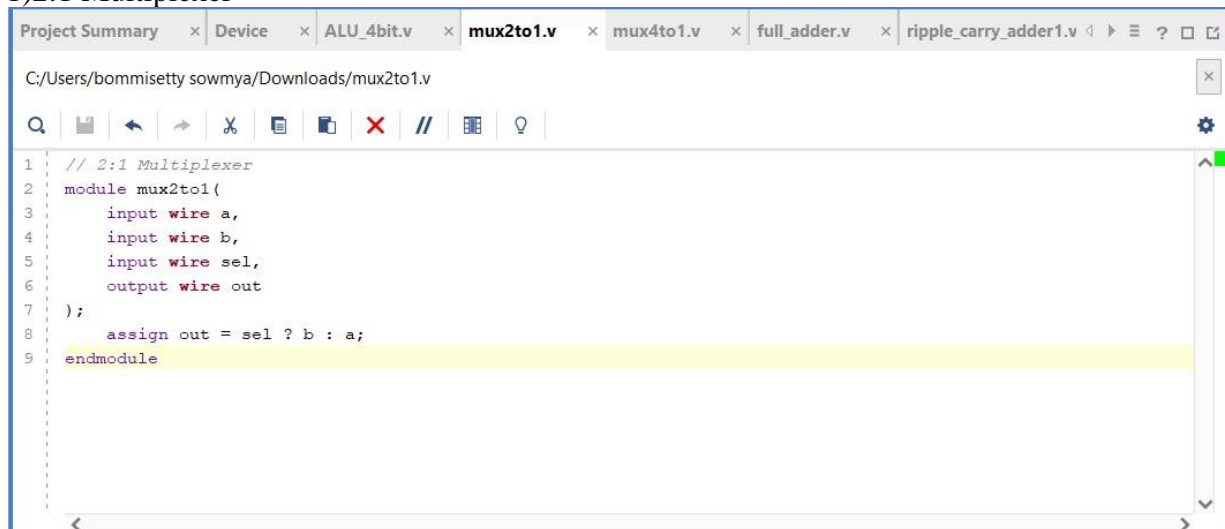
- XOR Operation (Exclusive-OR):
 - Output is HIGH (1) only when the inputs are different.
 - Boolean Expression:
$$Y = A \oplus B$$
- NOT Operation (Complement):
 - Inverts each bit of the input.
 - Boolean Expression:
$$Y = \sim A$$

Design Methodology:

- **Functional Block Diagram:** The ALU design includes interconnected modules for arithmetic (RCA adder/subtractor) and logic operations (XOR, NOT), controlled by a multiplexer-based selection unit.

Verilog Code Snippets:

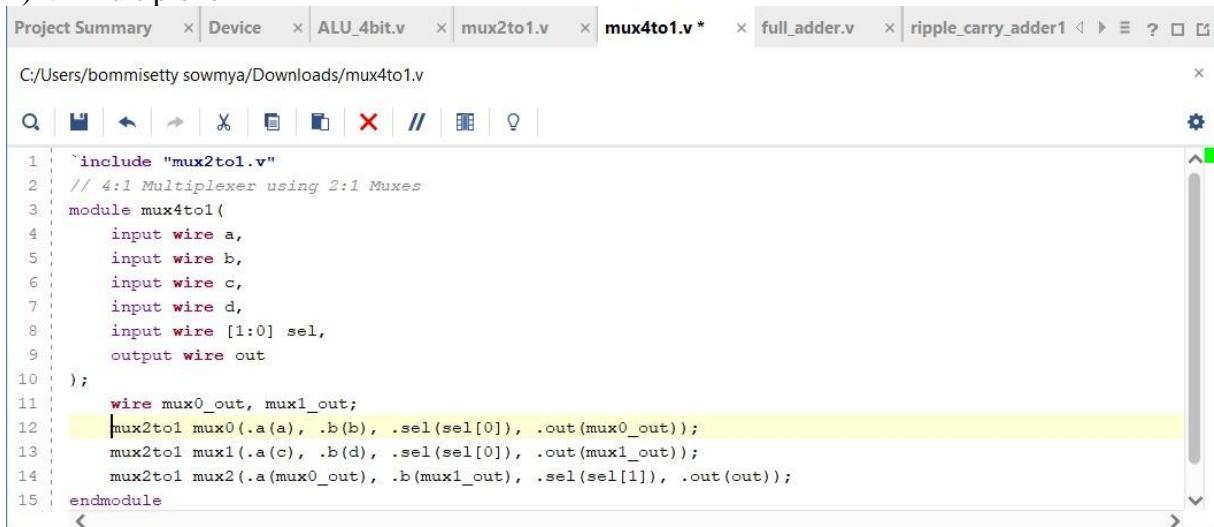
1)2:1 Multiplexer



The screenshot shows a Verilog code editor with a tab labeled 'mux2to1.v'. The code defines a 2:1 multiplexer module. The file path is 'C:/Users/bommisetty sowmya/Downloads/mux2to1.v'. The code is as follows:

```
1 // 2:1 Multiplexer
2 module mux2to1(
3     input wire a,
4     input wire b,
5     input wire sel,
6     output wire out
7 );
8     assign out = sel ? b : a;
9 endmodule
```

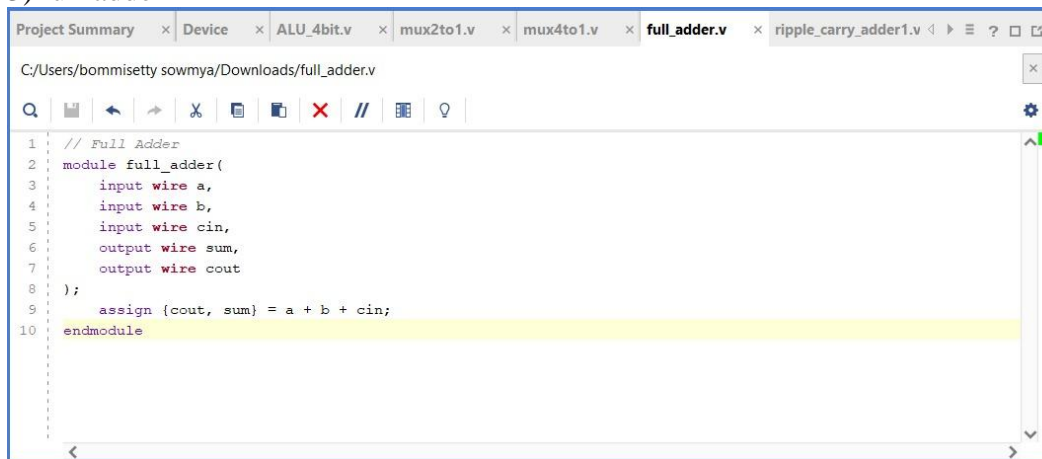
2)4:1 Multiplexer



The screenshot shows a Verilog code editor with a tab labeled 'mux4to1.v *'. The code defines a 4:1 multiplexer module using two 2:1 multiplexers. The file path is 'C:/Users/bommisetty sowmya/Downloads/mux4to1.v'. The code is as follows:

```
1 `include "mux2to1.v"
2 // 4:1 Multiplexer using 2:1 Muxes
3 module mux4to1(
4     input wire a,
5     input wire b,
6     input wire c,
7     input wire d,
8     input wire [1:0] sel,
9     output wire out
10 );
11     wire mux0_out, mux1_out;
12     mux2to1 mux0(.a(a), .b(b), .sel(sel[0]), .out(mux0_out));
13     mux2to1 mux1(.a(c), .b(d), .sel(sel[0]), .out(mux1_out));
14     mux2to1 mux2(.a(mux0_out), .b(mux1_out), .sel(sel[1]), .out(out));
15 endmodule
```

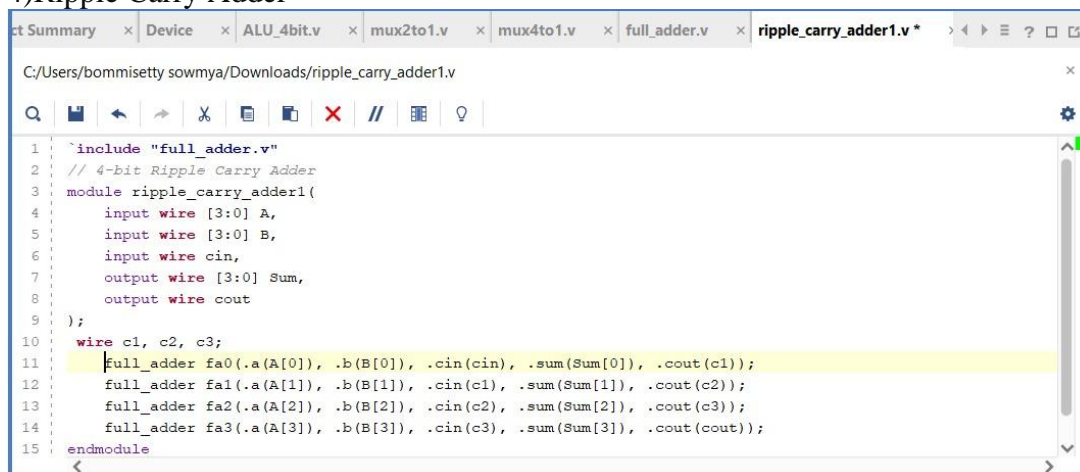
3) Full adder



The screenshot shows a Verilog code editor with the file name `full_adder.v`. The code defines a module `full_adder` with three inputs: `a`, `b`, and `cin`, and two outputs: `sum` and `cout`. The logic is implemented using an `assign` statement to calculate the sum and carry-out.

```
1 // Full Adder
2 module full_adder(
3     input wire a,
4     input wire b,
5     input wire cin,
6     output wire sum,
7     output wire cout
8 );
9     assign {cout, sum} = a + b + cin;
10 endmodule
```

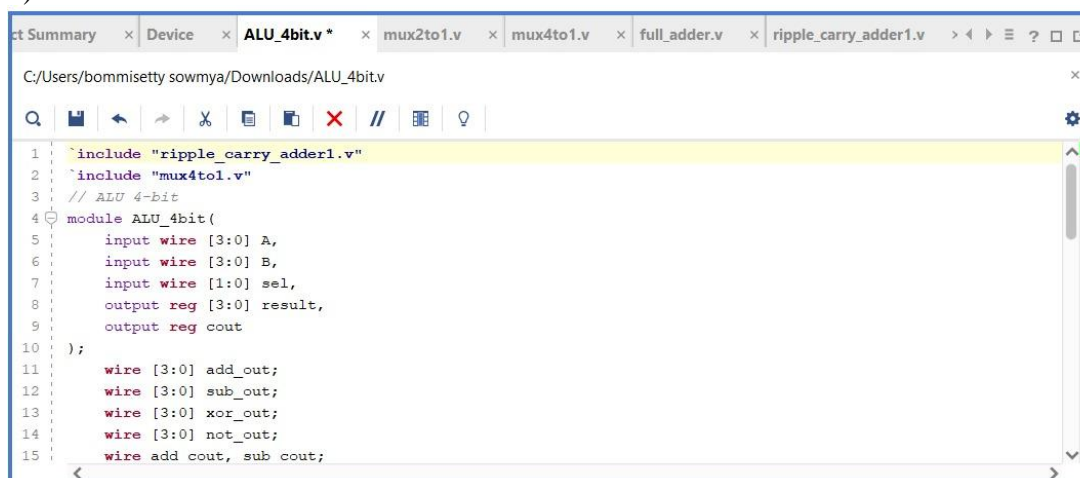
4) Ripple Carry Adder



The screenshot shows a Verilog code editor with the file name `ripple_carry_adder1.v`. The code defines a module `ripple_carry_adder1` that takes two 4-bit inputs `A` and `B`, and an input carry `cin`. It produces a 4-bit sum `Sum` and a carry-out `cout`. The implementation uses four instances of the `full_adder` module to perform the ripple-carry addition.

```
1 `include "full_adder.v"
2 // 4-bit Ripple Carry Adder
3 module ripple_carry_adder1(
4     input wire [3:0] A,
5     input wire [3:0] B,
6     input wire cin,
7     output wire [3:0] Sum,
8     output wire cout
9 );
10 wire c1, c2, c3;
11 full_adder fa0(.a(A[0]), .b(B[0]), .cin(cin), .sum(Sum[0]), .cout(c1));
12 full_adder fa1(.a(A[1]), .b(B[1]), .cin(c1), .sum(Sum[1]), .cout(c2));
13 full_adder fa2(.a(A[2]), .b(B[2]), .cin(c2), .sum(Sum[2]), .cout(c3));
14 full_adder fa3(.a(A[3]), .b(B[3]), .cin(c3), .sum(Sum[3]), .cout(cout));
15 endmodule
```

5) 4-bit ALU



The screenshot shows a Verilog code editor with the file name `ALU_4bit.v`. The code defines a module `ALU_4bit` that takes two 4-bit inputs `A` and `B`, and a 2-bit select input `sel`. It produces a 4-bit result `result` and a carry-out `cout`. The implementation includes logic for addition, subtraction, XOR, and NOT operations, along with carry propagation.

```
1 `include "ripple_carry_adder1.v"
2 `include "mux4to1.v"
3 // ALU 4-bit
4 module ALU_4bit(
5     input wire [3:0] A,
6     input wire [3:0] B,
7     input wire [1:0] sel,
8     output reg [3:0] result,
9     output reg cout
10 );
11 wire [3:0] add_out;
12 wire [3:0] sub_out;
13 wire [3:0] xor_out;
14 wire [3:0] not_out;
15 wire add_cout, sub_cout;
```

```
ct Summary x Device x ALU_4bit.v * x mux2to1.v x mux4to1.v x full_adder.v x ripple_carry_adder1.v > < > < ? □
```

C:/Users/bommisetty sowmya/Downloads/ALU_4bit.v

```
16 wire cin_add, cin_sub;
17 // Internally define cin based on operation
18 assign cin_add = 1'b0; // For addition
19 assign cin_sub = 1'b1; // For subtraction
20 // Ripple carry adder for Addition
21 ripple_carry_adder1 adder(
22     .A(A),
23     .B(B),
24     .cin(cin_add),
25     .Sum(add_out),
26     .cout(add_cout)
27 );
28 // Ripple carry adder for Subtraction (A + (~B) + 1)
29 ripple_carry_adder1 subtractor(
30     .A(A),
31     .B(~B),
32     .cin(cin_sub),
33     .Sum(sub_out),
34     .cout(sub_cout)
35 );
36 // XOR
37 assign xor_out = A ^ B;
38 // NOT
39 assign not_out = ~A;
40 // Output selection
41 always @(*) begin
42     case(sel)
43         2'b00: begin result = add_out; cout = add_cout; end // Addition
44         2'b01: begin result = sub_out; cout = sub_cout; end // Subtraction
45         2'b10: begin result = xor_out; cout = 1'b0; end // XOR
46         2'b11: begin result = not_out; cout = 1'b0; end // NOT
47         default: begin result = 4'b0000; cout = 1'b0; end
48     endcase
49 end
50 endmodule
```

ct Summary x Device x ALU_4bit.v * x mux2to1.v x mux4to1.v x full_adder.v x ripple_carry_adder1.v > < > < ? □

C:/Users/bommisetty sowmya/Downloads/ALU_4bit.v

```
29 ripple_carry_adder1 subtractor(
30     .A(A),
31     .B(~B),
32     .cin(cin_sub),
33     .Sum(sub_out),
34     .cout(sub_cout)
35 );
36 // XOR
37 assign xor_out = A ^ B;
38 // NOT
39 assign not_out = ~A;
40 // Output selection
41 always @(*) begin
42     case(sel)
43         2'b00: begin result = add_out; cout = add_cout; end // Addition
44         2'b01: begin result = sub_out; cout = sub_cout; end // Subtraction
45         2'b10: begin result = xor_out; cout = 1'b0; end // XOR
46         2'b11: begin result = not_out; cout = 1'b0; end // NOT
47         default: begin result = 4'b0000; cout = 1'b0; end
48     endcase
49 end
50 endmodule
```

Test Bench:

```
Project Summary x tb_ALU_4bit.v * x
```

C:/Users/bommisetty sowmya/Downloads/tb_ALU_4bit.v

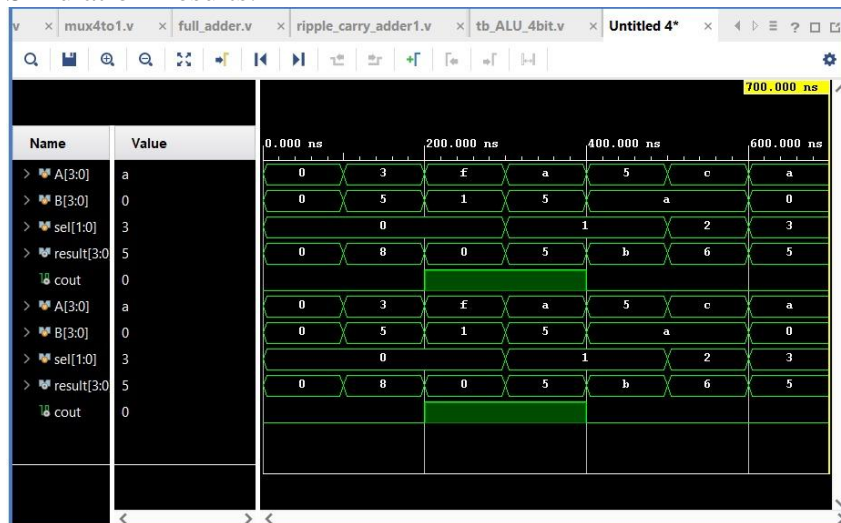
```
1 `timescale 1ns / 1ps
2 module tb_ALU_4bit;
3     // Inputs
4     reg [3:0] A;
5     reg [3:0] B;
6     reg [1:0] sel;
7     // Outputs
8     wire [3:0] result;
9     wire cout;
10    // Instantiate the Unit Under Test (UUT)
11    ALU_4bit uut (
12        .A(A),
13        .B(B),
14        .sel(sel),
15        .result(result),
16        .cout(cout)
17    );
```

```
Project Summary x tb_ALU_4bit.v x
C:/Users/bommisetty sowmya/Downloads/tb_ALU_4bit.v
18 initial begin
19     // Initialize Inputs
20     A = 4'b0000;
21     B = 4'b0000;
22     sel = 2'b00;
23     // Wait 100 ns for global reset to finish
24     #100;
25     // Add stimulus here
26     // Test case 1: Addition (3 + 5)
27     A = 4'b0011;
28     B = 4'b0101;
29     sel = 2'b00;
30     #100;
31     // Test case 2: Addition with carry (15 + 1)
32     A = 4'b1111;
33     B = 4'b0001;
34     sel = 2'b00;
```

```
Project Summary x tb_ALU_4bit.v x
C:/Users/bommisetty sowmya/Downloads/tb_ALU_4bit.v
35     #100;
36     // Test case 3: Subtraction (10 - 5)
37     A = 4'b1010;
38     B = 4'b0101;
39     sel = 2'b01;
40     #100;
41     // Test case 4: Subtraction (5 - 10)
42     A = 4'b0101;
43     B = 4'b1010;
44     sel = 2'b01;
45     #100;
46     // Test case 5: XOR
47     A = 4'b1100;
48     B = 4'b1010;
49     sel = 2'b10;
50     #100;
51     // Test case 6: NOT
```

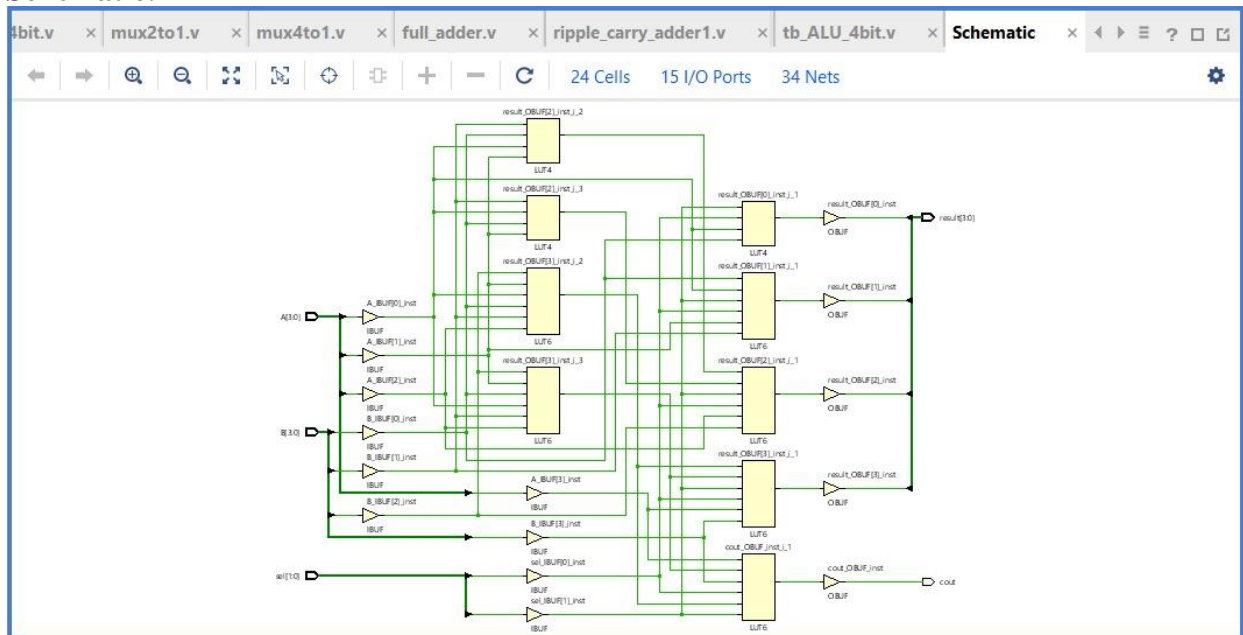
```
52     A = 4'b1010;
53     B = 4'b0000; // B doesn't matter for NOT
54     sel = 2'b11;
55     #100;
56     // Finish simulation
57     $finish;
58 end
59 // Monitor the results
60 initial begin
61     $monitor("Time=%0t: A=%4b B=%4b sel=%2b | Result=%4b Cout=%1b",
62         $time, A, B, sel, result, cout);
63 end
64 endmodule
```


Simulation Results:



The simulation verifies the 4-bit ALU performing addition, subtraction, XOR, and NOT operations based on selection inputs. The result and cout outputs match expected values, confirming correct ALU functionality.

Schematic:

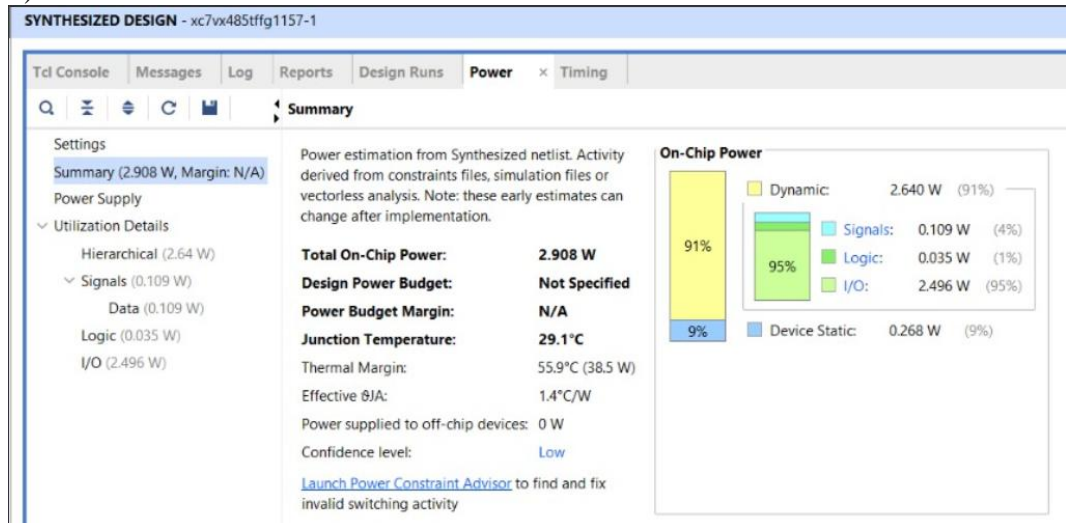


- **Synthesis Results:**

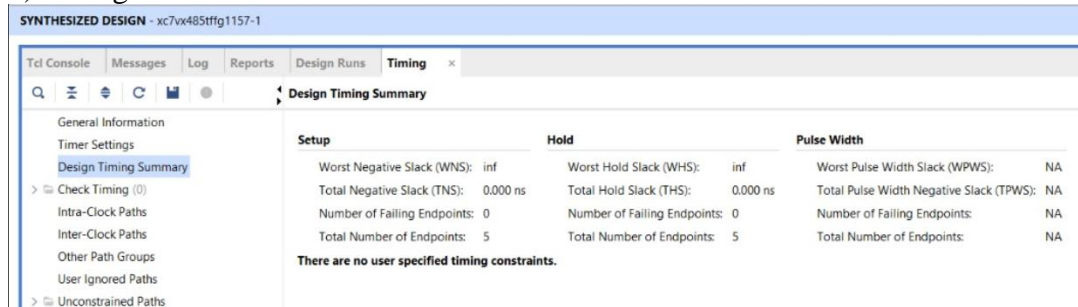
- 1)Area Utilisation



- 2)Power



- 3)Timing



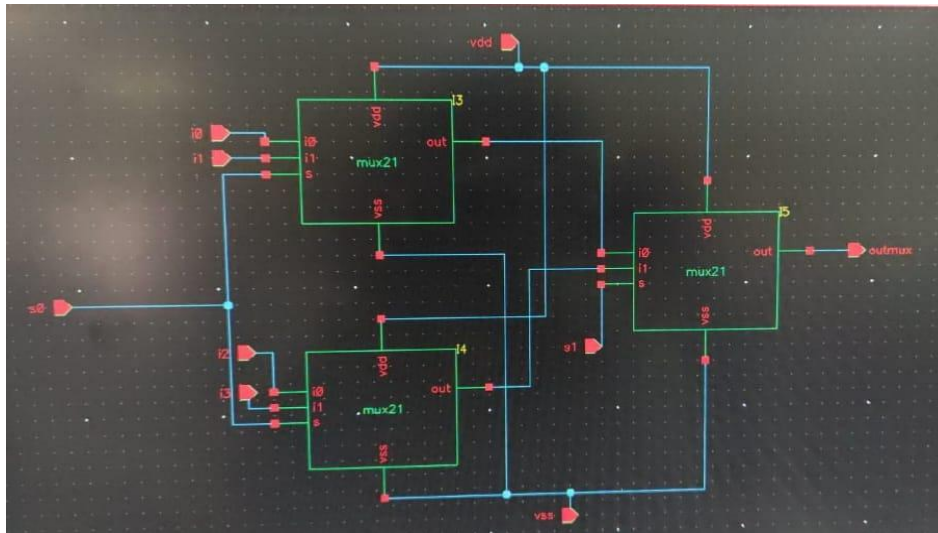
- **CMOS Circuit Level ALU Design(Cadence):**

Design Approach:

- Full Adder and MUX using Pass-Transistor Logic in Cadence Virtuoso through transient analysis.

Simulation Results:

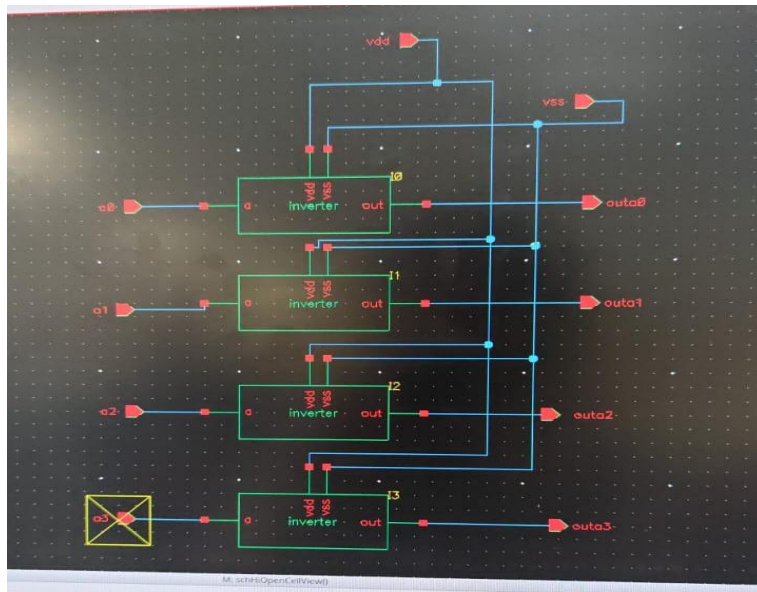
4:1 MUX (using 2:1 MUX)



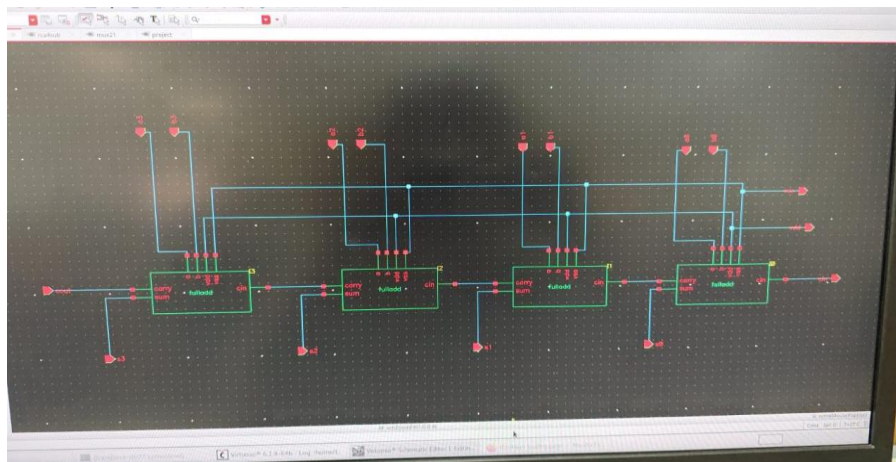
XOR gate



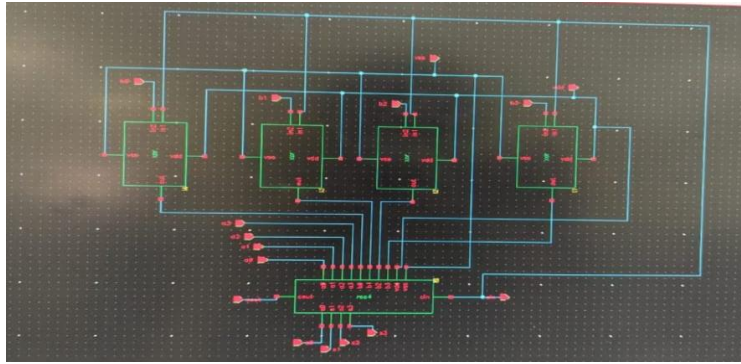
Inverter



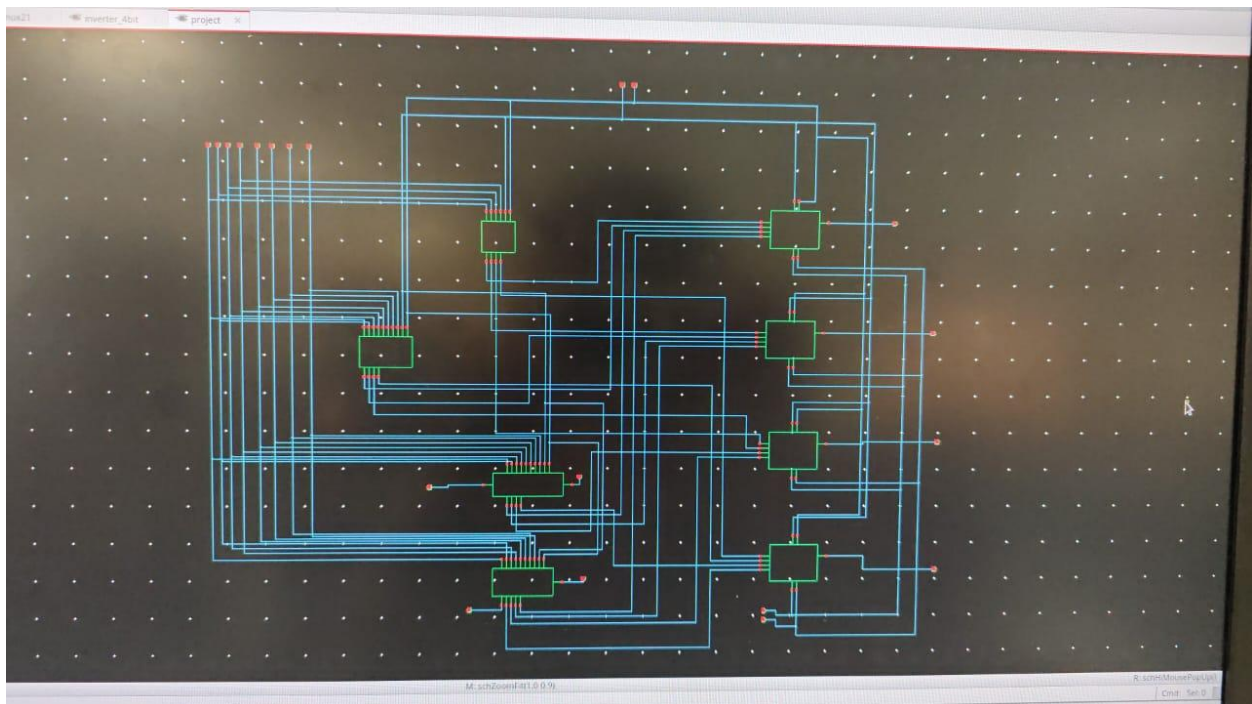
Ripple Carry Adder

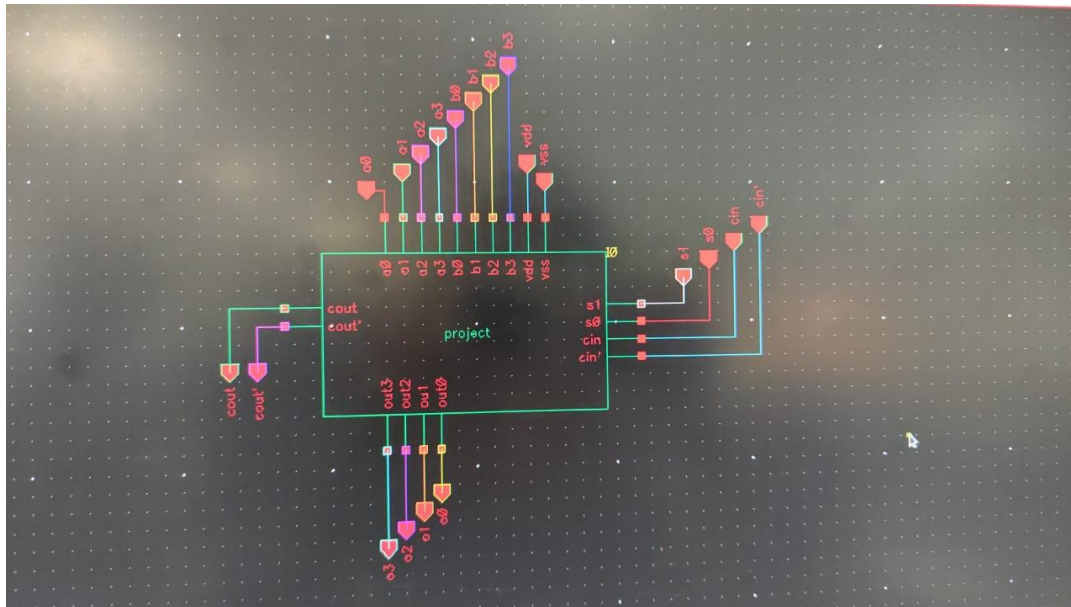


Subtractor

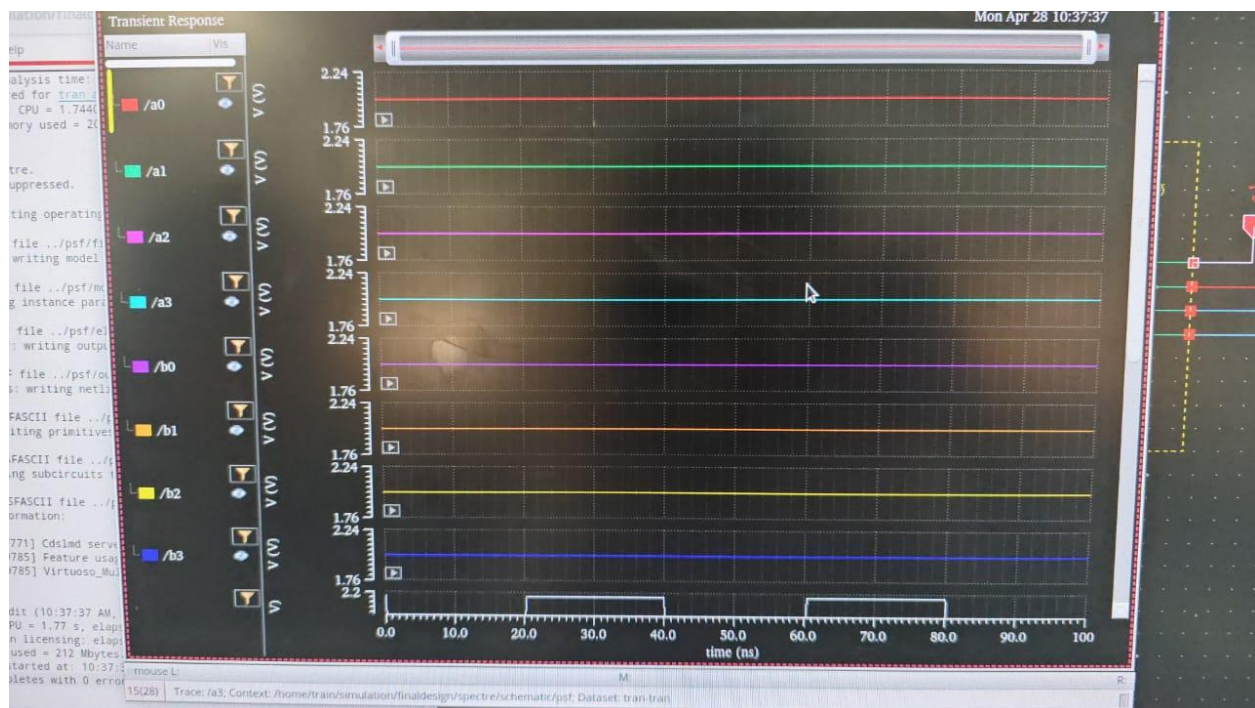


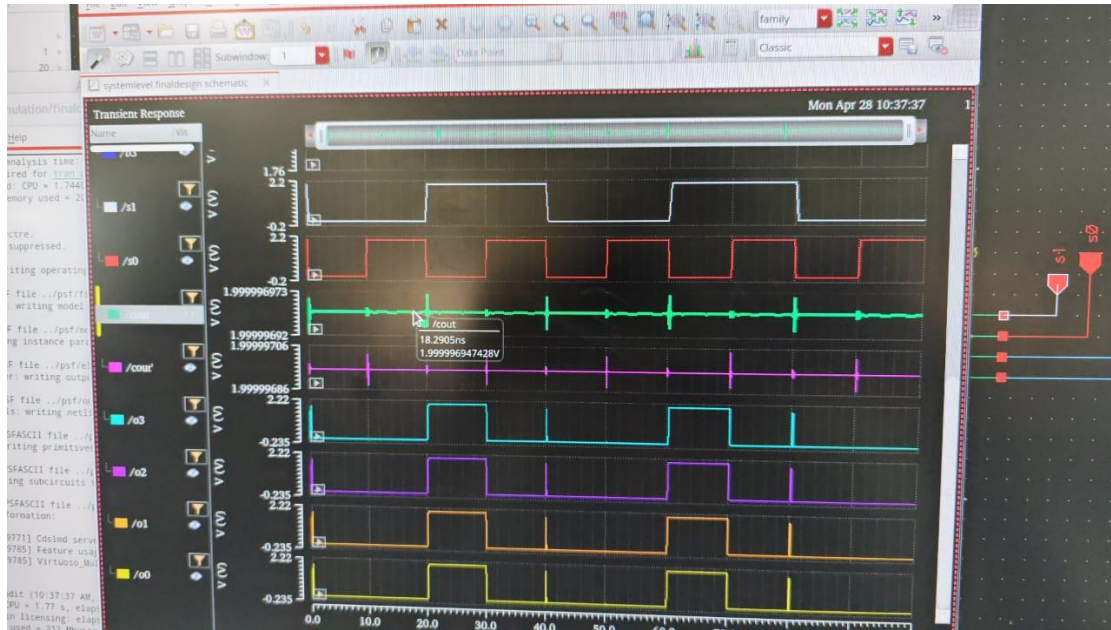
4-bit ALU:





Simulation Result :





Comparative Analysis:

Aspect	CMOS Design (Transistor-Level)	Verilog Design (RTL-Level)
Abstraction Level	Very low (detailed control at transistor level)	High (describes logic behavior, not hardware details)
Design Time	Slow and complex (manual, detailed work)	Fast and efficient (automatic synthesis possible)
Optimization	Highly optimized for power, area, and speed	Limited optimization, relies on synthesis tools
Debugging & Verification	Difficult (waveforms, manual checking)	Easier (testbenches, simulation at functional level)

Challenges faced:

- Simulation issues were resolved by adjusting tool settings.
- Wrong outputs were corrected by carefully checking and fixing circuit connections.
- Tool errors were solved by reinstalling necessary files and properly setting up paths.

Conclusion:

- Successfully designed and simulated 4-bit ALU blocks (XOR, NOT, RCA, Subtractor, MUX) at both transistor (CMOS) and RTL (Verilog) levels.
- Verified correct functionality through transient analysis in CMOS and behavioral simulation in Verilog.
- Understood trade-offs between transistor-level accuracy (CMOS) and faster design time (Verilog).
- Future work can extend the ALU to include pipelining, more operations, and power optimization.

