Mini-CPU Project-Midterm Review

Implementation of an 8-Bit RISC CPU (45 nm CMOS Technology)

Keerthi Patil, Dept. of Electrical Engineering, Wright State University, patil.115@wright.edu

---

Abstract - *This midterm report presents the design and verification of the main functional blocks of an 8-bit RISC processor using Cadence Virtuoso with the FreePDK45 (45 nm) CMOS technology. Each module - Arithmetic Logic Unit (ALU), Accumulator, Decoder, Program Counter, Instruction Memory, Multiplexer- was implemented using basic logic gates constructed at the transistor level with CMOS technology. All circuits were simulated to verify functional behavior. Power consumption and propagation delay were measured for each block to evaluate their performance. The simulation results confirmed correct functionality, with outputs observed under the applied clock and reset conditions. This phase successfully completes the design and simulation of all processor blocks, forming the foundation for the upcoming integration of the processor.*

Keywords - Reduced Instruction Set Computer (RISC) Processor, Cadence Virtuoso, FreePDK45, CMOS Transistor-Level Design, Gate-Level Implementation, Spectre Simulator, Power and Delay Analysis.

I. INTRODUCTION

The design of processors based on the Reduced Instruction Set Computer (RISC) architecture plays a key role in understanding digital system design and computer organization. This project focuses on developing an 8-bit RISC processor in Cadence Virtuoso using FreePDK45 (45 nm) CMOS technology. The processor comprises modules such as the Arithmetic Logic Unit (ALU), Accumulator, Decoder, Program Counter, Instruction Memory, Multiplexer,Memory, Ring Oscillator. Each logic gate was designed from the transistor level using CMOS logic, and verified gates were hierarchically combined to form higher-level functional blocks following a bottom-up design flow. Transient simulations verified logical behavior, timing, and signal transitions under clock and reset conditions. The goal of this phase is to complete transistor-level implementation, analyze propagation delay and power consumption, and ensure proper functionality of each module. The verified designs will serve as the foundation for full processor integration in later stages.

II. IMPLEMENTATION METHODOLOGY

A. Design Flow

Basic logic gates (NAND2/3, NOR2/3, XOR2, AND2/3) were designed at the transistor level using *FreePDK45 (45 nm)* CMOS technology in *Cadence Virtuoso*. Verified gates were reused to build higher-level modules like ALU, Accumulator, Decoder, and Memory. This hierarchical design ensured reusability and consistency across the processor.

B. Symbol Creation and Hierarchy

Each verified schematic was converted into a symbol in *Virtuoso* to enable hierarchical integration. Pin labeling simplified block-level connections and maintained clarity during processor assembly.

C.Simulation and Performance Analysis

Simulations were performed using the *Spectre* simulator to verify logic transitions, timing, and output behavior. Average and instantaneous power were calculated for each block to evaluate efficiency and delay performance.

D. Design Verification

All modules were individually verified to confirm logical correctness and response to clock and input signals. Verified blocks were prepared for top-level processor

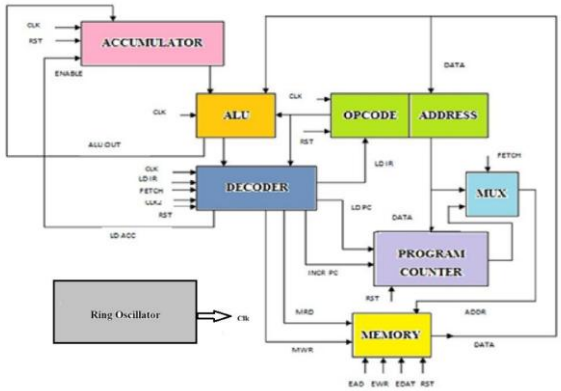


Fig. 1. Block-level architecture of the 8-bit RISC processor.

Functional Blocks

Block	Operation	Input Width (bits)	Output Width (bits)
Ring Oscillator	Generates the internal clock for synchronized processor operation.	-	1 (Clock Signal)
Instruction Register (IR)	Holds the fetched instruction and separates opcode and operand.	8	8 (3-bit opcode + 5-bit address)
Accumulator	Stores intermediate and final results for arithmetic and logic operations.	8	8
Arithmetic Logic Unit (ALU)	Executes arithmetic and logic functions.	8	8
Memory (SRAM)	Stores both data and program instructions with read/write/reset control.	8 (Data) / 5 (Address)	8 (Data Out)
Program Counter (PC)	Tracks and updates the next instruction address during execution.	5	5
Decoder & Control Unit	Decodes opcodes and produces control signals for all modules.	3	8 (Control Signals)
Multiplexer (MUX)	Selects data or address paths for controlled signal routing.	Variable (2, 4, 8)	1
I/O Buffer	Handles data exchange between the CPU and external devices.	8	8

Processor Operation Cycle

The processor operates through a four-stage instruction cycle- **Fetch, Decode, Execute,** and **PC Update-** which ensures proper synchronization among the PC, Decoder, ALU, Accumulator, and other modules, enabling smooth and continuous instruction flow during processor operation.

Fetch: The Program Counter (PC) sends the instruction address to Memory and the fetched instruction is loaded into the Instruction Register (IR).

Decode: The Decoder decodes the instruction and generates control signals to coordinate data flow among the ALU, Accumulator, and Program Counter.

Execute: The ALU performs the defined arithmetic or logic operation, and the resulting output is stored in the Accumulator while the PC prepares for the next instruction.

PC Update: After execution, the Program Counter increments to the next instruction.

III. PROCESSOR ARCHITECTURE

A. BASIC GATES

The processor was developed using basic logic gates such as NAND, NOR, XOR, and AND, each designed at the transistor level using 45 nm CMOS technology in *Cadence Virtuoso*. Verified gates were converted into symbols to enable hierarchical design and reuse in higher-level modules. These symbols were then interconnected to construct major functional blocks such as the ALU, Accumulator, Decoder, Program Counter, Memory, Multiplexer, and Clock Generator, forming the complete 8-bit RISC processor architecture.

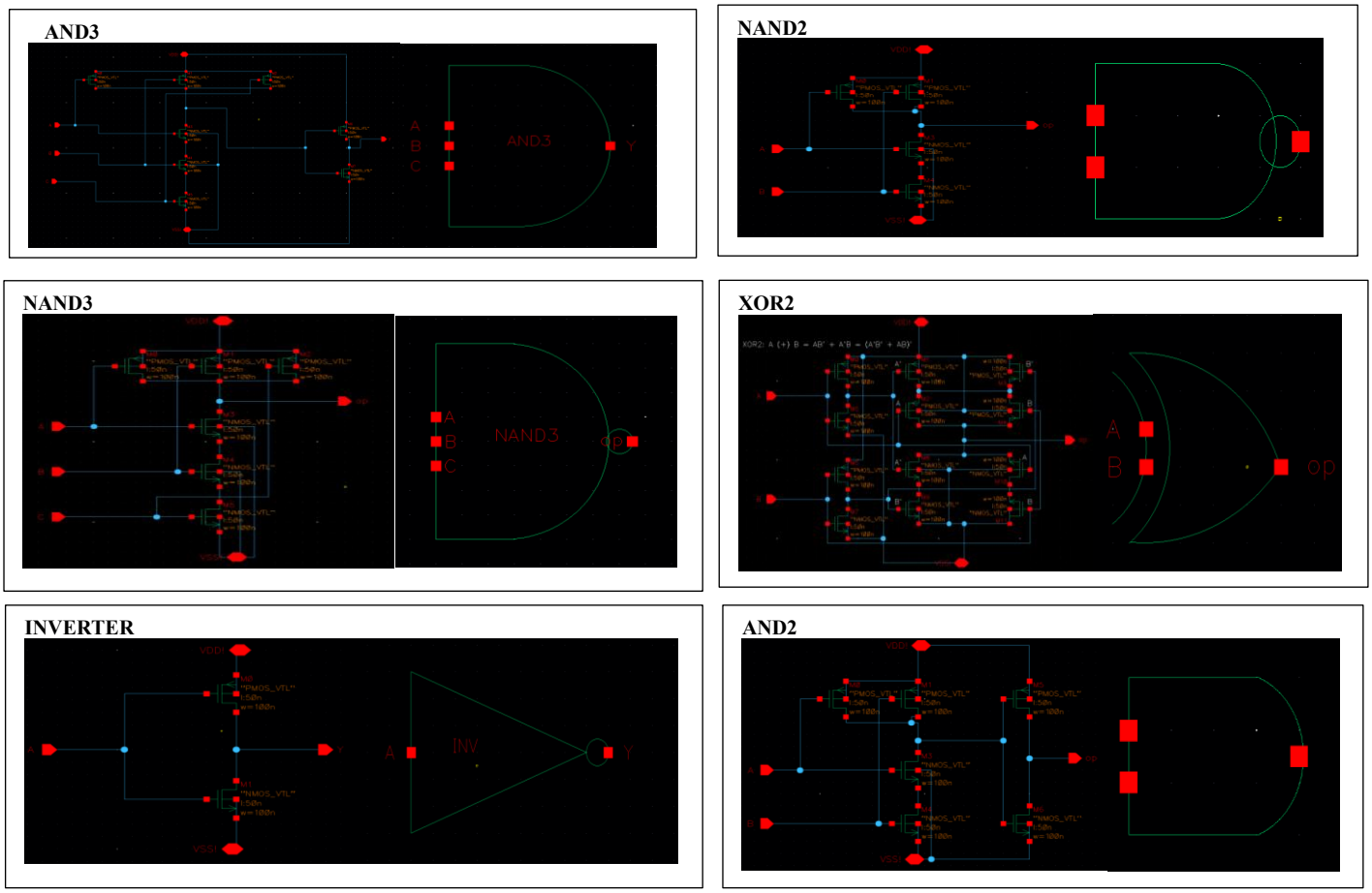


Figure2 : Basic gates designed at transistor level and their symbols

B. Program Counter (PC)

Functionality: The Program Counter (PC) is a sequential circuit responsible for maintaining the correct order of instruction execution in the processor. It stores the address of the current instruction and increases its value by one with each clock pulse to point to the next instruction in memory, ensuring smooth and continuous program flow. During branch or jump instructions, the PC loads a new address from the instruction itself, enabling non-sequential execution when required. In the integrated processor, the PC responds to control signals such as LD PC (load), INCR PC (increment), and RST (reset) generated by the Decoder, which help coordinate instruction sequencing and timing across all modules.

Design and Implementation: The PC was implemented using five Master–Slave T Flip-Flops connected synchronously to form a 5-bit binary up counter. The initial version built with simple T Flip-Flops showed level-sensitive behavior, acting like a latch during the active clock phase. To correct this, the Master–Slave configuration was adopted to achieve proper edge-triggered operation. Using the derived excitation equations and truth table, the circuit's functionality was verified. The final design increments sequentially from 00000 (0) to 11111(31) on each clock pulse, providing stable and accurate address transitions that ensure reliable instruction sequencing in the processor.

1-bit T Flip-Flop : The T Flip-Flop (TFF) is a sequential circuit that changes its output state on every active clock edge when T = 1, and holds its previous value when T = 0. In the initial design using NAND3 and NAND2 gates, the circuit did not perform as expected and behaved more like a T latch rather than an edge-triggered flip-flop. The simulation waveform showed that the output varied multiple times during the high clock period, indicating level-sensitive behavior. This happened because the input and feedback paths were both active when the clock was high, which caused unintended transitions. To resolve this issue, the circuit was redesigned as a Master–Slave T Flip-Flop, ensuring edge-triggered operation where the output toggles only once per clock pulse and remains stable during the inactive phase.

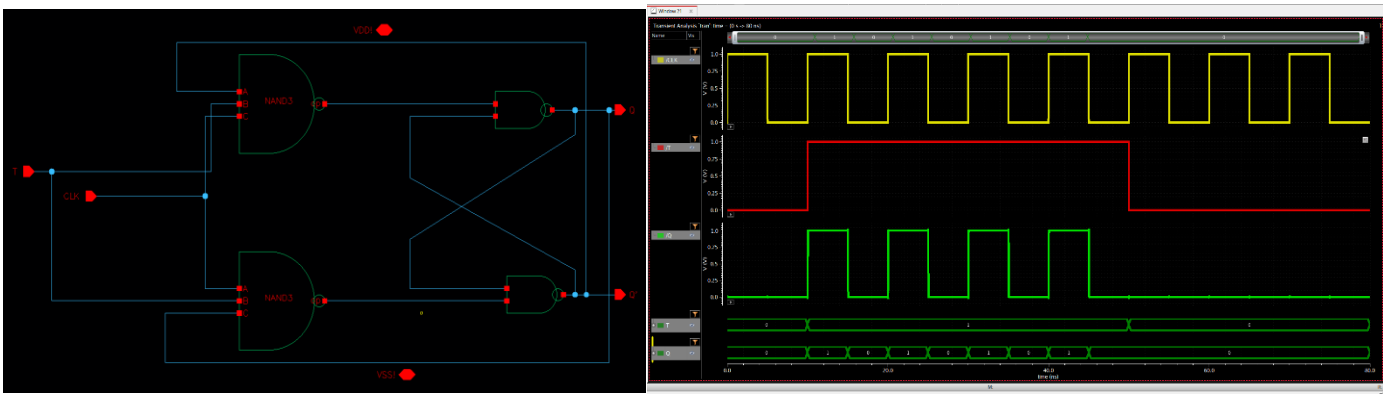
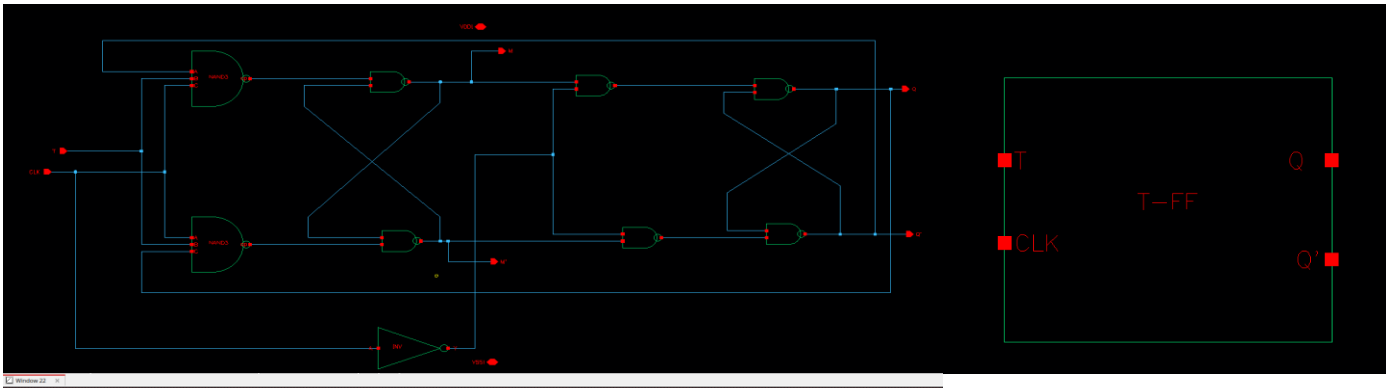
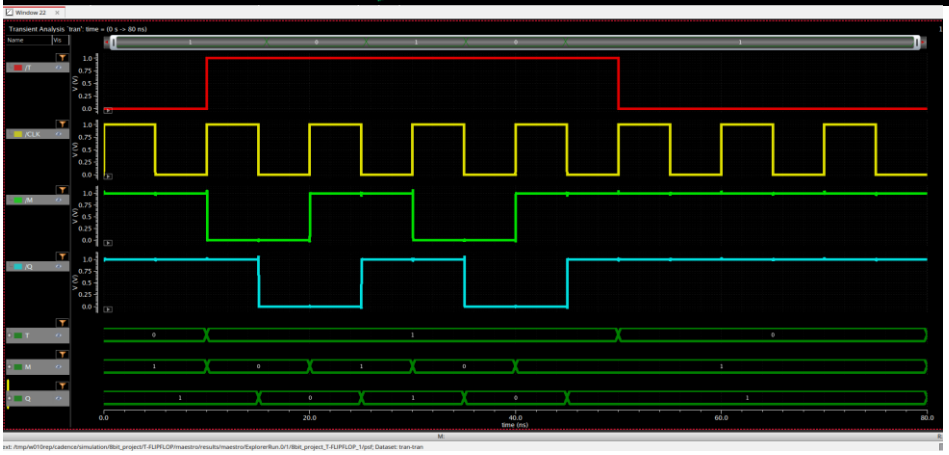


Figure3 : Circuit diagram and simulation waveform of the 1-bit T Flip-Flop

1-bit Master–Slave T Flip-Flop: The 1-bit Master–Slave T Flip-Flop was designed to correct the level-sensitive behavior observed in the basic T Flip-Flop. In the initial design, when the clock was high, the output changed continuously with the input, functioning like a latch instead of an edge-triggered flip-flop. To overcome this, two latches were connected in series - forming a master and a slave stage - operated by complementary clock signals. The master latch is active when the clock is high, and the slave latch is active when the clock is low, ensuring that only one latch operates at a time. This configuration enables edge-triggered operation, allowing the output to toggle only once per clock pulse. The Master–Slave TFF provides stable and synchronized transitions, making it suitable for use in counters and other sequential circuit designs.

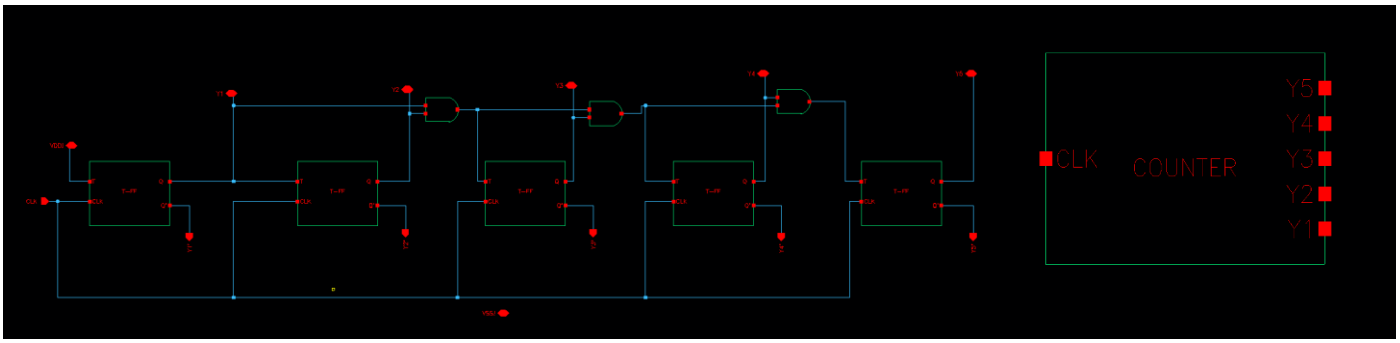




T (Input)	Q_n (Present State)	Q_{n+1} (Next State)	Operation
0	0	0	No Change
0	1	1	No Change
1	0	1	Toggle
1	1	0	Toggle

Figure4 : Design and verification of the 1-bit Master–Slave T Flip-Flop showing the truth table, symbol, circuit diagram, and simulation waveform.

5-bit Program Counter (PC): The 5-bit Program Counter (PC) was implemented as a synchronous binary counter using five Master–Slave T Flip-Flops. All flip-flops share a common clock input, allowing them to toggle simultaneously on each rising clock edge, ensuring accurate and stable transitions. The T inputs for each stage were derived using excitation equations to achieve the correct counting sequence. Simulation results showed the counter incrementing sequentially from 00000(0) to 11111(31), confirming proper synchronous operation. This design exhibits reliable edge-triggered behavior and serves as a key element in maintaining instruction sequencing within the processor.



Q4 Q3 Q2 Q1 Q0 (n)	Q4 Q3 Q2 Q1 Q0 (n+1)	T4 T3 T2 T1 T0
00000	00001	00001
00001	00010	00011
00010	00011	00001
00011	00100	00111
00100	00101	00001
00101	00110	00011
00110	00111	00001
00111	01000	01111
01000	01001	00001
01001	01010	00011
01010	01011	00001
01011	01100	00111
01100	01101	00001
01101	01110	00011
01110	01111	00001
01111	10000	11111

$T_0 = 1$

$T_1 = Q_0$

$T_2 = Q_1 \cdot Q_0$

$T_3 = Q_2 \cdot Q_1 \cdot Q_0$

$T_4 = Q_3 \cdot Q_2 \cdot Q_1 \cdot Q_0$

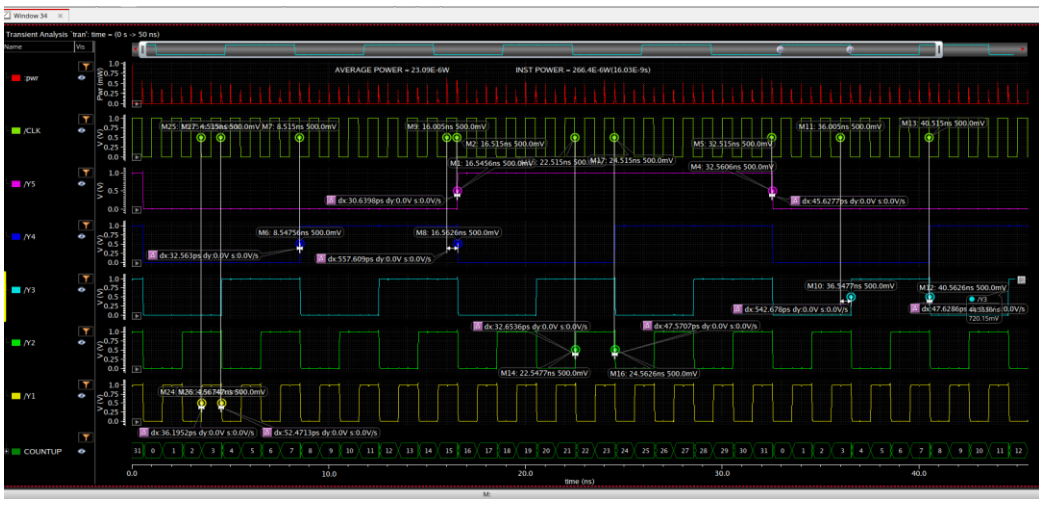


Figure 5: Design and verification of the 5-bit synchronous Program Counter implemented using Master–Slave T Flip-Flops, showing the circuit diagram, symbol, truth table, Boolean equations, and simulation waveform verifying correct sequential counting from 00000 to 11111.

Simulation and Validation: The transient simulation verified correct synchronous counting of the 5-bit Program Counter from 00000 to 11111, with all flip-flops toggling simultaneously on each rising clock edge. The design showed stable operation with an average power of 23.09 µW and peak power of 266.4 µW.

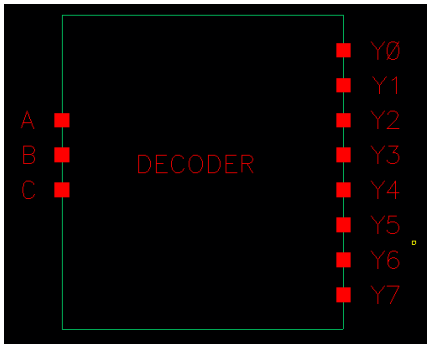
C. Decoder

Functionality: The Decoder is a combinational circuit that serves as the control unit of the processor. In the current stage, a 3-to-8 line decoder has been implemented to demonstrate the basic decoding concept. It converts a 3-bit binary input into eight unique output lines, ensuring that only one output is active for each input combination. This one-hot output behavior is essential for generating distinct control signals corresponding to different input patterns.In the next stage, this design will be developed into a complete processor control unit capable of interpreting instruction opcodes from the Instruction Register (IR) and generating control signals such as LD PC, INCR PC, MRD, MWR, and LD ACC. These signals will manage the coordinated operation of the ALU, Program Counter, Accumulator, and Memory, ensuring proper sequencing and synchronization during instruction execution.

*Design and Implementation:*The Decoder was implemented as a 3-to-8 line decoder using CMOS logic gates in Cadence Virtuoso with FreePDK45 (45 nm) technology. It consists of three binary inputs (A, B, C) that represent the instruction bits, and eight output lines (Y0–Y7) that correspond to decoded control signals.Each output is generated by combining the true and complemented forms of the inputs using AND gates, ensuring one-hot output behavior- only one output line is high for a given input combination.

A	B	C	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

$$Y_0 = \overline{A}\,\overline{B}\,\overline{C}$$
$$Y_1 = \overline{A}\,\overline{B}\,C$$
$$Y_2 = \overline{A}\,B\,\overline{C}$$
$$Y_3 = \overline{A}\,B\,C$$
$$Y_4 = A\,\overline{B}\,\overline{C}$$
$$Y_5 = A\,\overline{B}\,C$$
$$Y_6 = A\,B\,\overline{C}$$
$$Y_7 = A\,B\,C$$



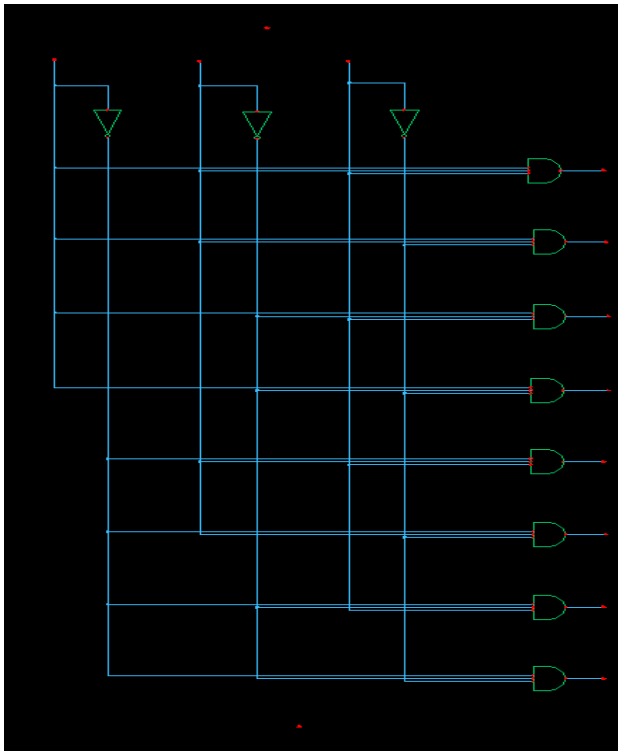


Figure 6: Gate-level design of the 3-to-8 line decoder showing the truth table, Boolean equations, symbol, and circuit diagram.

Simulation and Validation: Waveform analysis verified the correct operation of the designed decoder. Each change in the input pattern produced a corresponding activation of one output line, ensuring distinct responses for all eight input states. The average power is 127.8 µW, and the peak instantaneous power is 550.2 µW.

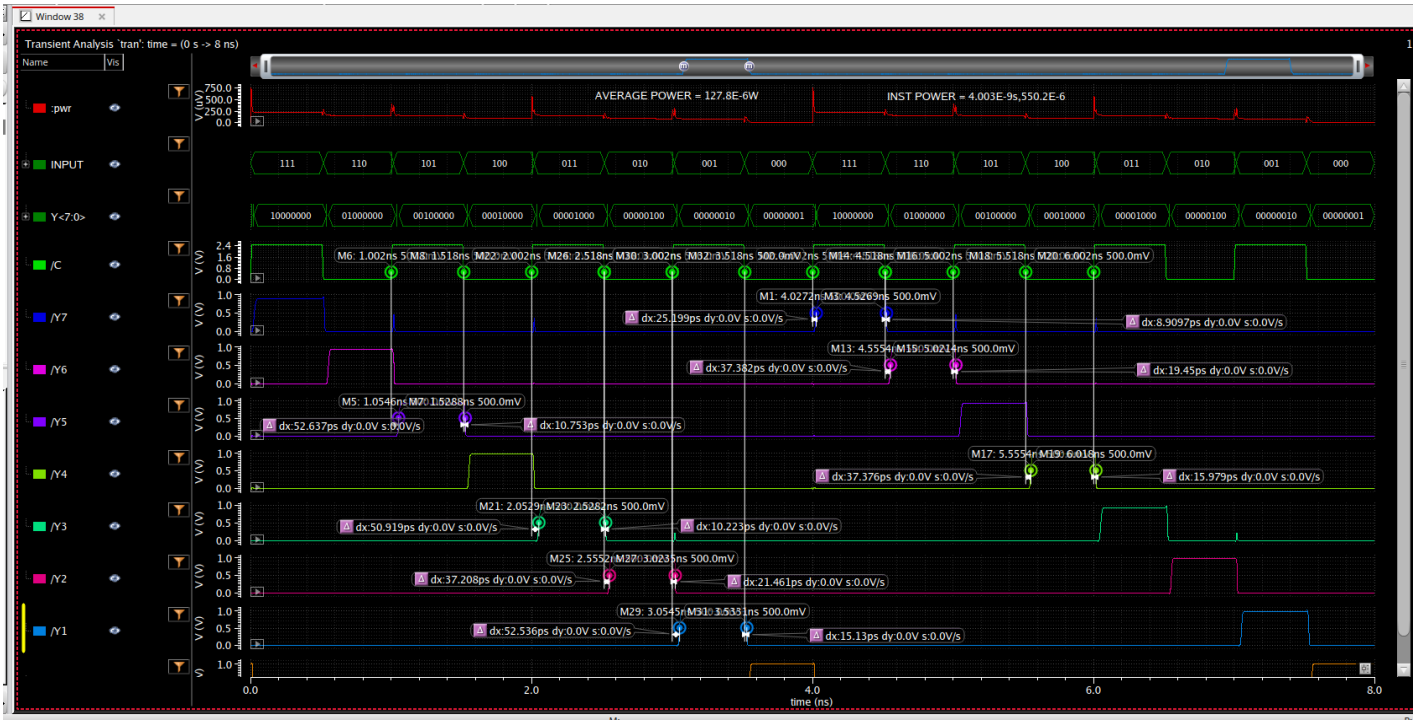


Figure 7: Simulation waveform of the 3-to-8 line decoder showing correct output transitions for all input combinations.

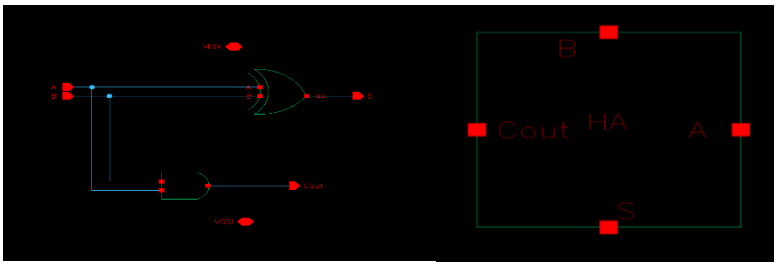
D. Arithmetic Logic Unit (ALU)

Functionality: The Arithmetic Logic Unit (ALU) is a combinational circuit that performs both arithmetic and logical operations based on the opcode coming from the instruction memory. It takes two 8-bit inputs - the accumulator output (A) and memory data (B) - along with a 3-bit operation select code (OP[2:0]) from the decoder. In this stage, we have built and verified individual blocks for addition, subtraction, 4×4 multiplication, binary increment, binary decrement, AND, XOR, inversion, shift left, and shift right. In the next stage, all these blocks will be combined to make a single ALU that can perform every operation through one control input in the processor. **Opcode Mapping : ADD/SUB=000, MUL=001, INC=010, DEC=011, AND=100, NOT=101, XOR=110, SHL/SHR=111**

Design and Implementation: The ALU was designed in Cadence Virtuoso using basic logic gates and verified sub-blocks. Each operation was created and checked separately. The adder and subtractor were made using full adders with an XOR control for subtraction. The 4×4 multiplier was designed using partial product generation and binary addition. The increment and decrement circuits were built using a 4-bit adder with fixed inputs of '1'(VDD) and '−1'(VSS). Logical operations such as AND, XOR, and inversion were made using simple gate-level designs. The shift left and shift right functions were implemented using a barrel shifter. All these blocks were simulated and verified individually. In the next stage, they will be connected together under one control input to form the complete ALU of the processor.

Sub-Blocks and Simulation and Validation:
(i) 8-bit Adder / Subtractor- *Performs 8-bit binary addition and subtraction operations using a single circuit controlled by a mode signal.*

Half adder: The Half Adder adds two single-bit binary inputs and gives Sum and Carry as outputs. The Sum is produced using an XOR gate, and the Carry is generated using an AND gate. It was designed and verified in Cadence Virtuoso (45 nm).



A	B	Sum (A ⊕ B)	Carry (A · B)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figure 8: Half Adder Truth table, circuit diagram and symbol.

1 bit Full Adder: The Full Adder adds three one-bit inputs - A, B, and Cin and produces a Sum and a Carry output. It was designed using compound gates, where the Sum is given by A ⊕ B ⊕ Cin, and the Carry is AB + B·Cin + A·Cin. In the lab, both standard-gate and compound-gate versions of the Full Adder were implemented and compared. The compound gate design showed lower power consumption and reduced delay, making it more efficient and suitable for high-speed circuit applications. Therefore, the compound gate-based Full Adder was used in this design to improve the overall performance of the ALU.

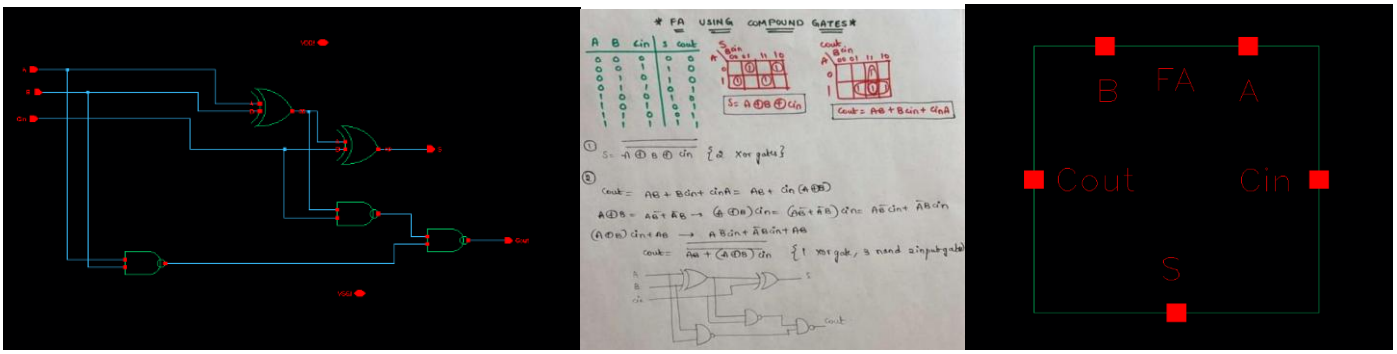


Figure 9:1-bit Full Adder Truth table, circuit diagram and symbol.

8- Bit Adder/Subtractor: The 8-bit Adder/Subtractor was built by connecting eight Full Adders in series to form a ripple-carry structure. Each Full Adder adds one bit from the two inputs A[7:0] and B[7:0] along with the carry from the previous stage. To perform subtraction, the second input B is passed through XOR gates controlled by a mode signal. When the mode is 0, the circuit works as an adder, and when the mode is 1, it performs subtraction by taking the 2's complement of B. The circuit was designed in Cadence Virtuoso (45 nm) using compound-gate Full Adders, which helped reduce power and delay compared to standard gate designs. The simulation confirmed correct results for both addition and subtraction operations. The average power is 60.71 µW, and the peak instantaneous power is 1.313 mW.

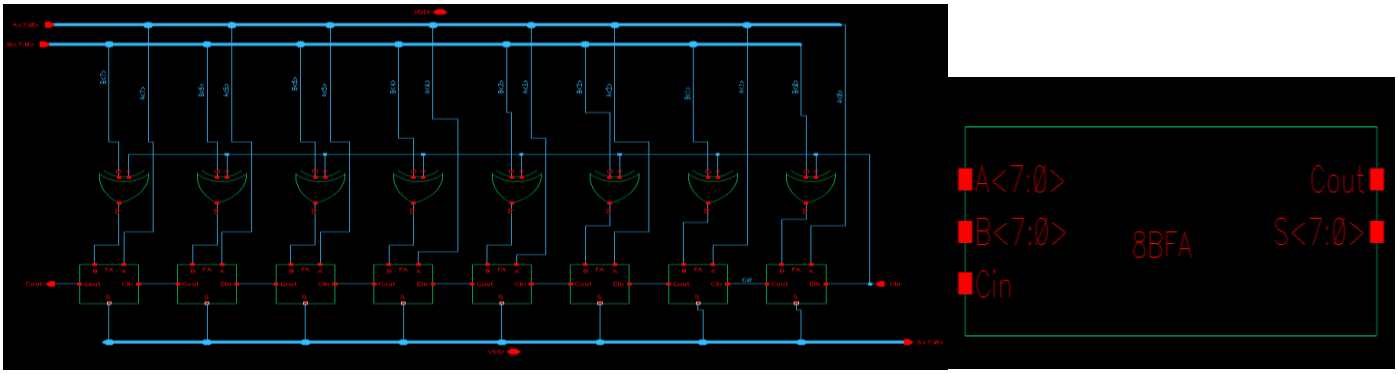


Figure 10: 8-Bit full Adder/Subtractor Circuit diagram and symbol

Addition operation using 8-bit adder/subtractor (Cin = 0).
Case 1: A = 11111111_2 B = 11111101_2 (Cin = 0) → Sum = 11111100_2, Carry = 1
Case 2: A = 00101111_2 B = 11110100_2 (Cin = 0) → Sum = 00100011_2, Carry = 1

Subtraction operation using 8-bit adder/subtractor (Cin = 1).
Case 1: A = 00110011_2, B = 11001100_2 → 2's complement of B = 00110100_2 → A + 2's complement of B = 01100111_2
Case 2: A = 11000011_2, B = 11000001_2 → 2's complement of B = 00111111_2 → A + 2's complement of B = 00000010_2

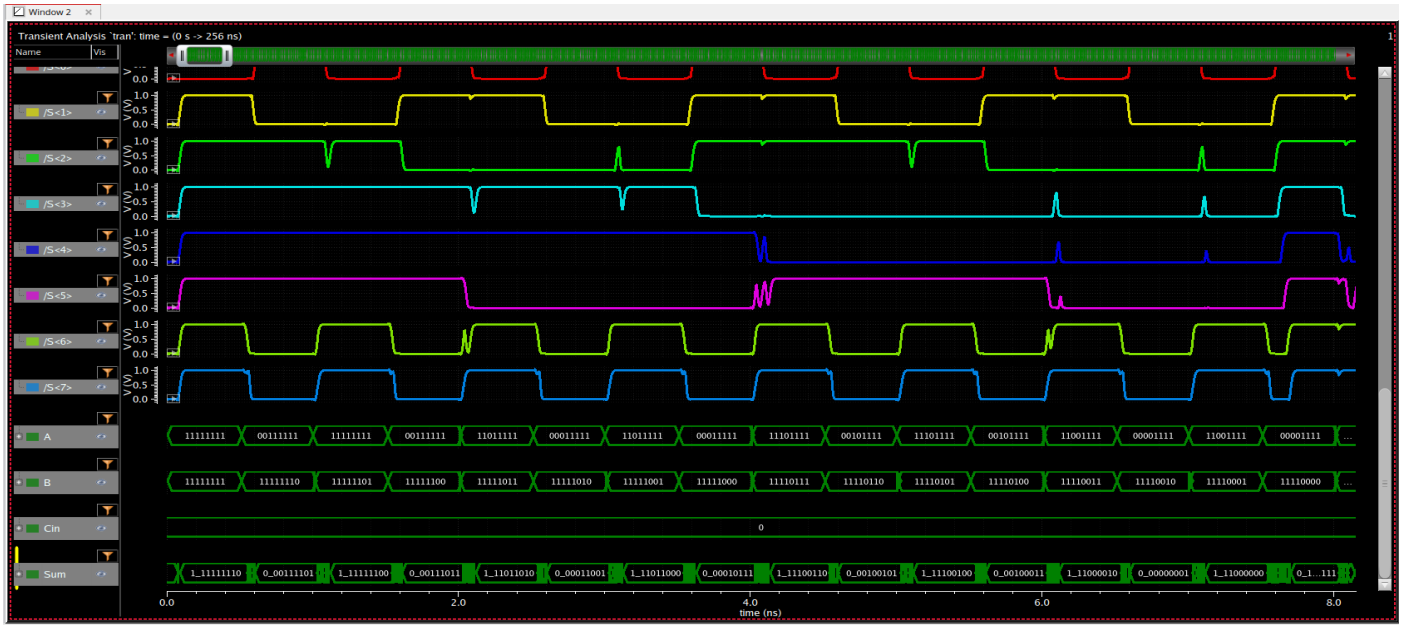


Figure 11: Waveform showing Addition operation(cin=0)

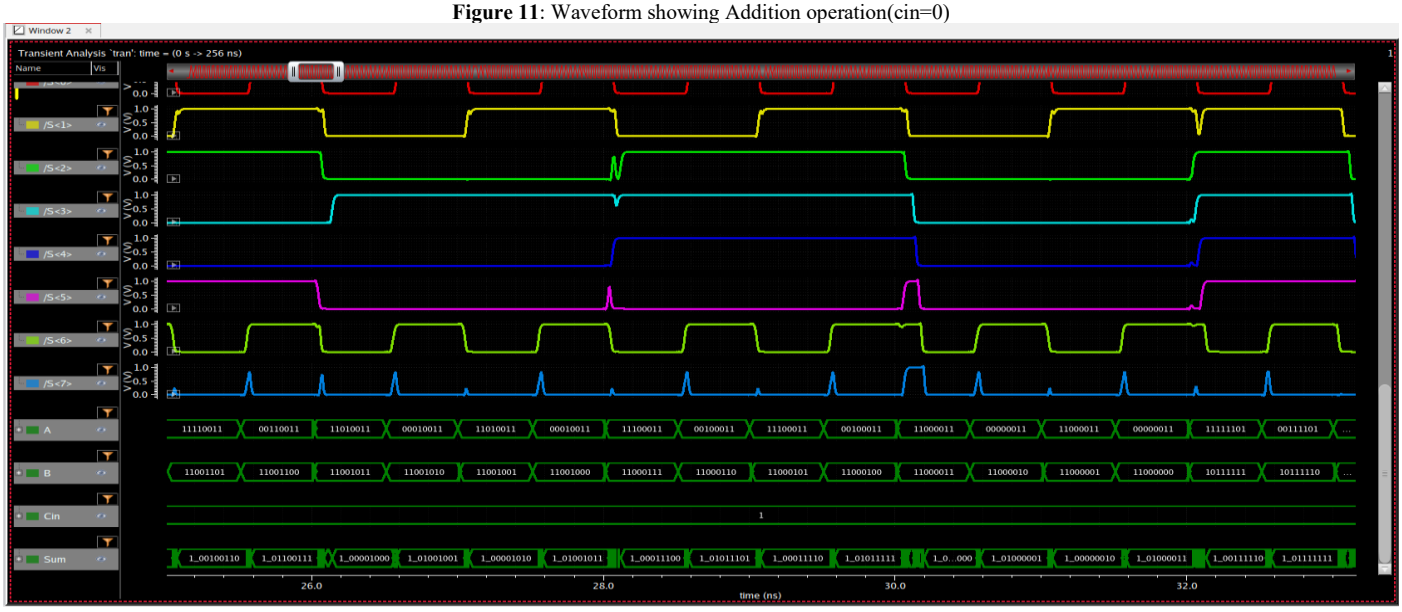


Figure 12: Waveform showing Subtraction operation (cin=1)

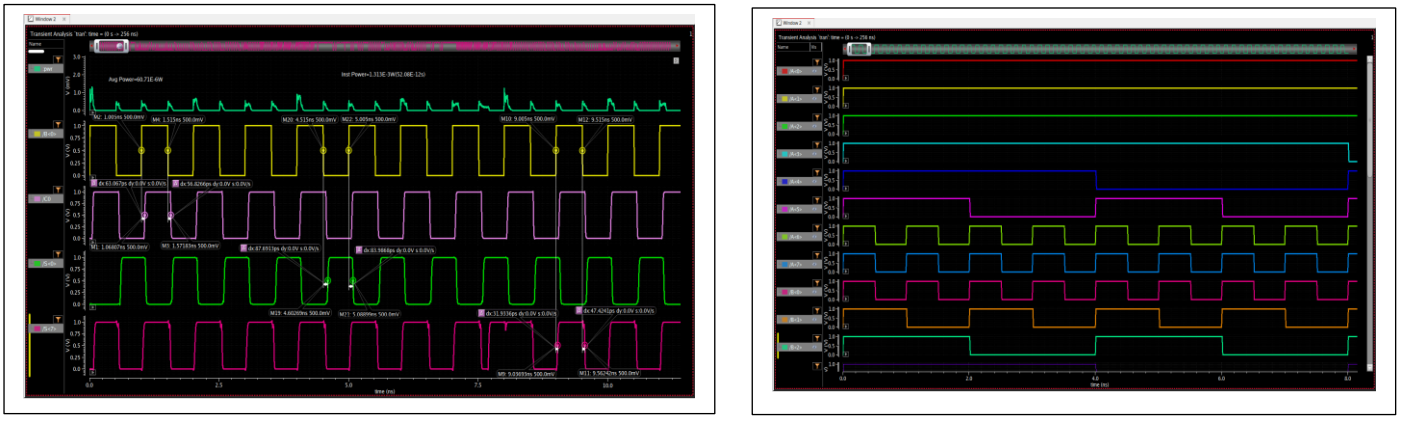
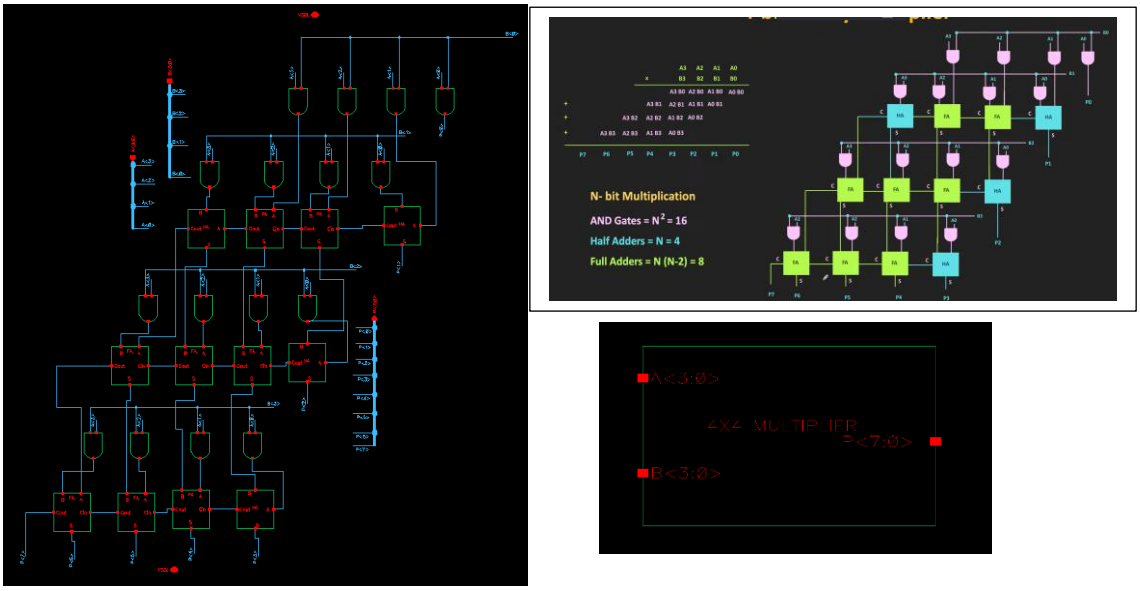


Figure 13: 8-Bit Adder/Subtractor Power and Delay

(ii) 4×4 Multiplier- *Performs 4×4 binary multiplication for two 4-bit binary inputs.*

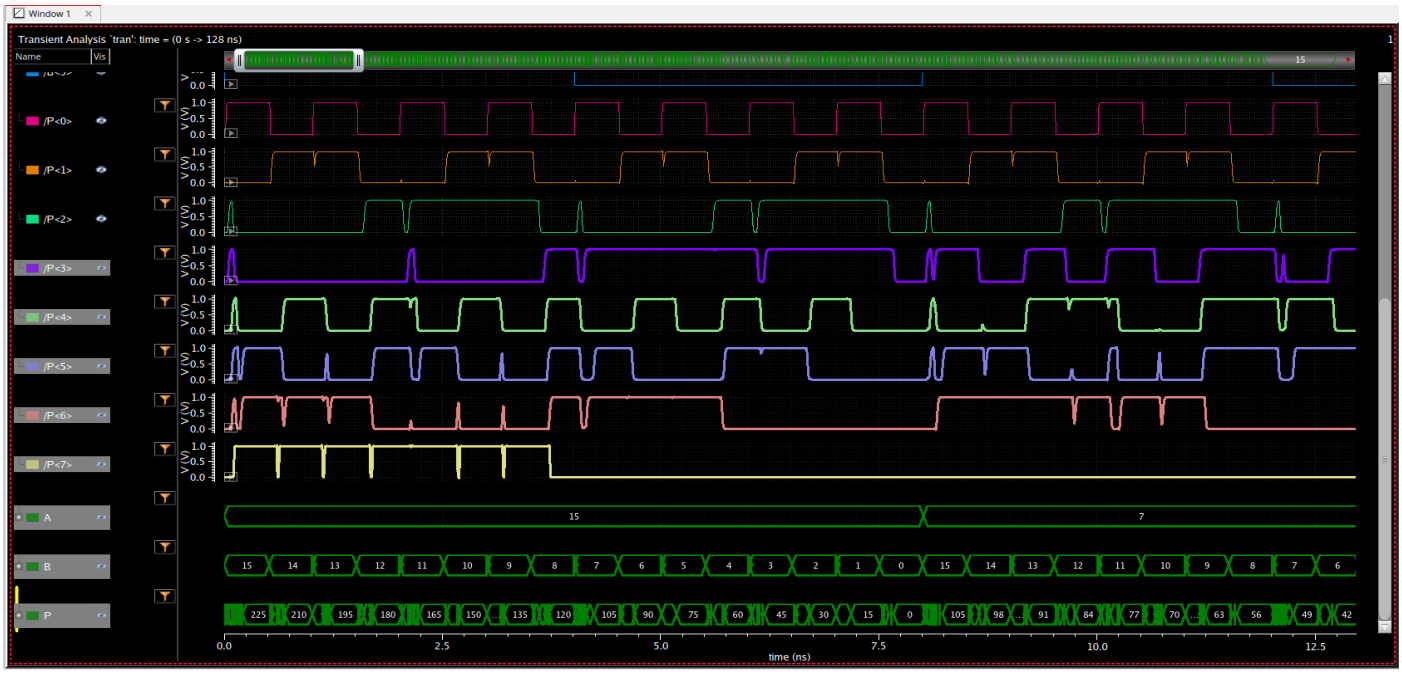
The 4×4 Binary Multiplier is designed to multiply two 4-bit binary numbers and generate an 8-bit product output. The circuit is implemented using AND gates, Half Adders (HAs), and Full Adders (FAs) arranged in a structured array configuration. The AND gates generate partial products by multiplying each bit of one operand with every bit of the other. These partial products are then properly aligned and summed using Half and Full Adders to obtain the final product. Half Adders are used in positions where only two bits need to be added, while Full Adders handle cases with three inputs, including the carry. Although advanced techniques such as Booth and Wallace Tree multipliers can be used for faster computation, this design using Half adders and Full adders are chosen for its simplicity, low power, and ease of implementation in Cadence Virtuoso. The multiplier was simulated successfully, and the waveform confirmed correct partial product generation and addition with smooth logic transitions. The design demonstrated accurate multiplication, efficient operation, and stable performance suitable for integration into arithmetic units or small-scale digital processors. For the 4×4 Binary Multiplier, the average power consumption was measured as 54.46 µW, and the peak instantaneous power was 1.126 mW.

Case: when **A = 1111 (15)** and **B = 1111 (15)**, the output product was **P = 11100001 (225)**, confirming correct multiplication.



The figure shows the 4×4 binary multiplier structure implemented using AND, Half Adder, and Full Adder blocks. Partial products (A_iB_j) are generated using 16 AND gates and summed column-wise with 4 Half Adders and 8 Full Adders to produce the final 8-bit output. This design was chosen for its simplicity, clarity, and easy implementation in Cadence Virtuoso.

Figure 14: 4x4 Binary Multiplier circuit diagram, structure and Symbol



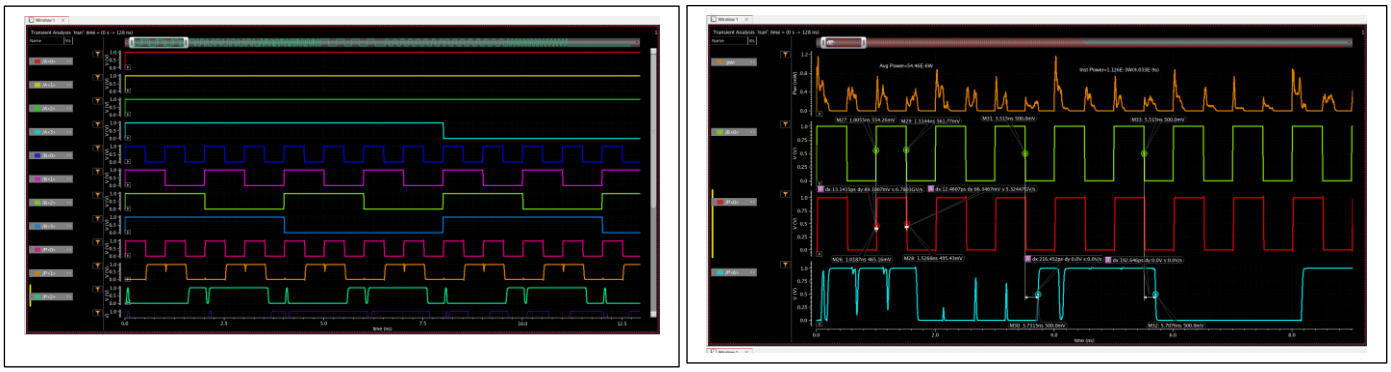


Figure 15: 4x4 Binary Multiplier simulation, Power and Delay

(iii) Incrementer- *Performs the operation of increasing a binary number by one.*

The 8-bit Binary Incrementer was designed to increase the value of input B by one for each operation. The circuit was built using a series of Half Adders connected in sequence, where the least significant bit (LSB) Half Adder receives a constant logic '1' to start the increment process. The carry generated from each stage is passed to the next higher bit, ensuring proper carry propagation across all eight bits. When the input reaches 11111111_2, the output rolls over to 00000000_2, generating a carry-out that indicates overflow. The final carry output represents overflow when the result exceeds the 8-bit range, and the sum outputs give the incremented value of B. During simulation, the circuit produced the correct output with smooth carry transitions through all stages. The average power consumption was measured as 6.35 µW, and the peak instantaneous power as 617.4 µW, showing that the design operates efficiently.

CASE : B = 00001111 (15);B + 1 = 00010000 (16)

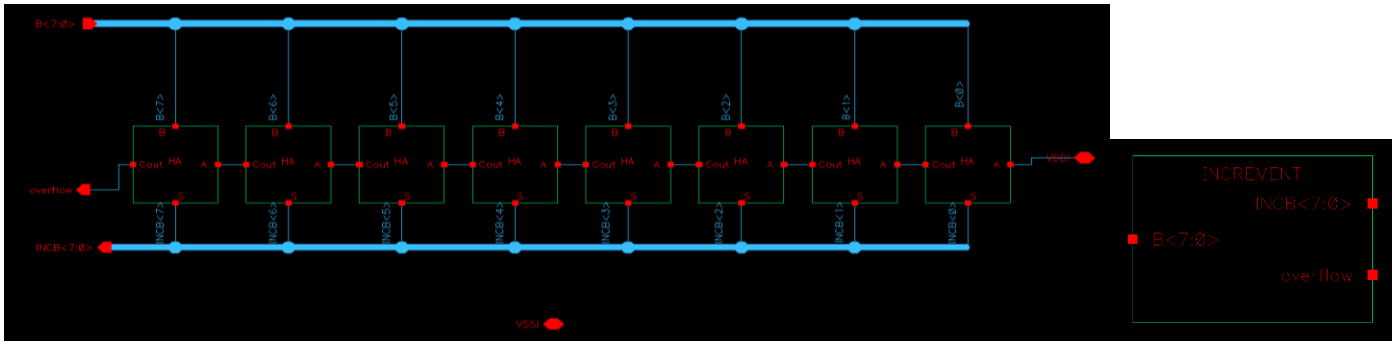
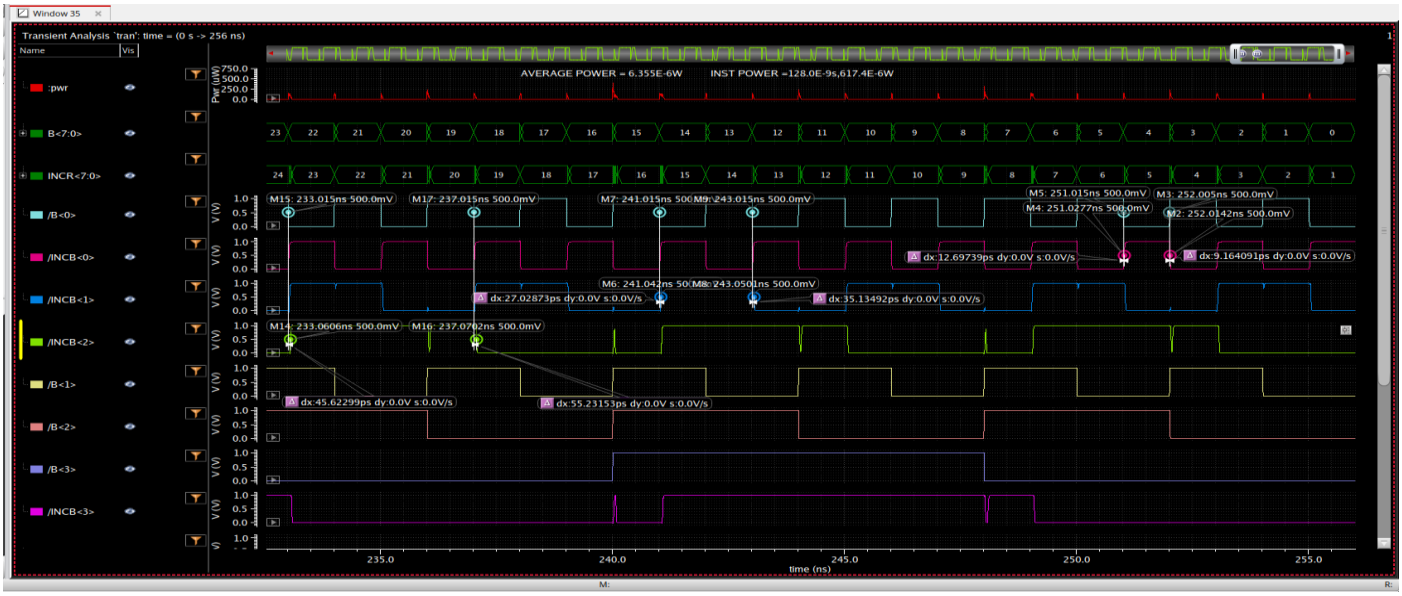


Figure 16:8-BIT INCREMENT Circuit, Symbol



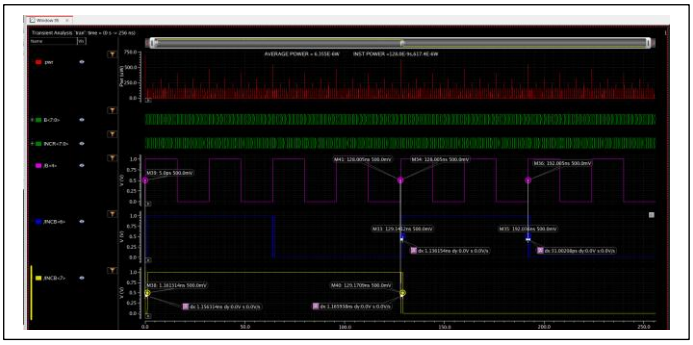


Figure 17: 8-BIT INCREMENT simulation Power and Delay

(iv) Decrementer - *Performs the operation of decreasing a binary number by one.*

The 8-bit Binary Decrementer was designed to reduce the value of input B by one for each operation. The circuit was built using a chain of Full Adders connected in sequence, where the least significant bit (LSB) Full Adder receives a logic '1' and a carry-in connected to logic '0' to start the decrement process. Each bit of B is complemented before addition, allowing the circuit to perform subtraction using the 2's complement method. During simulation, the circuit showed proper carry propagation and smooth logic transitions, correctly reducing the value of B by one at each step. A slight delay was observed at the output, which will be verified and optimized in the next stage of design. The average power consumption was measured as 11.76 µW, and the peak instantaneous power as 34.36 µW.

CASE: B = 00010000 (16) ; B − 1 = 00001111 (15)

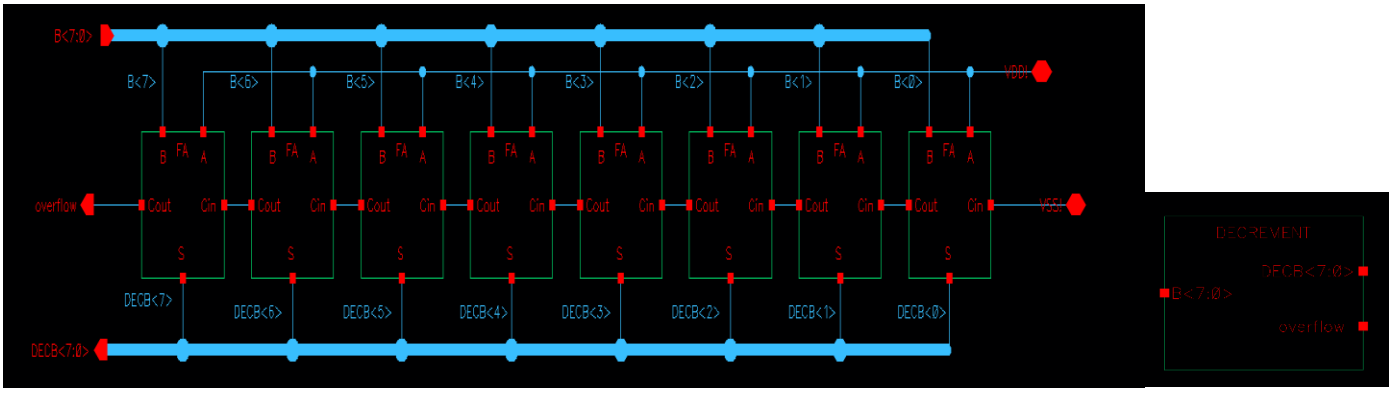
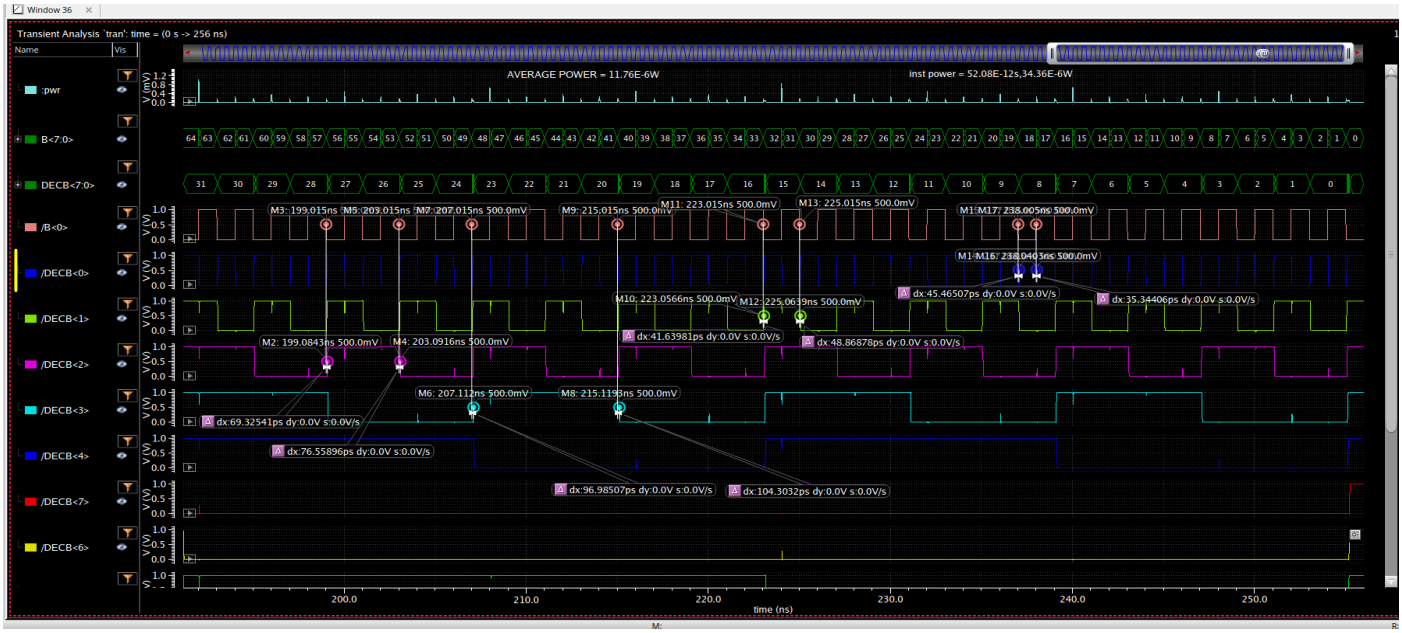


Figure 18:8-BIT DECREMENT Circuit, Symbol



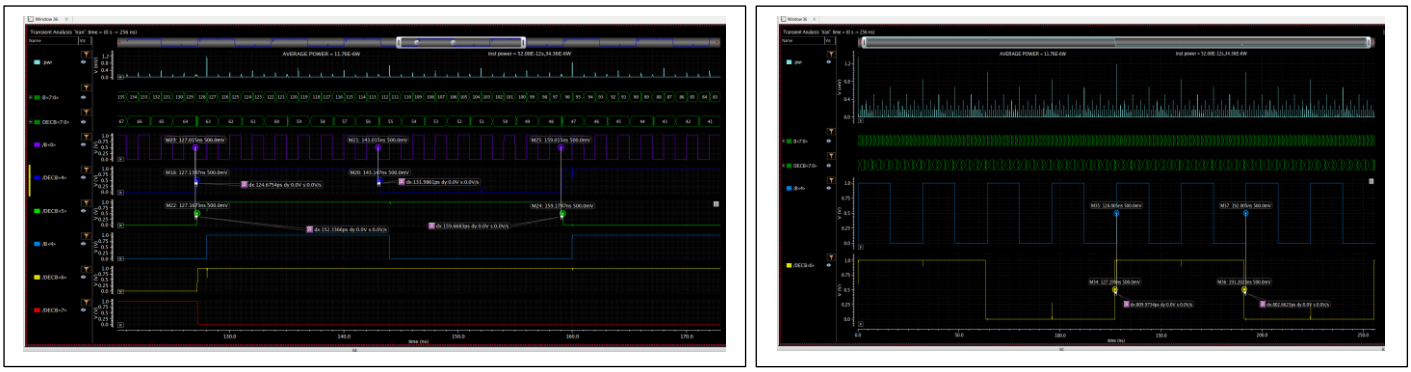


Figure 19: 8-BIT DECREMENT simulation Power and Delay

(v) Logical AND Block*-Performs bitwise AND operation between two 8-bit binary inputs.*

The 8-bit AND Gate was designed to perform a bitwise logical AND operation between two 8-bit inputs, A and B. Each output bit is generated using a two-input AND gate, which produces a logic high ('1') only when both input bits are high. The circuit consists of eight AND gates connected in parallel, allowing all bits to be processed simultaneously. During simulation, the circuit produced correct outputs for every input combination, confirming accurate bitwise AND functionality and stable operation. The transitions between logic levels were smooth throughout the waveform. The measured average power consumption was 3.818 µW, and the peak instantaneous power was 127.7 µW, demonstrating that the design operates efficiently.

Case: A = 10001000, B = 11001000 → Output Y = 10001000, confirming correct bitwise AND operation.

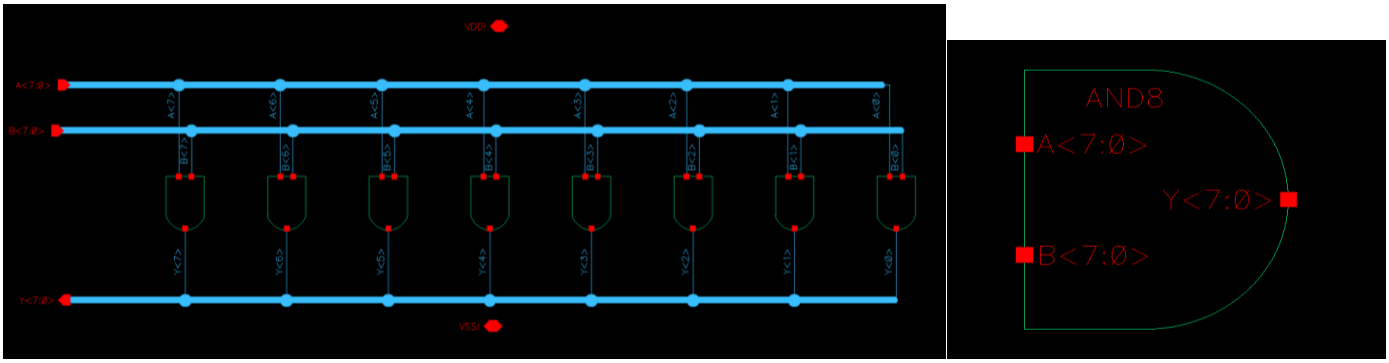


Figure 20:8-BIT LOGICAL AND Circuit, Symbol

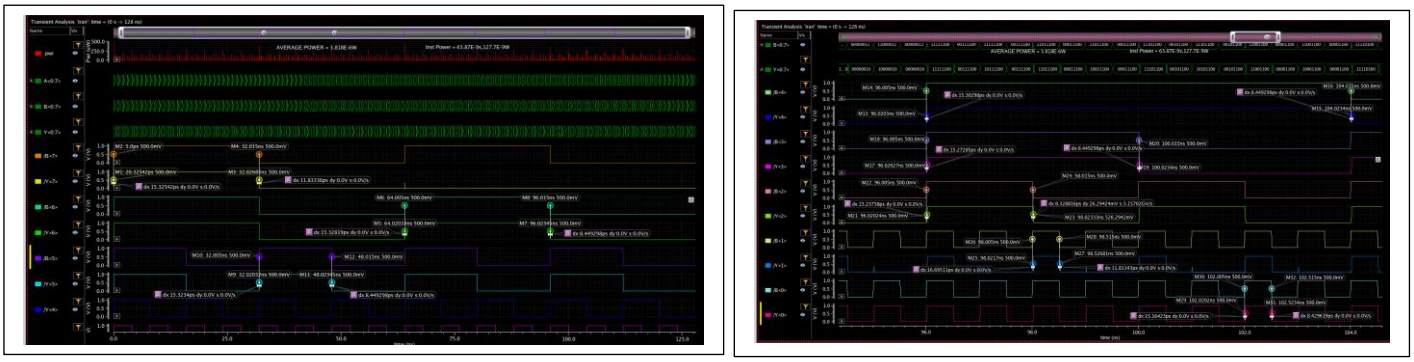


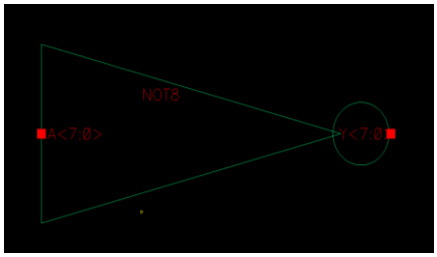
Figure 21: 8-BIT LOGICAL AND simulation Power and Delay

(vi) Logical NOT Block- *Performs bitwise logical inversion of an 8-bit binary input.*

The 8-bit Inverter was designed to perform bitwise logical inversion of the input A. The circuit consists of eight NOT gates connected in parallel, where each gate inverts one corresponding bit of the input. When the input is logic high ('1'), the output becomes logic low ('0'), and vice versa. This configuration allows all bits to be inverted simultaneously, making the design simple, efficient, and suitable for logic control or signal inversion applications in digital systems. During simulation, the inverter produced accurate outputs for all input combinations with smooth and stable transitions, confirming correct functionality.

The power analysis showed that the circuit consumes very little power while maintaining high reliability. The average power consumption was measured as 1.079 µW, and the peak instantaneous power was 132.4 µW.

Case: For input A = 11100000, the output Y = 00011111, confirming correct bitwise inversion of all bits.



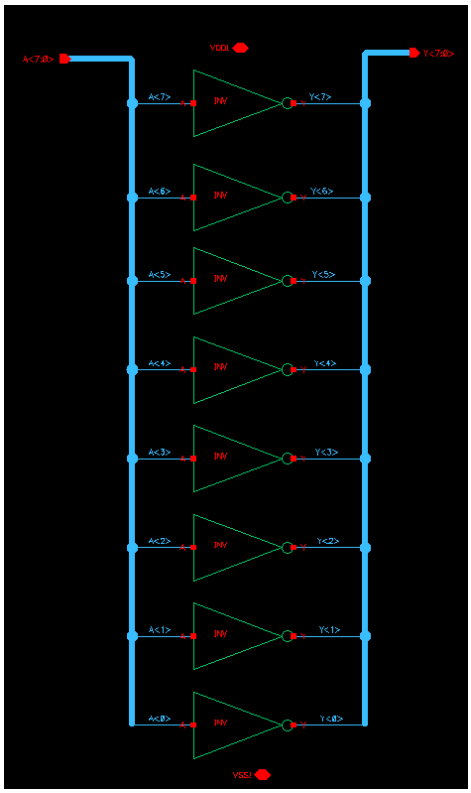
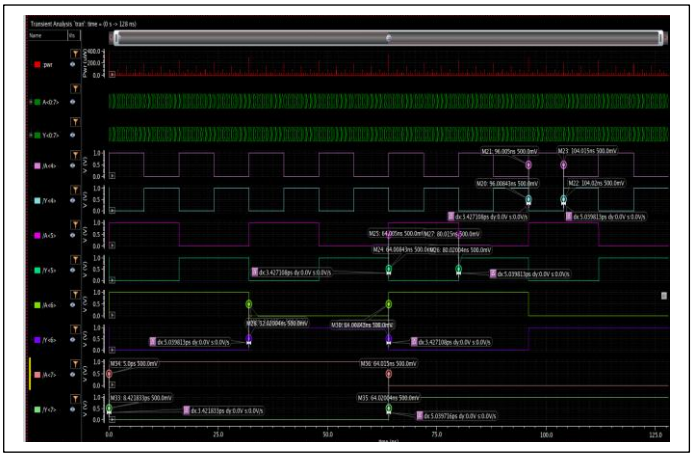


Figure 22:8-BIT LOGICAL NOT Circuit, Symbol



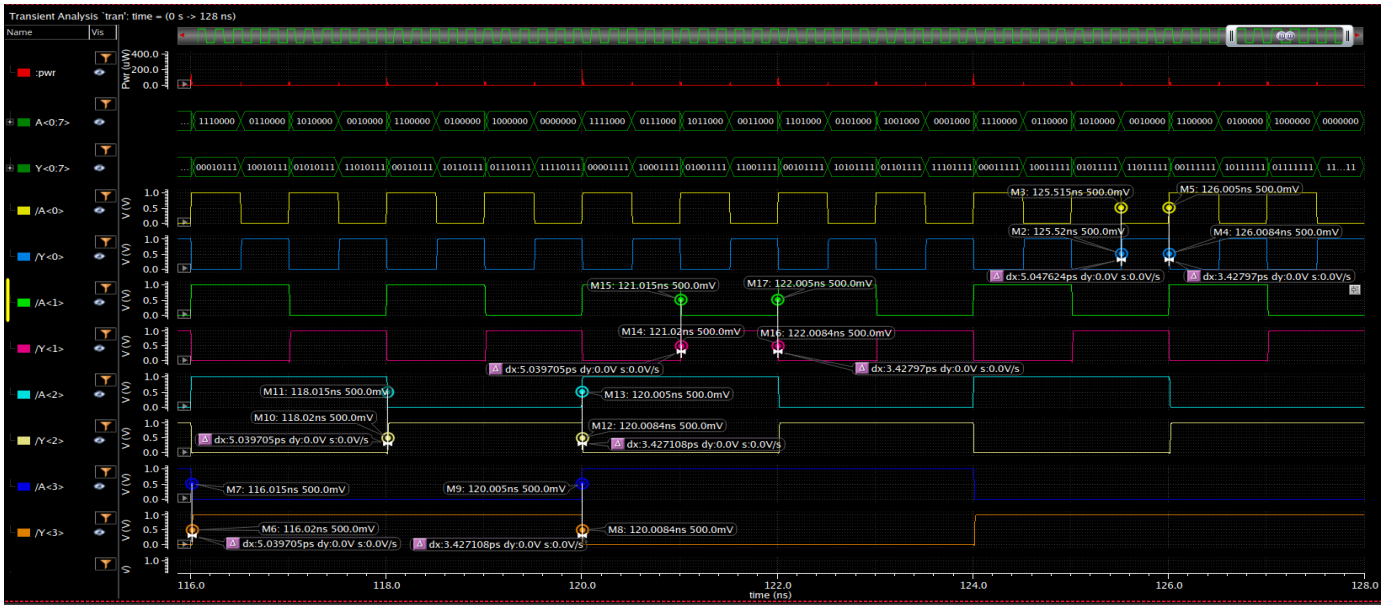
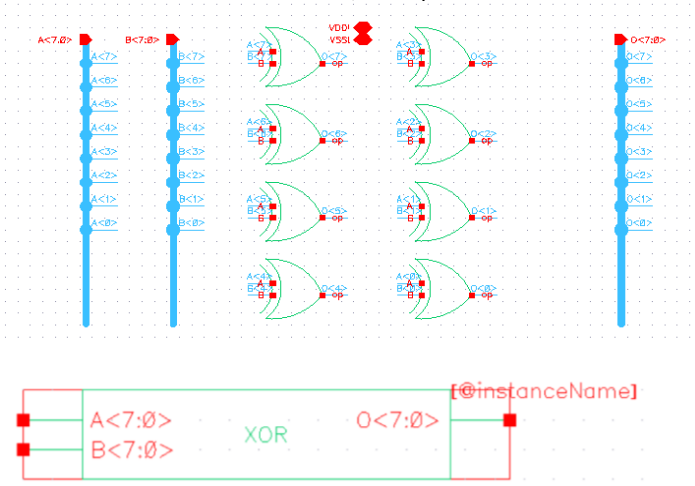


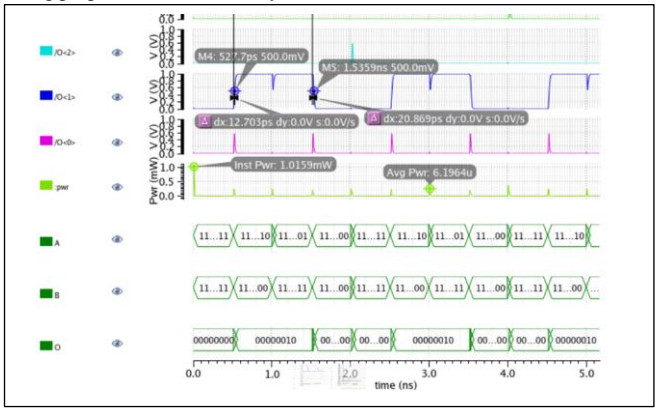
Figure 23: 8-BIT LOGICAL NOT simulation Power and Delay

(vii) Logical XOR Block- *Performs bitwise XOR operation between two 8-bit binary inputs.*

The 8-bit XOR Gate was designed to perform a bitwise exclusive OR operation between two 8-bit binary inputs, A and B. The circuit was implemented using eight two-input XOR gates connected in parallel, where each gate produces a logic high ('1') only when the corresponding bits of A and B are different. This allows all bits to be processed at the same time, ensuring fast and accurate logic operations. During simulation, the circuit produced correct outputs for every input combination with smooth transitions between logic levels. The average power consumption was measured as 6.196 µW, and the peak instantaneous power as 1.015 mW.

Case : A = 11111100 and B = 01111100, the output obtained was Y = 10000000, confirming proper XOR functionality.





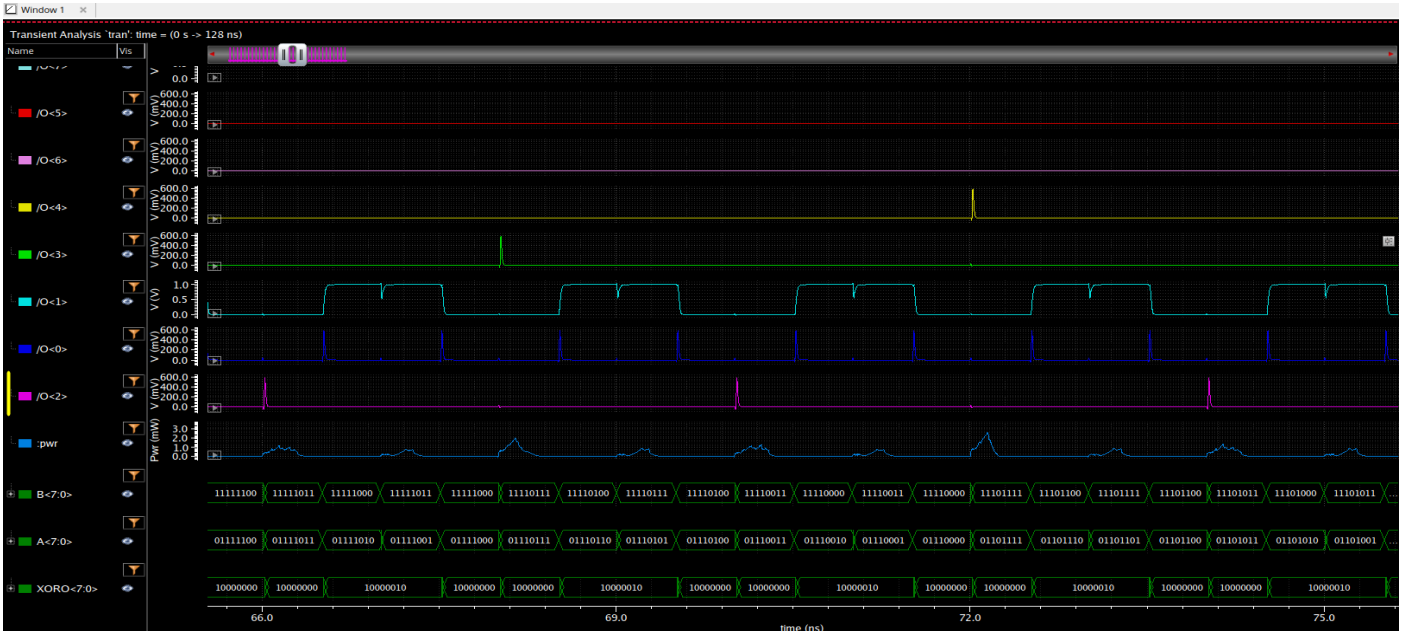


Figure 24: 8-BIT LOGICAL XOR Circuit, Symbol, simulation waveform, Power and Delay

***(viii) Logical Shift Block**-Performs bitwise logical left or right shift operations on an 8-bit binary input based on the control signals.*

The designed circuit represents an 8-bit barrel shifter implemented using cascaded 2:1 multiplexers in Cadence Virtuoso. It performs logical shifting of an 8-bit input (A7-A0) based on the three select inputs (S2, S1, S0), which determine the number of bit positions the data is shifted. The first stage performs a 1-bit shift, the second stage performs a 2-bit shift, and the third stage performs a 4-bit shift, allowing shifts from 0 to 7 bits. When all select lines are low (S2S1S0 = 000), the output remains the same as the input, and as the select combinations change, the bits shift toward the right/left in a parallel and synchronized manner. The transient simulation confirmed that all output bits (Q0–Q7) responded correctly for every select line combination, showing stable and accurate shifting behavior. From the power analysis, the average power consumption was 202.079 µW, while the instantaneous power peaked at around 3.462 mW.

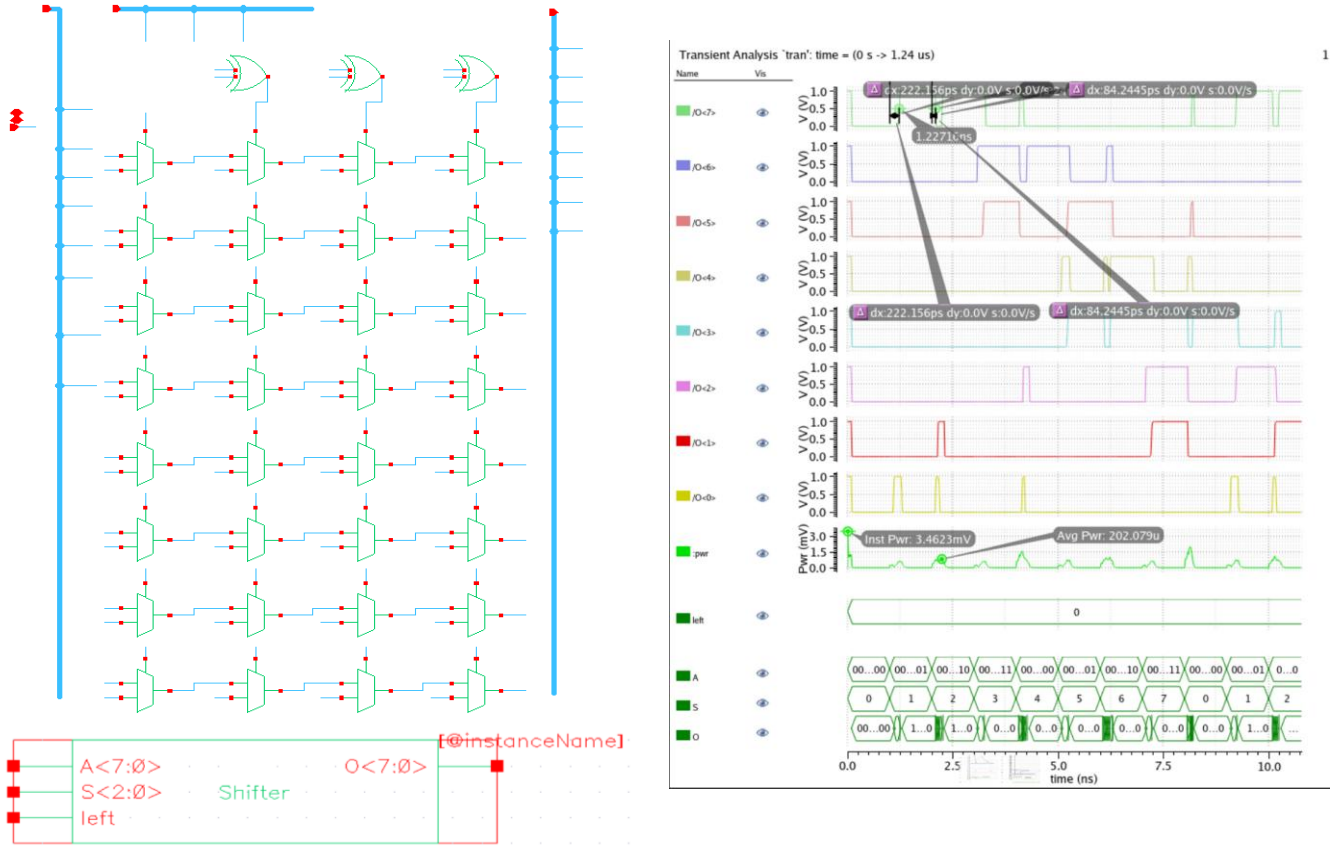


Figure 25: 8-BIT LOGICAL SHIFT (LEFT/RIGHT) Circuit, Symbol, simulation waveform, Power , Delays

Figure 26: 8-BIT LOGICAL SHIFT simulation Power and Delay

E. Multiplexer (MUX)

Functionality: In this design, the multiplexer (MUX) serves as a key control element that manages how data moves within the processor. It works by selecting one of the input signals based on the control line, ensuring that only the required data path is active at a given time. When the fetch signal is high, the MUX allows the Program Counter (PC) output to pass through to fetch the next instruction from memory, and when the fetch signal is low, it switches to the Instruction Register (IR) to access the operand needed for execution. This controlled switching helps maintain proper coordination between instruction fetch and execution stages. The MUX also allows multiple components to share a common data bus, which simplifies the circuit design and reduces hardware usage. Overall, it plays an important role in maintaining smooth data flow, minimizing timing errors, and enhancing the overall performance and efficiency of the processor.

Design and Implementation:

(i)2X1 MUX:

The designed circuit implements a 2:1 multiplexer at the transistor level using complementary CMOS logic in Cadence Virtuoso. It includes two data inputs (A0 and A1), one select input (S), and an output (OP), with an inverter used to generate the complement of the select signal (S′). When S = 0, the output follows A0, and when S = 1, it follows A1, corresponding to the logic equation OP = (A0 AND ~S) OR (A1 AND S). The transient simulation was carried out for a time range of 0 -16 ns, and the waveform confirmed that the output changes correctly with respect to the select signal, producing accurate and stable logic transitions. The circuit demonstrated smooth switching between inputs without any glitches. From the power analysis, the average power consumption was measured as 2.326 µW, and the instantaneous power peaked at approximately 309.276 µW.

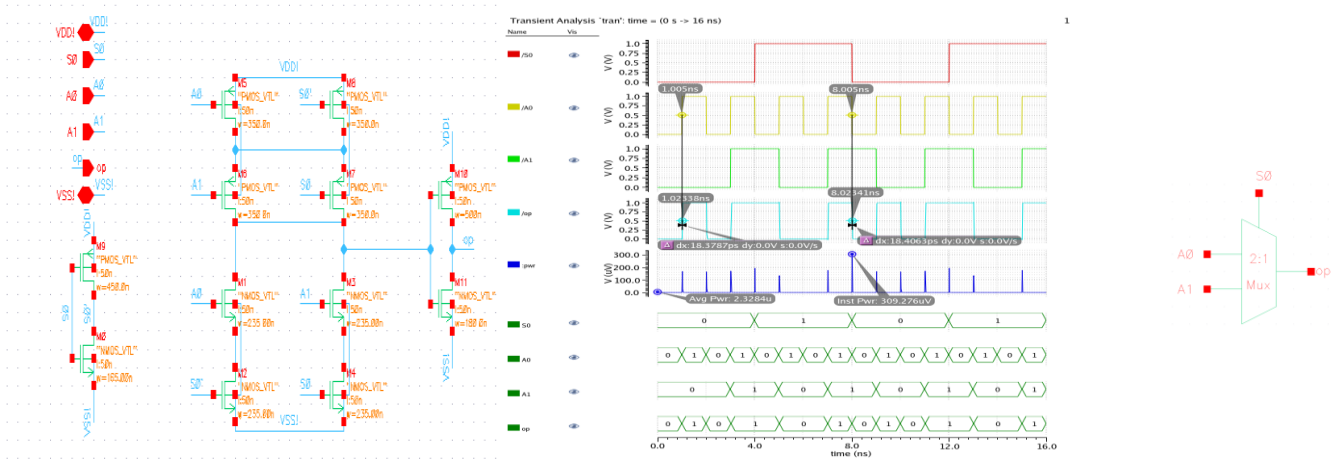


Figure 27: 2x1 MUX schematic, symbol and waveform

(ii)4X1 MUX:

The designed circuit represents a 4:1 multiplexer built at the transistor level using complementary CMOS logic in Cadence Virtuoso. It has four data inputs (A0, A1, A2, A3), two select inputs (S0 and S1), and one output (OP). Depending on the select line combination, the output follows A0 when S1S0 = 00, A1 when S1S0 = 01, A2 when S1S0 = 10, and A3 when S1S0 = 11. The circuit was simulated for a time range of 0–16 ns, and the waveform verified that the output changed correctly with each select input combination, showing proper logic operation. From the power analysis, the average power consumption was found to be 3.254 µW, and the instantaneous power reached approximately 576.28 µW.

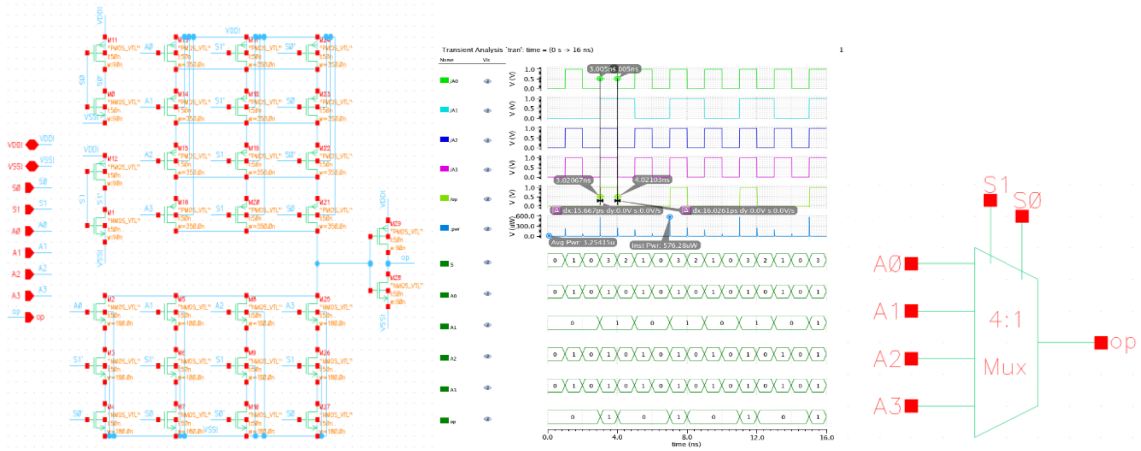


Figure 28: 4x1 MUX schematic, symbol and waveform

(iii)8X1 MUX using 4X1 MUX and 2X1 MUX:

The designed circuit implements an 8:1 hierarchical multiplexer using two 4:1 multiplexers and one 2:1 multiplexer in Cadence Virtuoso. It has eight data inputs (A0–A7), three select inputs (S0, S1, S2), and one output (OP). The two 4:1 multiplexers form the lower stage, each selecting one of four inputs based on S0 and S1, while the upper 2:1 multiplexer uses S2 to choose between the outputs of the lower-stage multiplexers to produce the final output. This hierarchical structure makes the design more efficient by reducing complexity. The transient simulation was carried out for 0-16 ns, and the waveform confirmed that the output switched correctly for all select input combinations. From the power analysis, the average power consumption was 10.042 µW, and the instantaneous power reached around 1.070 mW.

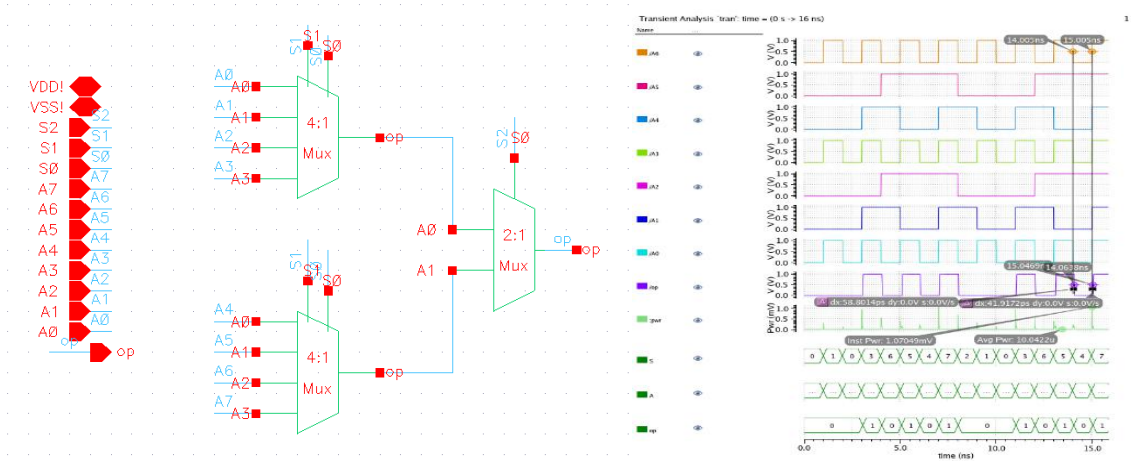


Figure 29: 8x1 hierarchical MUX schematic, symbol and waveform

(iv) 8X1 MUX:

The designed circuit represents an 8:1 multiplexer implemented at the transistor level using complementary CMOS logic in Cadence Virtuoso. It consists of eight data inputs (A0–A7), three select inputs (S0, S1, S2), and one output (OP). The output changes based on the combination of select signals, allowing one of the eight inputs to be transmitted at a time. The transient simulation was performed for a time range of 0-16 ns, and the waveform confirmed that the output responded correctly to all select input combinations, showing clean and stable switching behavior. From the power analysis, the average power consumption was measured as 3.195 µW.

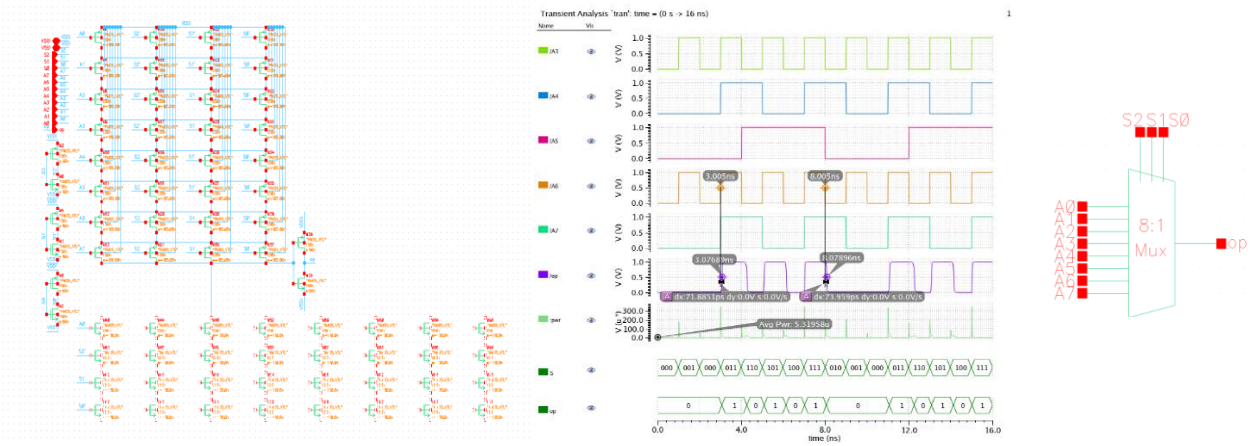


Figure 30: 8x1 MUX schematic, symbol and waveform

F. Instruction Register (IR)

The instruction register (IR) is used to store the instruction fetched from memory before it is executed. It consists of an 8-bit instruction, where the upper 3 bits represent the opcode and the lower 5 bits represent the address field. On the rising edge of the clock, when the load signal is active, the instruction is captured and sent to the decoder for further processing. This ensures proper sequencing and synchronization during program execution. In the current design, T flip-flops (TFFs) were used for testing purposes, which caused the output to toggle with every clock pulse. This was done as a trial implementation, and in the final design, D flip-flops (DFFs) will be used to provide stable and accurate output for reliable instruction storage and execution.

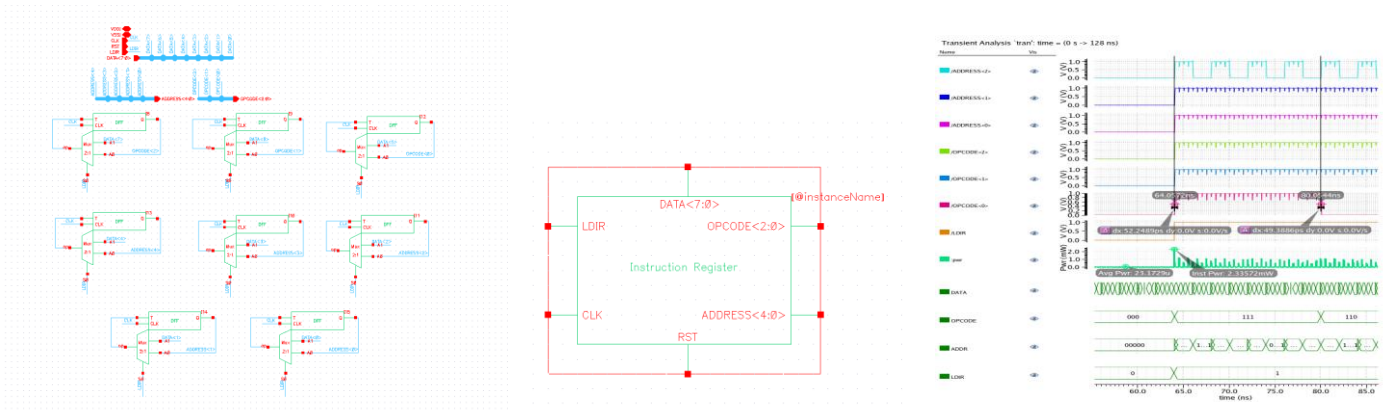


Figure 31: Instruction Register schematic and symbol.

G. Accumulator

Functionality: The accumulator is an important register in the processor that stores the output from the ALU after arithmetic or logical operations. It works as a temporary storage element, keeping the result that can be used directly in the next operation without the need to access memory repeatedly. In the current design, the 8-bit accumulator is implemented using D flip-flops (DFFs) to provide stable and accurate data storage. The accumulator receives inputs such as the clock, load (LDAC), reset, and the ALU output (ALOUT), and gives the stored result as ACOUT. On the rising edge of the clock, when both load and reset are active, the accumulator captures and stores the data from the ALU output. The stored value is then used for further processing, ensuring smooth data flow and synchronization, which helps improve the overall speed and efficiency of the processor.

Trial Implementation:

In the initial trial, a custom D flip-flop with enable (DFF-EN) and reset was designed using basic logic gates such as AND, OR, and NOT to control the flow of data with respect to the clock and enable signals. During simulation, the circuit did not hold the data as expected and instead showed toggling behavior at the output due to timing mismatches between the flip-flop and the connected multiplexer. This trial helped in understanding the importance of synchronization and data stability in sequential circuits. After this observation, the design was improved by implementing a verified D flip-flop with enable (DFFE), which provided stable data storage and ensured proper operation of the accumulator in the final implementation.

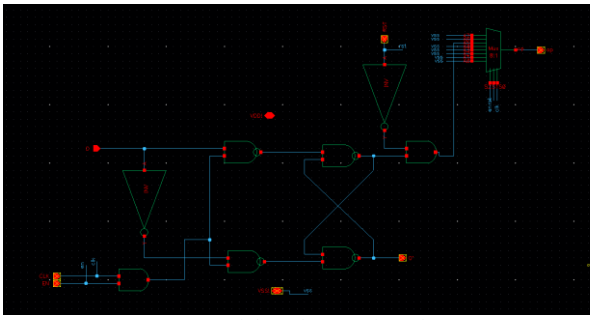
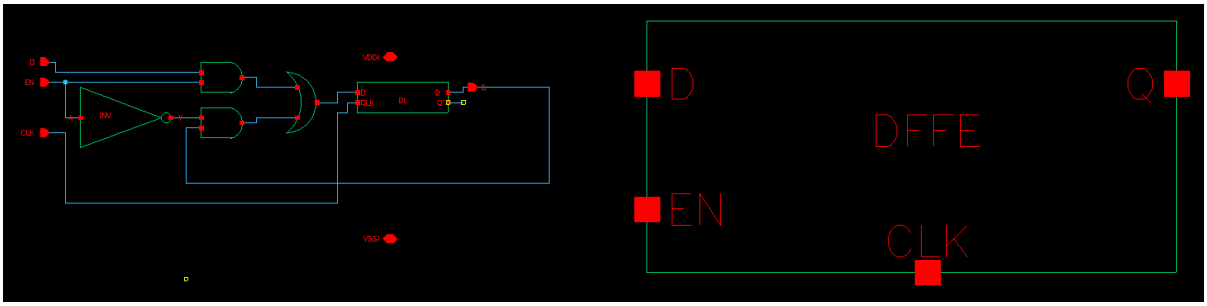


Figure 32: Trial Implementation of D-FF

Design and Implementation:

The 8-bit accumulator is designed using D flip-flops with enable (DFFE), where each flip-flop stores one bit of data from the ALU output (ALOUT). The accumulator receives clock (CLK), enable/load (LDAC), and reset (RST) signals. When the load signal is active and the clock rises, the accumulator captures the ALU output and stores it. The stored output is then available at ACOUT[7:0] and can be used for further processing by the ALU in the next cycle. The use of DFFEs helps ensure stable data storage, preventing unwanted toggling when the load signal is inactive.



. **Figure 33:** D flip flop schematic and symbol.

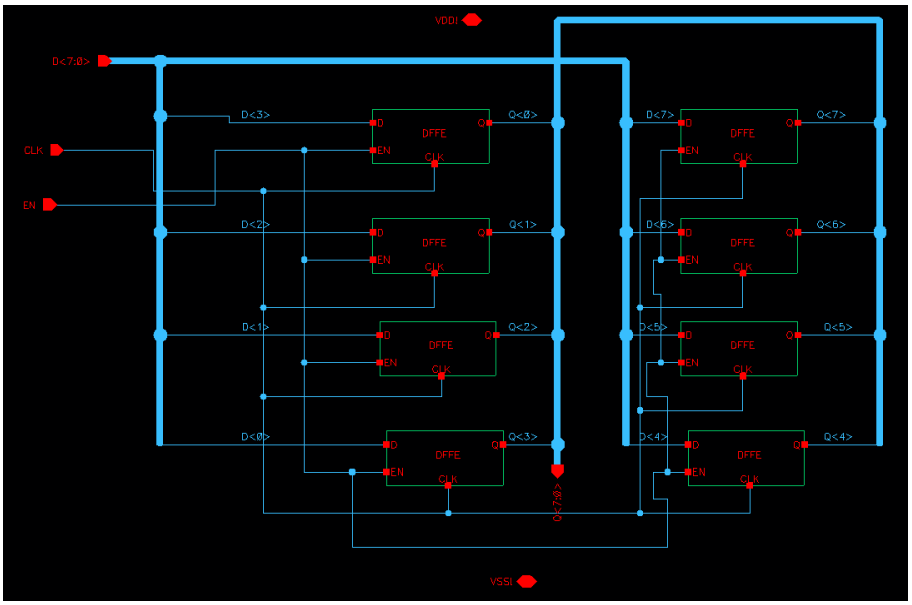


Figure 34: 8-BIT Accumulator schematic

Simulation and Validation:

The transient simulation verified that the accumulator correctly captures and holds data from the ALU output on each active clock edge when enabled, showing stable and accurate 8-bit output transitions.

Figure 35: Accumulator simulation waveform

IV. SIMULATION RESULTS

The simulation results verified the correct functionality of each designed block in the 8-bit RISC processor. All transient waveforms demonstrated accurate logical behavior, and proper synchronization under applied clock and reset conditions. The measured performance metrics for each module, including propagation delay and power consumption, confirmed efficient operation within the expected design limits. Overall, the results validated that the transistor-level implementation using 45 nm CMOS technology provides reliable performance suitable for integration into a complete processor.

Table : Performance Summary of Designed Modules

Module	Average Power (µW)	Instantaneous Power (mW)
PROGRAM COUNTER	23.09	0.266
DECODER	127.8	0.55
8-BIT ADDER / SUBTRACTOR	60.71	1.313
4×4 MULTIPLIER	54.46	1.126
INCREMENTER	6.35	0.617
DECREMENTER	11.76	0.034
LOGICAL AND	3.818	0.127
LOGICAL XOR	6.196	1.015
LOGICAL NOT	1.079	0.132
LOGICAL SHIFTER	202.079	3.462
ACCUMULATOR	202.079	3.462
8X1 MULTIPLEXER	3.195	-
2X1 MUX	2.326	0.309
4X1 MUX	3.254	0.576
8X1 MUX (HIER)	10.042	1.07
INSTRUCTION REGISTER	23.17	2.33

V. DISCUSSION AND FUTURE WORK

This phase successfully demonstrated the design and simulation of individual processor modules using a hierarchical CMOS design approach. Each module worked as expected and helped in understanding circuit behavior, timing, and power performance. In the upcoming phase, we plan to design and implement the memory unit, ring oscillator, and instruction memory to make the processor fully functional. Future improvements will include optimizing the clock, analyzing power-delay trade-offs, and adding pipeline stages to improve performance and to remove glitches.Then next step will focus on integrating all the main components-Program Counter, Instruction Register, Decoder, ALU, Memory and Accumulator—into a single processor to enable complete instruction execution. Complete instruction flow testing will be carried out to ensure proper coordination between the control and data paths for reliable and efficient operation of the final processor.

VI. CONCLUSION

To conclude, this phase of the project focused on designing and verifying the key modules of an 8-bit RISC processor at the transistor level using Cadence Virtuoso in 45 nm CMOS technology. All individual blocks were successfully implemented and tested, showing correct logic behavior, stable operation, and efficient power usage. The work completed so far provided a strong understanding of CMOS circuit design, timing analysis, and module-level integration. This stage also helped build practical skills in analyzing waveforms, debugging circuit behavior, and optimizing performance. Moving forward, the next phase will involve combining all designed modules to form a complete processor and developing additional components such as the memory unit, ring oscillator, and instruction memory to enhance the processor's functionality and overall system performance.

VII. REFERENCES

[1] IJRASET, "Design of a 16-Bit Harvard Structure RISC Processor in Cadence 45 nm Technology." https://www.irjet.net/archives/V7/i6/IRJET-V7I6194.pdf
[2] Cadence Design Systems, "Cadence University Program — Digital Design and Verification Training Modules,"2025.
*[3] P. Magnusson and A. Hirano, *The RISC-V Reader: An Open Architecture Atlas*, 2nd ed., Kindle edition, 2018.*
https://www.cs.sfu.ca/~ashriram/Courses/CS295/assets/books/HandP_RISCV.pdf