IMPLEMENTATION OF AN 8-BIT RISC CPU

(45 nm CMOS Technology)

Keerthi Patil
Department of Electrical Engineering
Wright State University
Dayton, OH, USA
patil.115@wright.edu

Ramya Vadde
Department of Electrical Engineering
Wright State University
Dayton, OH, USA
vadde.16@wright.edu

Pierce Rob Edward
Department of Computer Engineering
Wright State University
Dayton, OH, USA
pierce.60@wright.edu

***Abstract* - This report presents the design and transistor-level implementation of an 8-bit RISC CPU using Cadence Virtuoso and the FreePDK45 (45 nm) CMOS technology. The processor was developed using a modular approach, where individual functional blocks-including the Instruction Pointer, Instruction Register, Decoder and Control Unit, Arithmetic Logic Unit (ALU), Accumulator, 8-bit Registers, Memory Block, Multiplexer network, and Ring Oscillator-were implemented using CMOS logic at the transistor level. Each block was independently designed and verified through transient simulations to confirm correct functional behavior under specified input and control conditions. Key performance parameters such as propagation delay, rise and fall times, power consumption, and Energy - Delay Product (EDP) were evaluated at the module level to assess circuit efficiency. After individual verification, the functional blocks were hierarchically integrated to form the complete CPU architecture, and system-level simulations were performed to validate signal connectivity, data flow, and control sequencing. This work demonstrates a complete transistor-level implementation of an 8-bit RISC CPU architecture and establishes a foundation for future optimization and physical layout development.**

Keywords - RISC CPU, Cadence Virtuoso, FreePDK45 CMOS, transistor-level design, Spectre simulation, propagation delay, power analysis, energy-delay product (EDP)

I. INTRODUCTION

Designing a small RISC-based processor provides a practical foundation for understanding how digital systems are organized, controlled, and verified at the transistor level. In this project, an 8-bit RISC CPU was implemented in Cadence Virtuoso using the FreePDK45 (45 nm) CMOS technology to explore the full design flow-from individual device-level circuits to a fully functioning processor. The CPU consists of several coordinated modules, including the Arithmetic Logic Unit, Accumulator, Decoder, Instruction Memory, Data Memory, Instruction Pointer, 8-bit Registers, Multiplexer circuitry, and a Ring Oscillator used as the internal clock source. Each module was built using transistor-level CMOS logic and represented hierarchically to create higher-level functional blocks.

A bottom-up methodology was adopted, beginning with the design and verification of basic logic gates and progressing toward the construction of complete processor components. Transient simulations were used at each stage to verify logic transitions, evaluate timing behavior, and confirm the correct operation of control signals under clock and reset conditions. Performance characteristics such as propagation delay, rise and fall times, power consumption, and the Energy-Delay Product (EDP) were examined to assess the efficiency of the implemented circuits. After individual module validation, all components were integrated to form the complete 8-bit RISC CPU. System-level simulations were used to verify signal connectivity, data flow, and control sequencing among the integrated modules, demonstrating a functional transistor-level implementation of an 8-bit RISC CPU architecture in 45 nm technology.

II. DESIGN METHODOLOGY

A. Design Objectives and Scope

The primary objective of this project is to design and verify a fully functional, transistor-level CPU capable of supporting a defined set of instruction operations using a synchronous datapath. The CPU consists of coordinated modules including the ALU, Accumulator, Decoder and Control Unit, Instruction Memory, Data Memory, Instruction Pointer (PC), 8-bit Registers, Multiplexer circuitry, and a Ring Oscillator used as the internal clock source. Each subsystem is implemented using CMOS logic, and the processor is constructed through hierarchical integration of these modules to demonstrate correct instruction sequencing, data manipulation, and control-signal generation within a stable clocked architecture.

The scope of this work includes designing every functional block at the transistor level, validating their individual behavior through simulation, and integrating them into a cohesive CPU datapath. The CPU supports essential operations such as data loading, arithmetic computation, accumulation, instruction decoding, and sequential instruction execution. Clocked register stages are used

to prevent transient glitches from propagating through the combinational datapath and ensure deterministic timing behavior. Simulation results are used to verify instruction flow, evaluate timing, and confirm correct interaction between the control unit and datapath. Rather than optimizing for speed or area, the project focuses on demonstrating functional correctness, architectural clarity, and complete transistor-level implementation of a minimal yet operational CPU.

B. Design Environment and Process Technology

The CPU was implemented and verified using the Cadence Virtuoso Custom IC Design Platform, which supports transistor-level schematic development, hierarchical cell construction, and mixed-signal simulation. All architectural blocks-including the ALU, Accumulator, Decoder and Control Unit, Instruction Memory, Data Memory, Program Counter, 8-bit Register File, Multiplexer network, and the on-chip Ring Oscillator- were designed using MOSFET devices from the FreePDK45 technology library. Circuit validation, including transient analysis, propagation delay extraction, and power estimation, was performed using the Cadence Spectre simulator within ADE Explorer, enabling accurate characterization of switching behavior and timing across the synchronous datapath.

The design targets the 45 nm FreePDK45 CMOS process, a predictive technology model that provides realistic parasitic behavior and timing characteristics suitable for transistor-level CPU implementation. Transistor sizing followed standard CMOS design practices to achieve balanced rise and fall transitions and sufficient drive strength for multi-stage datapath operation. The combined use of the Virtuoso design environment, Spectre simulator, and FreePDK45 process technology enabled accurate transistor-level modeling, reliable timing verification, and functional validation of the complete CPU architecture.

C. Bottom-Up Design Flow

A bottom-up design flow was followed to construct the CPU through progressive integration of verified circuit blocks. The process began with transistor-level realization of fundamental CMOS components, which were individually validated to satisfy functional correctness and timing constraints before abstraction into reusable modules. These validated components were systematically assembled into intermediate subsystems, including the ALU bit-slice, accumulator storage element, program counter stage, decoder logic, and multiplexing network. Each subsystem was simulated independently using the Cadence Spectre simulator to verify correct signal sequencing, control-signal coordination, and timing alignment prior to system-level integration. The complete CPU datapath was subsequently formed by interconnecting the verified subsystems through explicitly defined control and data interfaces, with all state updates synchronized using clocked register stages. This bottom-up flow enabled early isolation of design errors, reduced integration complexity, and ensured consistent timing behavior across the final transistor-level CPU implementation.

D. CMOS Gate-Level Development and Characterization

The CMOS gate-level implementation of the CPU was based on complementary pull-up and pull-down transistor networks realizing standard logic functions such as INV, NAND, NOR, AND, OR, XOR, and transmission-gate structures. Each gate was implemented using FreePDK45 MOSFET devices and sized to balance PMOS and NMOS drive strengths, minimize rise-fall delay mismatch, and support fan-out requirements within multi-stage logic paths. Gate-level characterization was performed using transient simulations in the Cadence Spectre simulator through ADE Explorer to extract propagation delays, output transition behavior, and switching integrity under representative operating conditions. The resulting characterization confirmed that the logic gates satisfied the timing requirements of the synchronous datapath. The validated gate-level cells were then encapsulated as reusable symbols and integrated into higher-level functional modules, including the ALU, decoder, and register structures.

E. Hierarchical Module Construction

Hierarchical module construction was carried out by assembling verified gate-level cells into clearly defined functional subsystems and progressively integrating them into the complete CPU architecture. Core modules-including the ALU, accumulator, decoder, program counter, register file, multiplexing network, instruction memory, and data memory-were structured according to their roles within the datapath and control path. Symbol-based hierarchy was preserved throughout the design to support modular reuse and simplify validation across abstraction levels. This organization separated combinational logic, sequential storage elements, and control circuitry, enabling scalable integration and predictable system-level behavior.

F. Simulation Methodology and Verification Steps

Simulation-based verification was performed at multiple abstraction levels to validate functional correctness and timing behavior of the CPU. Individual gates and subsystems were verified using transient (tran) simulations in the Cadence Spectre simulator, with

input stimuli applied to exercise functional modes and control conditions. System-level simulations were then conducted under clocked operation to observe control sequencing, datapath activity, address generation, and memory access behavior. This staged simulation methodology ensured reliable operation of the complete transistor-level CPU.

G. On-Chip Clock Generation

An on-chip ring oscillator was designed at the transistor level to demonstrate internal clock generation capability within the 8-bit RISC CPU architecture. The oscillator consists of 23 cascaded inverter stages connected in a closed feedback loop, producing sustained oscillations without the need for an external clock source. Transient simulations verified stable oscillation, with the oscillation frequency measured to be approximately 1.07 GHz under nominal operating conditions. Although the ring oscillator was fully designed and verified as a standalone block, it was not used as the clock source during full CPU functional verification. Instead, an externally applied clock stimulus operating at 2 GHz was used during system-level simulations to provide controlled timing conditions and ensure reliable evaluation of datapath and control behavior. The ring oscillator block is included to demonstrate clock-generation capability and to support potential future integration into a fully self-clocked.

H. Top-Level System Assembly

Top-level system assembly was completed by integrating all validated subsystems into a unified CPU schematic using well-defined control, datapath, and memory interfaces. Datapath components such as the ALU, registers, accumulator, and multiplexers were interconnected under the control of signals generated by the decoder and control unit, while address and memory paths were connected to support instruction fetch and data access operations. Clock distribution was applied consistently across all sequential elements to maintain synchronous behavior throughout the system. This integration stage finalized the construction of the complete transistor-level CPU and enabled full-system functional verification.

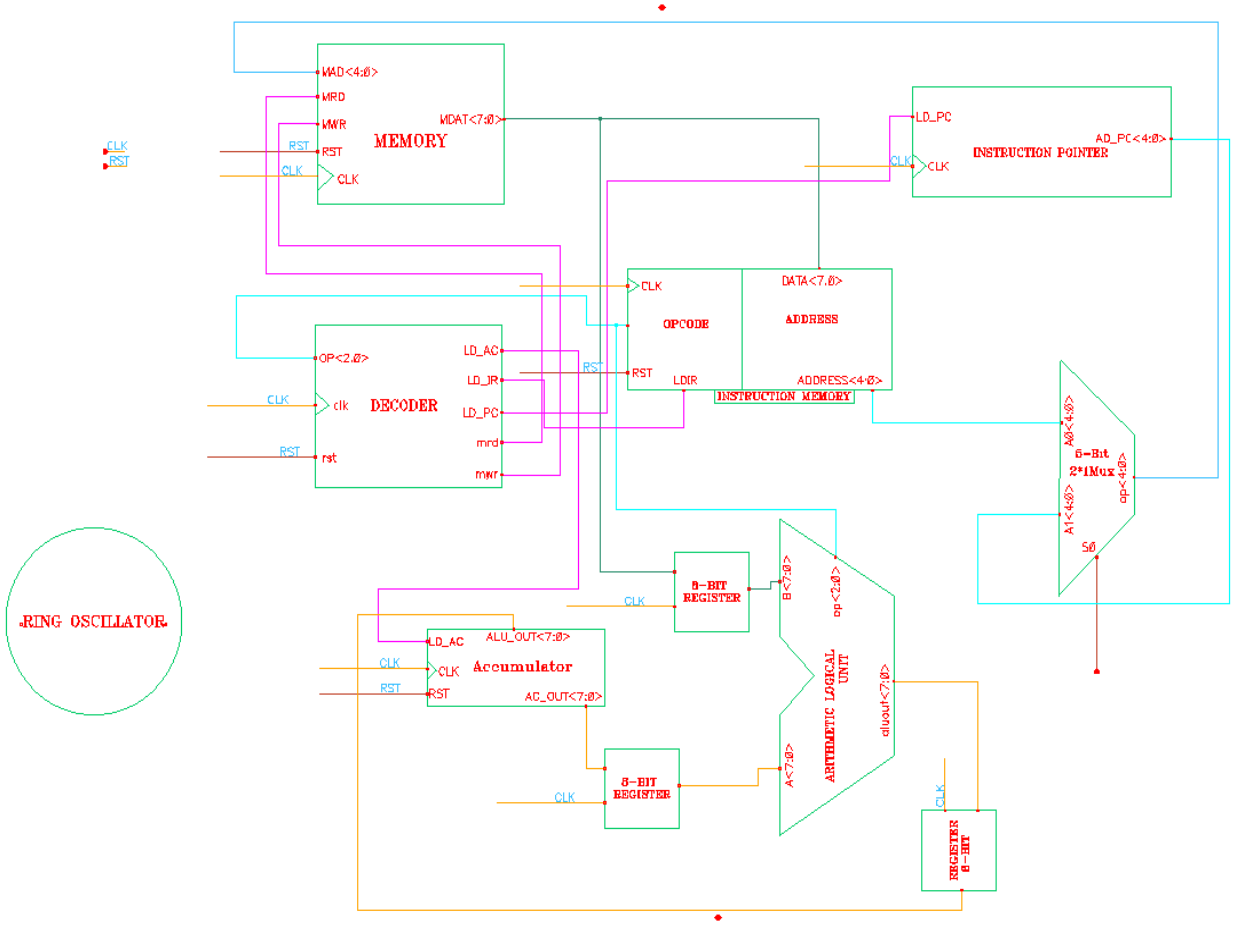


Figure 1. Top-level schematic of the 8-bit RISC CPU architecture.

I. Performance Evaluation Metrics

Performance evaluation focused on timing, power, and area characteristics derived from transistor-level simulations. Timing performance was characterized by measuring propagation delays, including high-to-low (tpHL) and low-to-high (tpLH) transitions, along critical datapath, address path, and memory access paths. Rise time and fall time were extracted to assess signal transition quality and timing consistency. Power analysis included both average and instantaneous power measured during representative switching activity. Silicon area was estimated based on the total number of logic gates, storage elements, and MOS transistors used in the design. The Energy–Delay Product (EDP) was computed to evaluate overall efficiency and to compare registered and unregistered CPU configurations.

J. Top-Level Architecture Representation

The top-level architecture representation depicts the structural interconnection of datapath components, control logic, memory subsystems, and clock distribution within the CPU. It highlights the flow of data, control signals, and timing relationships across instruction fetch, decode, execute, and write-back stages.

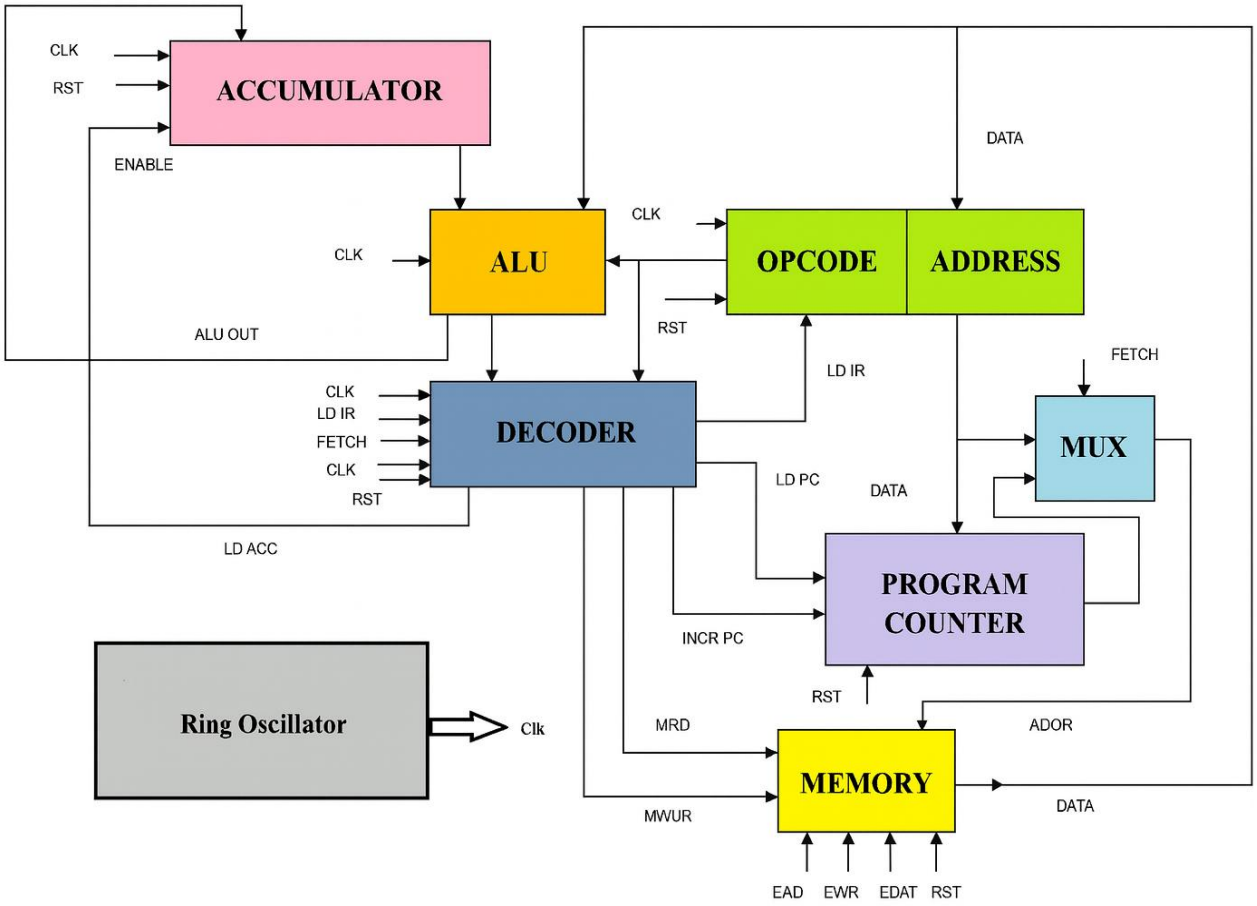


Figure 2. Block level architecture of the 8-bit RISC PROCESSOR.

III. CPU FUNCTIONAL BLOCKS AND OPERATION

A. Overview of Functional Blocks

The 8-bit RISC CPU is organized as a set of coordinated functional blocks that together enable synchronous instruction execution. The processor datapath includes an instruction pointer (program counter), instruction memory, arithmetic logic unit (ALU), accumulator, 8-bit registers, data memory, and a multiplexer network used for controlled data routing. A decoder and control unit interprets the opcode of each instruction and generates the control signals required to configure the datapath during every instruction cycle. An on-chip ring oscillator was designed to demonstrate internal clock generation capability; however, functional verification of the processor was carried out using an externally applied clock stimulus in order to maintain a fixed operating frequency of 2 GHz during simulation. The interaction among these functional blocks ensures correct instruction sequencing, data manipulation, and coordinated control, resulting in a complete and operational 8-bit RISC CPU architecture.

Table 1: Functional Blocks of the 8-bit RISC CPU

Block	Operation	Input Width (bits)	Output Width (bits)
Instruction Pointer (PC)	Increments instruction address when LD_PC is asserted	5	5
Instruction Register	Stores fetched instruction and separates opcode and operand	8	3 (Opcode) / 5 (Operand)
Decoder & Control Unit	Decodes opcode and generates control signals	3 (Opcode)	5 (Control signals, 1-bit each)
Arithmetic Logic Unit (ALU)	Performs arithmetic and logical operations	8	8
Accumulator	Stores intermediate and final ALU results	8	8
8-bit Registers	Temporary storage for operands and results	8	8
Data Memory	Stores data for read and write (load/store) operations	8 (Data In) / 5 (Address)	8(Data Out)
Multiplexer (MUX)	Selects and routes datapath signals	Variable (5-bit inputs)	5
Ring Oscillator	Generates the internal clock for synchronized processor operation	-	1 (Clock)

B. Processor Operation Cycle

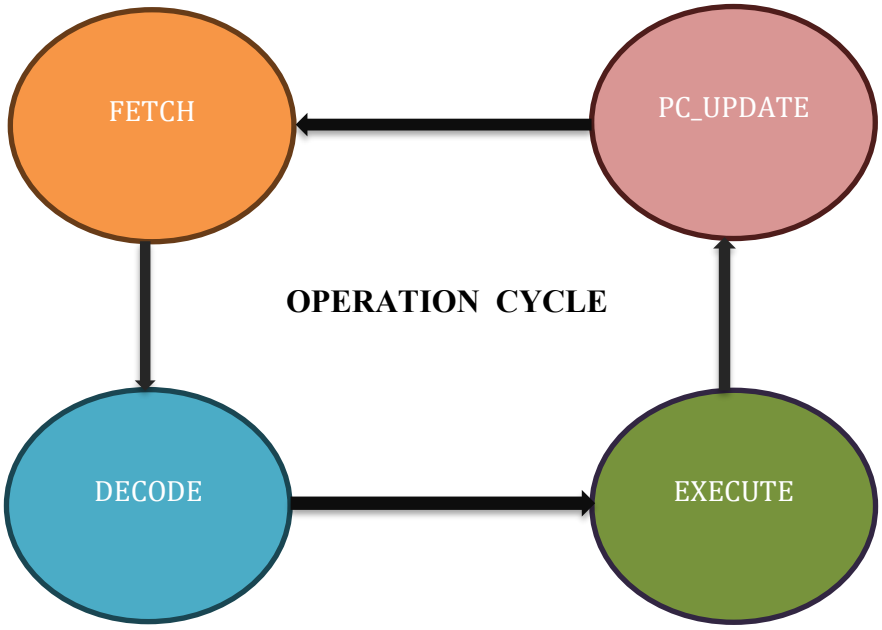
The proposed 8-bit RISC CPU operates using a four-stage instruction execution cycle that ensures coordinated operation among the datapath and control blocks. Each instruction progresses sequentially through the following stages.

- **Fetch:** During the fetch stage, the Program Counter (PC) supplies the address of the current instruction to the memory block. The fetched instruction is then loaded into the Instruction Register, where it is separated into opcode and operand fields required for subsequent stages.

- **Decode:** In the decode stage, the decoder interprets the opcode from the Instruction Register and generates the appropriate control signals. These signals configure the datapath by enabling registers, selecting multiplexer paths, and preparing the ALU and accumulator for execution.

- **Execute:** During execution, the Arithmetic Logic Unit performs the specified arithmetic or logical operation based on the decoded instruction. The resulting value is stored in the accumulator.

- **PC Update:** In the final stage, the Program Counter increments to point to the next instruction address, completing the current instruction cycle and preparing the processor for the next fetch operation.



IV. MODULE DESCRIPTIONS

A. Basic CMOS Logic Gates:

All basic logic gates used in the 8-bit RISC CPU were designed and implemented at the transistor level using the FreePDK45 (45 nm) CMOS technology. The gate library includes inverters, AND, NAND, OR, NOR, and XOR gates with varying input counts, each realized using complementary pull-up and pull-down transistor networks. These gates were independently verified through transient simulations to ensure correct logical behavior. The validated transistor-level gate schematics were encapsulated as reusable symbols and hierarchically integrated to construct higher-level functional modules, including the decoder, arithmetic logic unit, registers, multiplexers, and control circuitry.

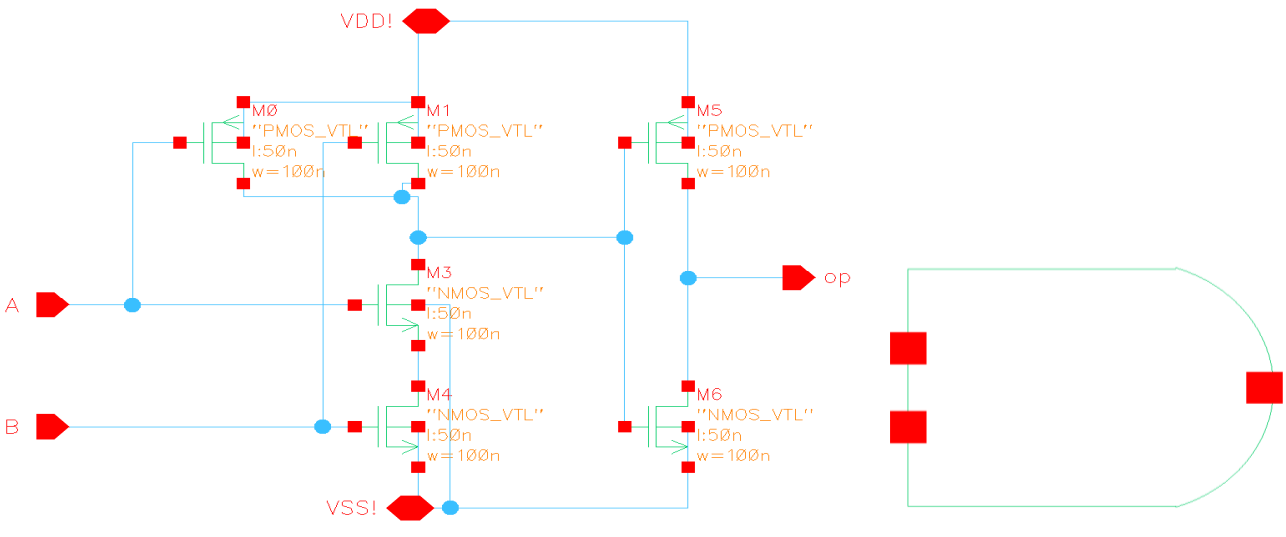


Figure 3. Transistor-level schematic and symbol representation of AND2 gate

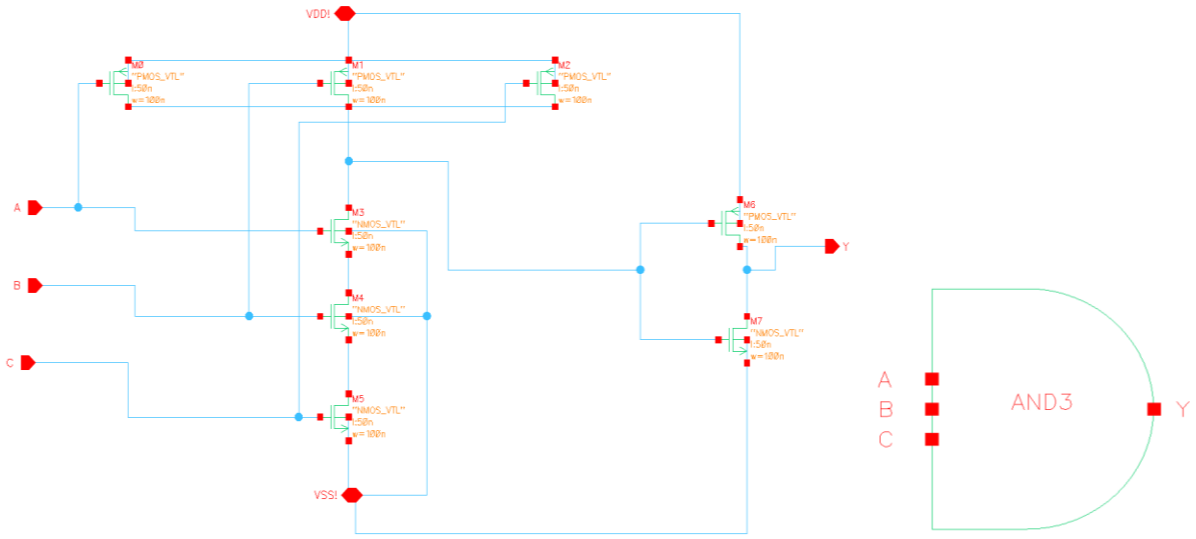


Figure 4. Transistor-level schematic and symbol representation of the AND3 gate.

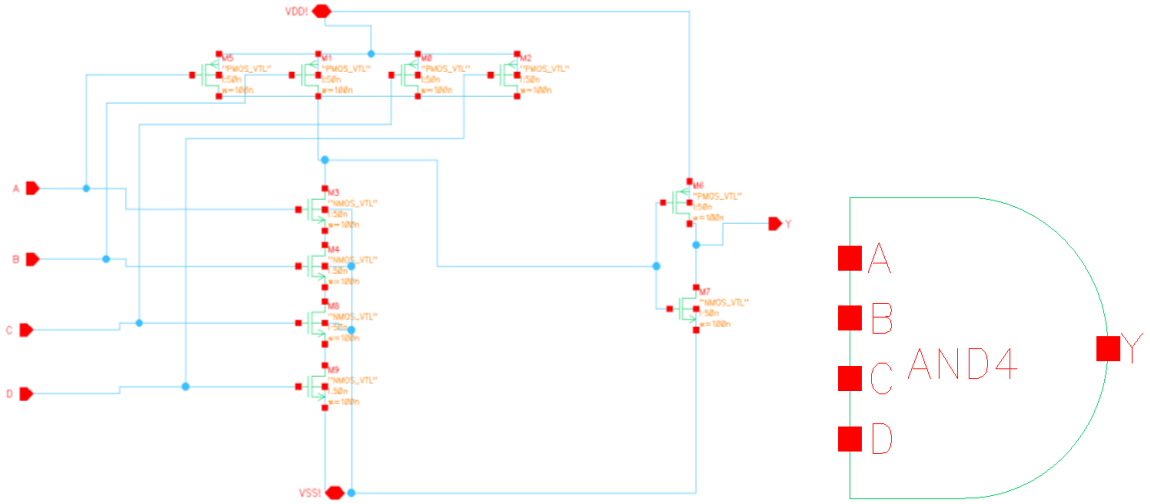


Figure 5. Transistor-level schematic and symbol representation of the AND4 gate.

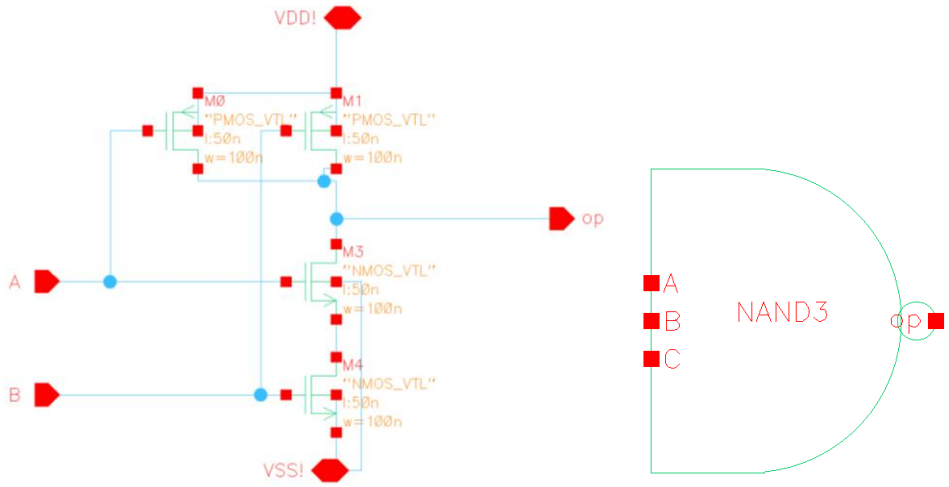


Figure 6. Transistor-level schematic and symbol representation of the NAND2 gate.

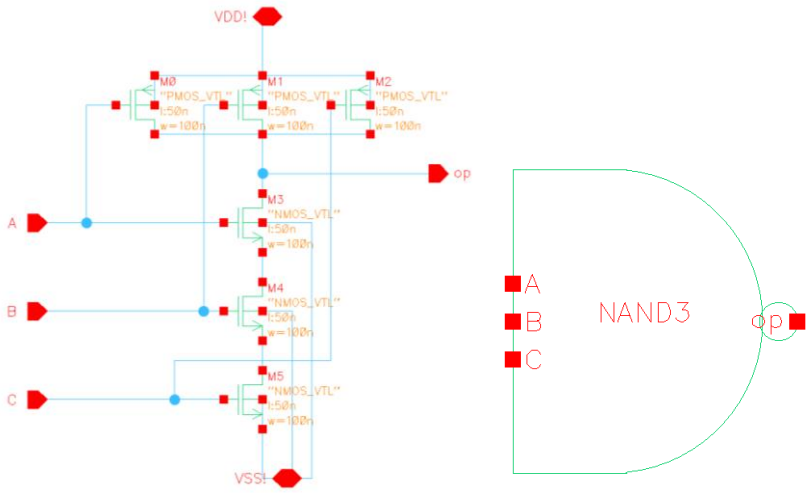


Figure 7. Transistor-level schematic and symbol representation of the NAND3 gate.

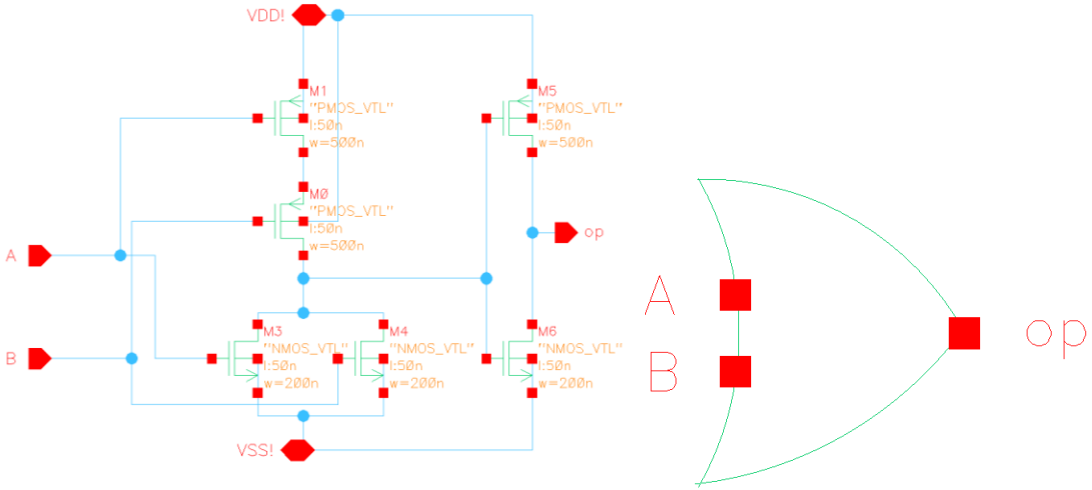


Figure 8. Transistor-level schematic and symbol representation of the OR2 gate.

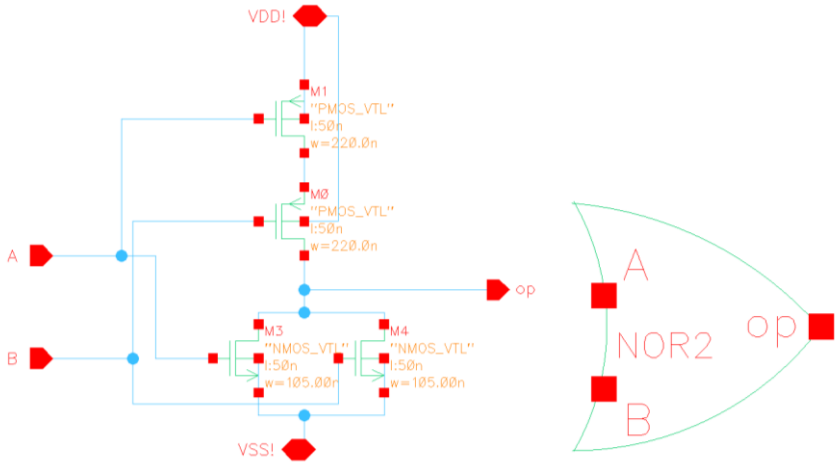


Figure 9. Transistor-level schematic and symbol representation of the NOR2 gate.

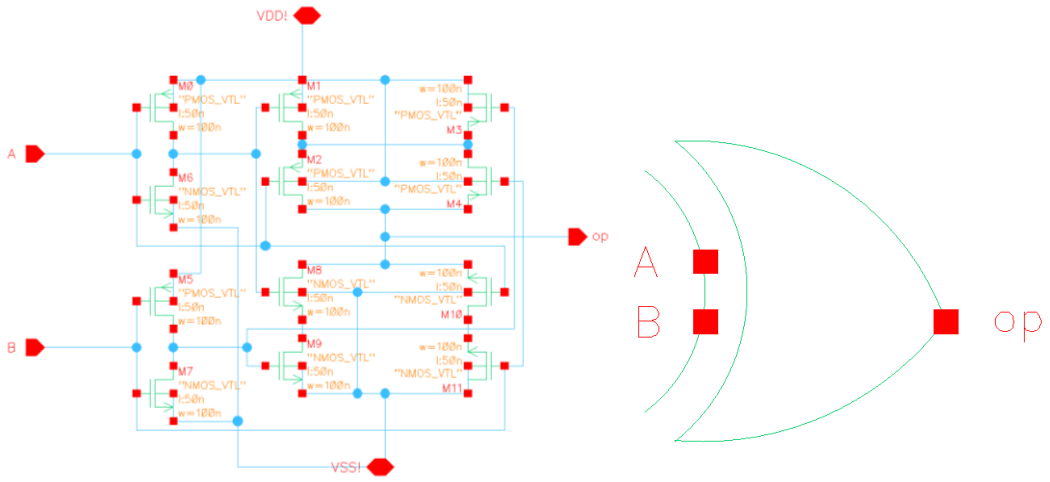


Figure 10. Transistor-level schematic and symbol representation of the XOR2 gate.

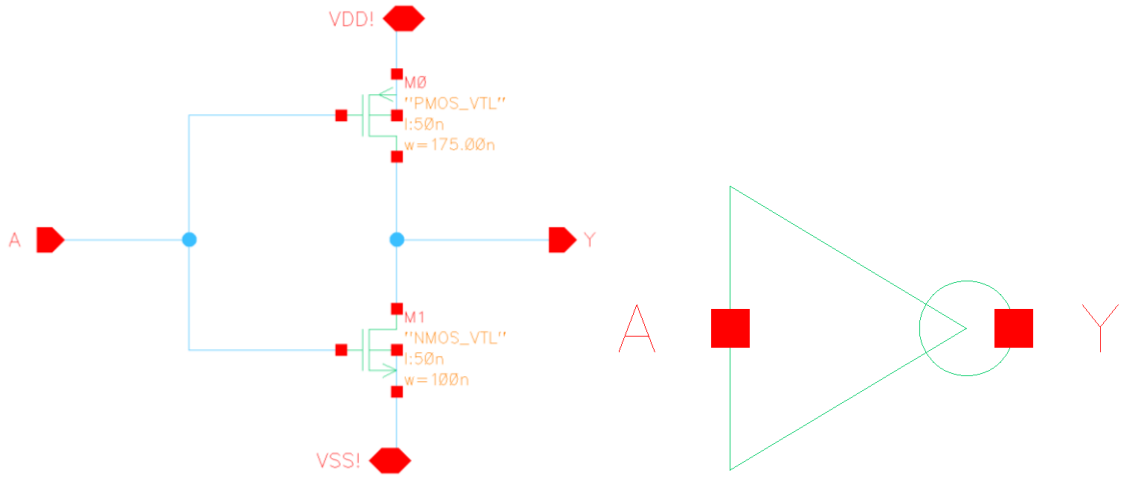


Figure 11. Transistor-level schematic and symbol representation of the inverter (INV).

B. DECODER:

- ***Module Functionality:*** The decoder and control unit are responsible for interpreting the 3-bit opcode field of the fetched instruction and generating the control signals required to configure the CPU datapath. This block serves as the primary control element of the processor by translating binary opcode information into corresponding hardware activations that control instruction execution sequencing within the CPU. In this design, the decoder operates as a 3-to-8 decoding logic, where each opcode value activates exactly one control output while all remaining control signals remain deasserted. The decoded outputs are directly mapped to named control signals such as LD_PC, LD_IR, LD_AC, MRD, and MWR, which control the ALU, registers, memory access logic, and program counter during each instruction cycle.

Table 2. Opcode-based control signal generation using a 3-to-8 decoder.

Opcode (IR[2:0])	LD_PC	LD_IR	LD_AC	MRD	MWR
000	0	0	0	0	0
001	0	0	0	0	0
010	0	0	0	0	0
011	0	0	0	0	1
100	0	0	0	1	0
101	1	0	0	0	0
110	0	1	0	0	0
111	0	0	1	0	0

The table above illustrates the 3-to-8 decoding behavior of the control unit, where each opcode asserts exactly one control signal while all others remain deasserted. Specific opcode values enable register loading, memory read/write operations, or program counter updates, while remaining opcodes correspond to unused or no-operation conditions.

- ***Architectural Specification:*** The decoder and control unit accept the 3-bit opcode output (OP<2:0>) from the instruction register as its primary input, along with clock (clk) and reset (rst) signals. Based on the decoded opcode, the control unit generates one-bit control outputs, including LD_AC, LD_IR, LD_PC, MRD, and MWR, which configure the operation of the accumulator, instruction register, program counter, and memory subsystem. Internally, the decoder logic maps each opcode to a unique control signal activation, enabling instruction-dependent control of arithmetic operations, data movement, memory access, and program counter updates. The decoder operates synchronously with the system clock to ensure stable and deterministic control sequencing across the instruction fetch, decode, execute, and program counter update stages.

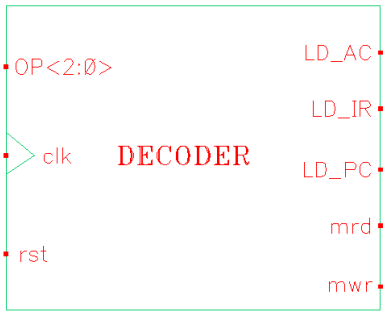


Figure 12. Decoder and control unit symbol illustrating the OP<2:0> opcode input, clock and reset inputs, and control signal outputs (LD_AC, LD_IR, LD_PC, MRD, MWR).

- ***Implementation Methodology:*** The decoder and control unit were implemented using CMOS logic in the FreePDK45 (45 nm) technology and realized as a 3-to-8 decoding structure in Cadence Virtuoso. The opcode inputs and their complements were generated using inverter stages and distributed through combinational logic to produce one-hot decoded outputs. Each decoded path activates a single control signal corresponding to a specific opcode while ensuring all other control outputs remain inactive. The decoded outputs are directly mapped to named control signals (LD_PC, LD_IR, LD_AC, MRD, and MWR) without the use of intermediate generic decoder outputs. The complete decoder schematic was constructed hierarchically using previously verified basic logic gate cells and integrated with clock and reset inputs to support synchronous control operation. This implementation enables reliable generation of control signals required for instruction execution and seamless integration with the CPU datapath.

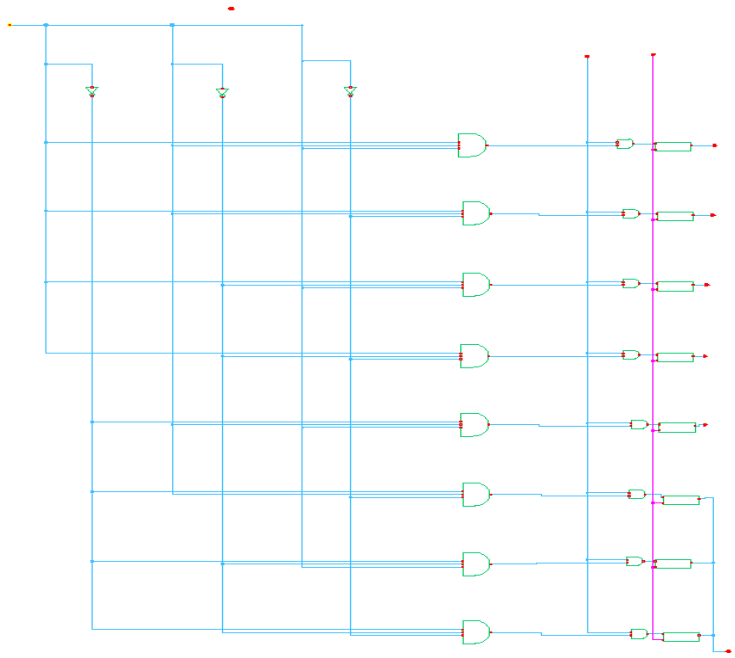


Figure 13. Schematic implementation of the 3-to-8 decoder used for control signal generation.

- ***Functional Behavior:*** During processor operation, the decoder and control unit continuously monitor the 3-bit opcode stored in the instruction register. When a valid opcode is detected, the decoder asserts the corresponding control signal according to the predefined 3-to-8 decoding logic. For each instruction cycle, only one control signal (LD_PC, LD_IR, LD_AC, MRD, or MWR) is activated, while all remaining outputs remain deasserted. These control signals configure the CPU datapath by enabling register loading, controlling memory read or write operations, and triggering program counter updates as required. All control signal transitions are synchronized with the system clock, ensuring stable operation and correct instruction sequencing.

- ***Verification and Simulation Analysis:***

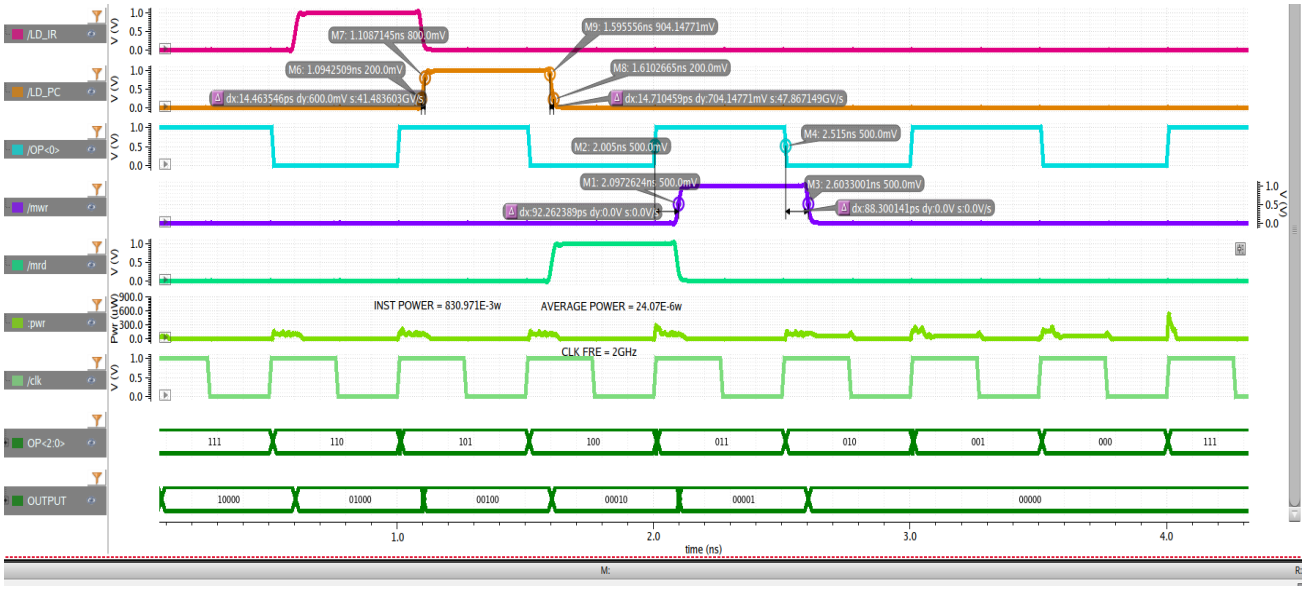


Figure 14. Simulation waveform verifying correct opcode decoding and control signal generation in the decoder and control unit.

The decoder and control unit were verified using transient simulations in the Cadence Spectre simulator under synchronous operation with an externally applied 2 GHz clock. During simulation, a sequence of 3-bit opcode values was applied to the input (OP<2:0>) while monitoring the generated control outputs (LD_AC, LD_IR, LD_PC, MRD, and MWR). The waveform demonstrates that for each opcode transition, exactly one control signal is asserted while all others remain deasserted, confirming correct 3-to-8 decoding behavior. Specifically, opcode 111 activates LD_AC, 110 activates LD_IR, 101 activates LD_PC, 100 enables memory read (MRD), and 011 enables memory write (MWR). All control signal transitions are aligned with the 2 GHz clock edges, with no overlapping activations or unintended glitches observed. These results validate correct opcode decoding, one-hot control signal generation, and reliable integration of the decoder with the synchronous CPU datapath.

- ***Performance Analysis:*** The decoder and control unit operate reliably at the target clock frequency, producing well-defined control outputs with consistent transition behavior. The measured propagation characteristics indicate that opcode decoding completes comfortably within a single clock cycle, ensuring correct control sequencing. Power consumption remains limited due to low switching activity, as control lines toggle only when opcode values change. The moderate transistor count and stable energy–delay behavior confirm that the control unit does not introduce timing or power bottlenecks in the overall CPU design.

Table 3. Performance summary of the decoder and control unit.

Frequency	Rise Time	Fall Time	Rising Propagation Delay	Falling Propagation Delay	Average Propagation Delay	Average Power	Instantaneous Power	No. of Transistors	EDP
2 GHz	14.46 ps	14.71 ps	92.26 ps	88.30 ps	90.28 ps	24.07×10^{-6} W	830.971×10^{-3} W	140	0.196 pJ·ps

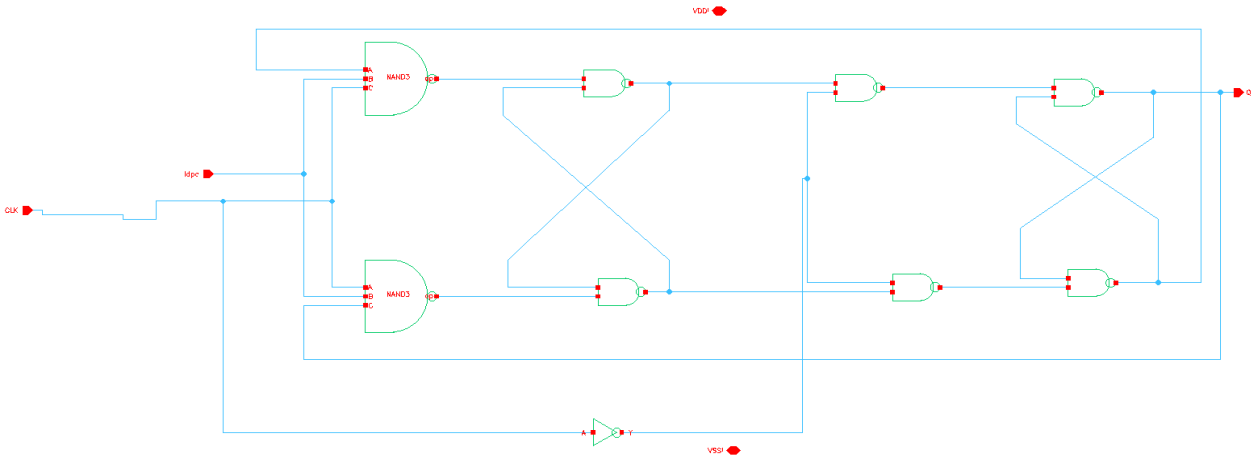
C. INSTRUCTION POINTER:

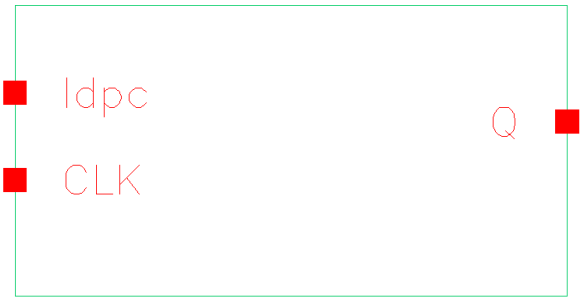
- ***Module Functionality:*** The Instruction Pointer, also referred to as the Program Counter (PC), is responsible for maintaining the address of the current instruction being executed by the processor. It ensures the sequential flow of instruction execution by updating the instruction address in a controlled manner. In this design, the program counter increments only when the control signal LD_PC is asserted, allowing the control logic to govern instruction sequencing. When LD_PC is low, the program counter does not update and maintains its stored address value. In the absence of a properly controlled instruction pointer, the processor would be unable to maintain correct instruction ordering, leading to incorrect instruction fetch behavior and loss of correct program sequencing.

- ***Architectural Specification:*** The Instruction Pointer is implemented as a 5-bit synchronous counter that generates the instruction address output AD_PC<4:0>. It accepts the system clock (CLK) and the load enable signal (LD_PC) as inputs. The output of the program counter is routed to a 5-to-1 multiplexer, which selects the appropriate address source based on control logic and forwards the selected address to the memory block. The selected address is then supplied to the instruction memory for instruction fetch operations. All state updates of the program counter occur synchronously with the active clock edge, and the use of a fixed-width counter defines the address space supported by the instruction memory.

Figure 15. Symbol representation of the Instruction Pointer (Program Counter).

- ***Implementation Methodology:*** The Instruction Pointer was designed using a hierarchical, bottom-up approach beginning with the implementation of a master–slave T flip-flop at the transistor level. The master–slave configuration ensures edge-triggered operation by isolating the input during the active clock phase and updating the output only on the clock transition, thereby eliminating race-through conditions and providing stable state transitions suitable for synchronous counter applications. The T flip-flop was realized using basic CMOS logic gates in the FreePDK45 (45 nm) technology and verified through transient simulations. The simulation waveform confirms that the flip-flop output toggles correctly on the active clock edge when the enable condition is satisfied and retains its previous state otherwise, validating its suitability as a fundamental building block for the Instruction Pointer counter.





LD_PC	CLK	Q(next)
0	↑	Q (holds previous value)
1	↑	$\bar{Q}$ (toggles)
X	0 / 1 (no edge)	Q (holds previous value)

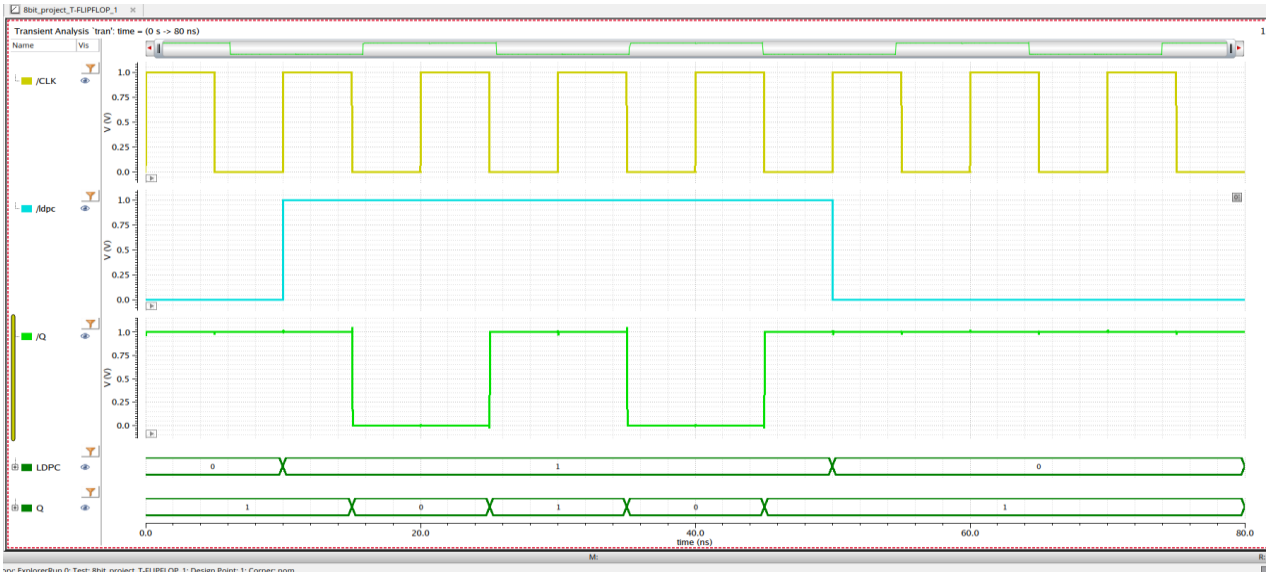


Figure 16. Symbol, transistor-level schematic, truth table and transient simulation waveform of the 1-bit master–slave T flip-flop implemented

Using the verified master–slave T flip-flop as a building block, the Instruction Pointer was constructed as a 5-bit synchronous counter. The least significant bit toggles when the control signal LD_PC is asserted, while higher-order bits toggle based on carry propagation logic implemented using basic CMOS gates. All flip-flop stages share a common clock input, ensuring synchronous updates across the counter. The outputs of the flip-flops collectively form the instruction address bus AD_PC<4:0>, which is routed through a multiplexer for address selection before being applied to the instruction memory. This implementation provides reliable and controlled instruction address incrementation while maintaining modularity and integration compatibility with the CPU control logic.

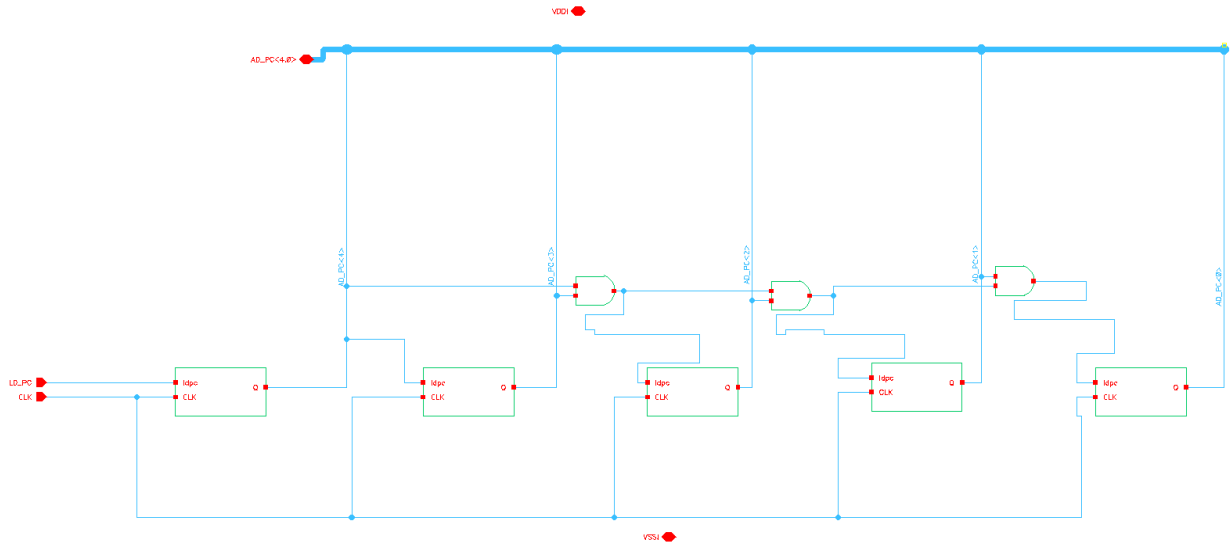


Figure 17. Transistor-level schematic of the Instruction Pointer (Program Counter)'

- ***Functional Behavior:*** During normal processor operation, the Instruction Pointer updates its output address only when LD_PC is high. On each rising edge of the clock with LD_PC asserted, the counter increments by one, advancing the instruction address. When LD_PC is low, the output address remains unchanged regardless of clock activity. This behavior allows the decoder and control unit to manage instruction sequencing precisely, ensuring that instruction fetch occurs only at the intended times.

- ***Verification and Simulation Analysis:***

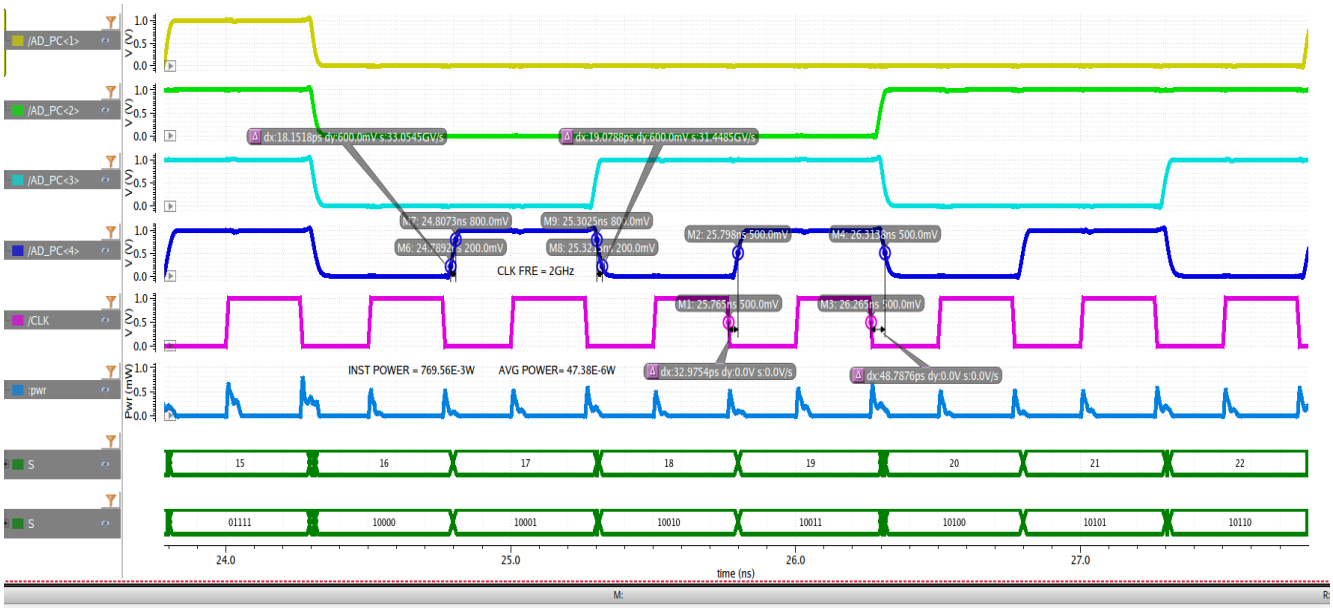


Figure 18. Simulation waveform verifying correct instruction pointer.

Functional verification of the Instruction Pointer was carried out using transient simulations in the Cadence Spectre simulator with a 2 GHz system clock applied. The simulation waveform shows that whenever the control signal LD_PC is asserted, the program counter increments synchronously on each active clock edge. As observed in the waveform, the instruction address output progresses sequentially from 15 → 16 → 17 → 18 → 19 → 20 → 21 → 22, confirming correct counter operation. This confirms correct gated incrementation controlled exclusively by LD_PC. When LD_PC is deasserted, the output address remains constant, indicating proper hold behavior. These results validate the correct synchronous increment functionality of the Instruction Pointer and its reliable operation under control of the load enable signal.

- ***Performance Analysis:*** The Instruction Pointer demonstrates stable synchronous operation with predictable timing characteristics governed by its flip-flop-based counter structure. The separation between rising and falling propagation delays reflects the internal gating and carry logic used for controlled incrementation. Since the counter updates only when enabled, unnecessary switching is minimized, resulting in moderate power consumption. Overall, the energy–delay behavior confirms efficient instruction sequencing without compromising timing reliability.

Table 4. Performance summary of instruction pointer

Frequency	Rise Time	Fall Time	Rising Propagation Delay	Falling Propagation Delay	Average Propagation Delay	Average Power	Instantaneous Power	Number of Transistors	EDP
2 GHz	18.15ps	19.07ps	32.97ps	48.78ps	40.72ps	47.38 x 10^{-6} W	769.56 x 10^{-3} W	56	0.0786 pJ·ps

D. ACCUMULATOR:

- ***Module Functionality:*** The 8-bit accumulator is a sequential storage block used to hold intermediate and final computation results generated by the Arithmetic Logic Unit (ALU). It captures the 8-bit ALU output when the load enable signal (LD_AC) is asserted and retains the stored value across clock cycles when LD_AC is deasserted. The accumulator provides a stable storage location for arithmetic and logical results, enabling sequential execution of instructions within the CPU datapath.

- ***Architectural Specification:*** The 8-bit accumulator accepts an 8-bit input bus from the ALU (ALU_OUT<7:0>), along with a system clock (CLK), load enable signal (LD_AC), and reset signal (RST). The output of the accumulator is provided on the 8-bit bus AC_OUT<7:0>, which is used by subsequent datapath elements. All accumulator bits operate synchronously on the same clock and control signals, ensuring uniform update behavior across the entire word.

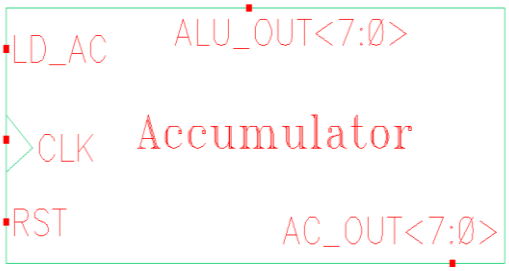
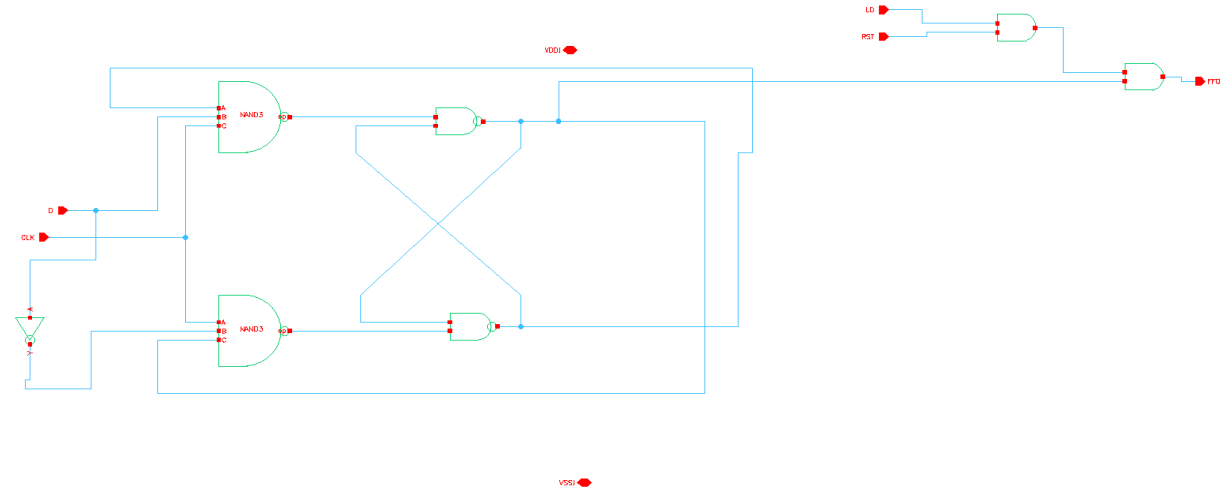


Figure 19. Symbol representation of 8-bit accumulator

- ***Implementation Methodology***: The design of the 8-bit accumulator followed a hierarchical, bottom-up methodology beginning with the implementation and verification of a 1-bit accumulator cell. Each 1-bit cell is based on a master–slave D flip-flop with load enable (LD) and reset (RST), implemented at the transistor level in FreePDK45 (45 nm) CMOS technology. Transient simulation results confirm that when RST is high, the output is cleared to logic zero. When RST is low and LD is high, the accumulator loads the input data on the active clock edge. When both RST and LD are low, the output remains at logic zero.



LD	RST	D (Input)	Q (Output)	Operation
0	0	X	0	Reset state
0	1	X	0	Output forced to zero
1	0	X	0	Reset active
1	1	0	0	Load input value
1	1	1	1	Load input value

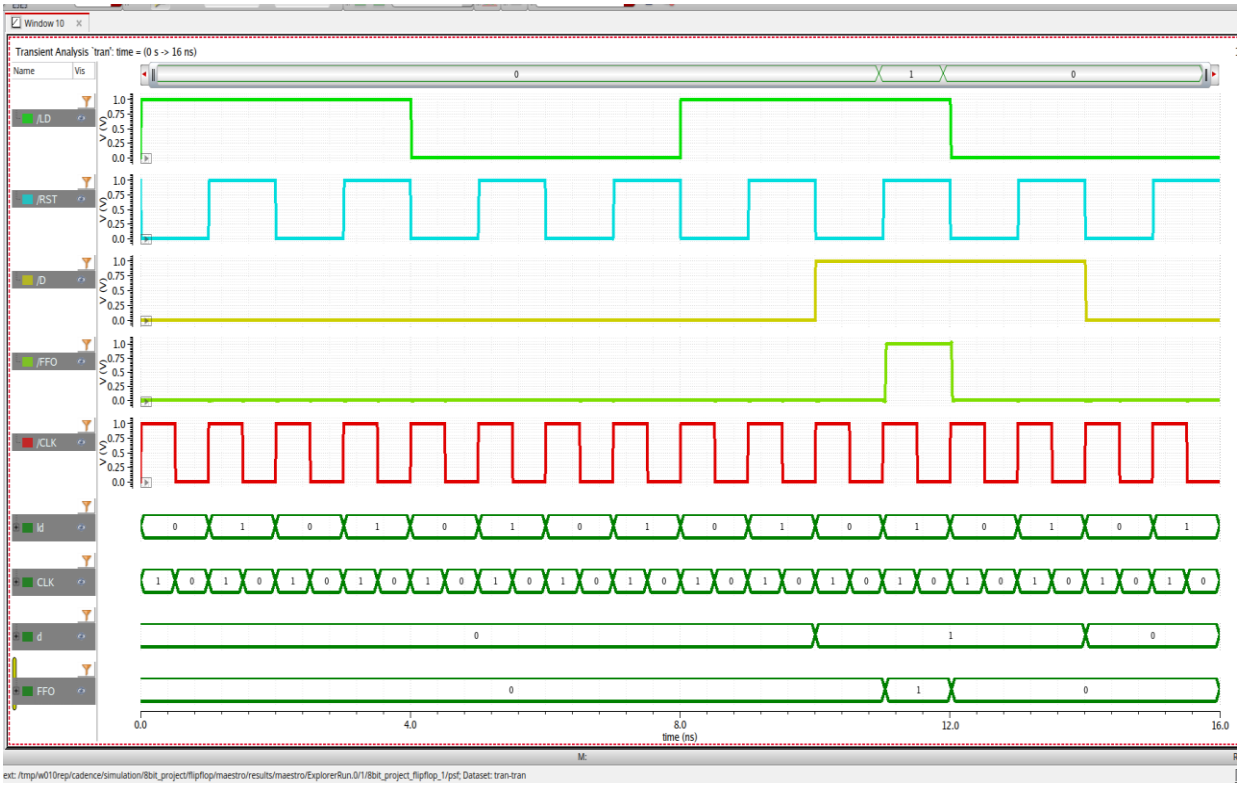


Figure 20. Symbol representation, transistor-level schematic, truth table, and transient simulation waveform of the 1-bit accumulator illustrating load and reset behavior.

After successful verification of the 1-bit accumulator, eight identical instances of the verified cell were instantiated in parallel to form the 8-bit accumulator. All accumulator bits share common control signals, namely LD_AC, CLK, and RST, while each bit receives its corresponding input from the ALU output bus. This hierarchical replication approach enables effective reuse of the validated 1-bit design and ensures uniform timing behavior and consistent functional operation across all bits of the accumulator.

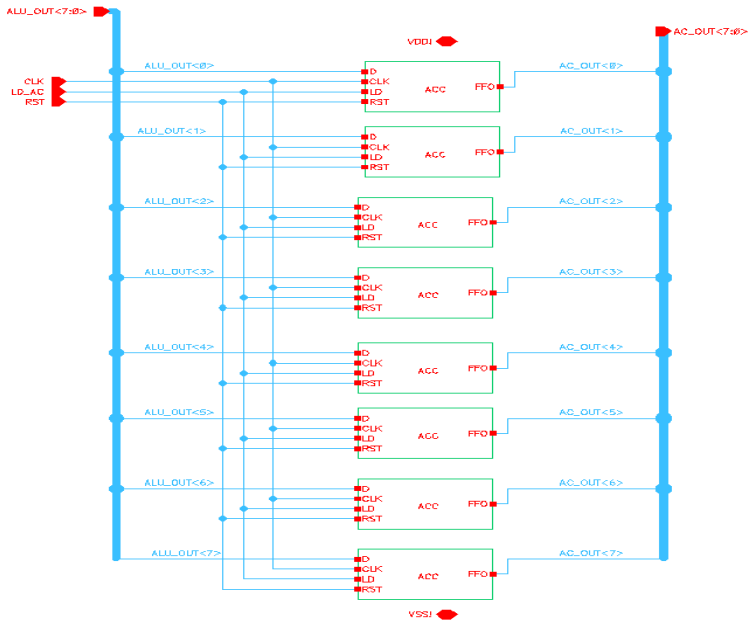


Figure 21. Transistor-level schematic of the 8-bit accumulator constructed using eight identical 1-bit accumulator cells.

- ***Functional Behavior:*** During operation, when LD_AC is asserted, the accumulator captures the 8-bit ALU output on the active clock edge and updates AC_OUT<7:0> accordingly. When LD_AC is deasserted, the accumulator holds the previously stored value regardless of changes at the ALU output. When the reset signal is asserted, all accumulator bits are cleared to a known logic state. All state transitions occur synchronously with the system clock, ensuring stable and coordinated storage behavior across the entire accumulator word.

- ***Verification and Simulation Analysis:***

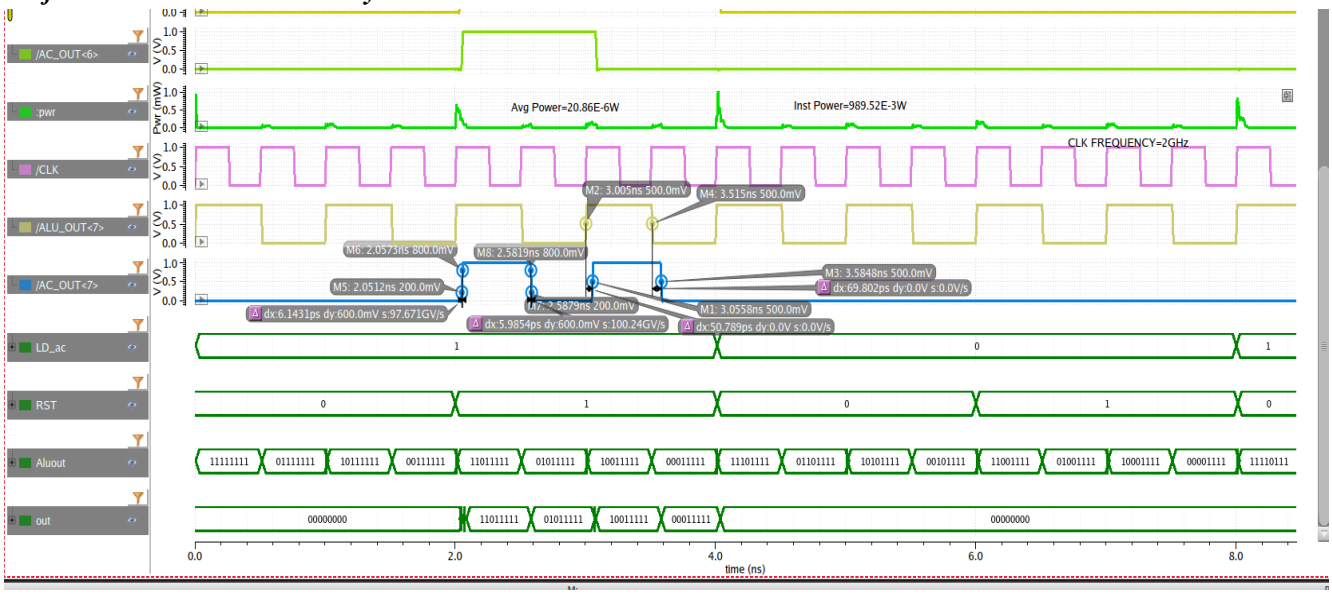


Figure 22. Simulation waveform verifying correct load, hold, and reset operation of the 8-bit accumulator under a 2 GHz clock.

Functional verification of the 8-bit accumulator was performed using transient simulations in the Cadence Spectre simulator with an externally applied 2 GHz clock. During simulation, different ALU output patterns were applied while controlling the LD_AC and RST signals. As observed in the waveform, when the ALU output is 11111111 and LD_AC = 0, the accumulator output remains at 00000000, indicating correct hold behavior. When LD_AC is set to 1 with RST = 0 and the ALU output changes to 11011111, the accumulator captures the input correctly and updates AC_OUT<7:0> to 11011111 on the active clock edge. The output transitions occur synchronously across all bits, and proper reset behavior is observed when RST is asserted. These results confirm correct multi-bit load, hold, and reset functionality of the 8-bit accumulator.

- ***Performance Analysis:*** The accumulator exhibits consistent timing behavior across all bits due to its replicated 1-bit storage architecture. Propagation delays indicate controlled data capture during load operations, while non-load cycles show minimal switching activity. Power dissipation is primarily associated with active load events, demonstrating efficient use of clocked storage. The measured energy–delay characteristics show that the accumulator is well suited for frequent arithmetic result storage within a synchronous datapath.

Table 5. Performance summary of Accumulator

Frequency	Rise Time	Fall Time	Rising Propagation Delay	Falling Propagation Delay	Average Propagation Delay	Average Power	Instantaneous Power	No. of Transistors	EDP
2Ghz	6ps	6ps	48.46ps	69.80ps	59.13ps	20.86 $\times 10^{-6}$ W	1.01 $\times 10^{-3}$ W	226	0.0729 pJ·ps

E. 8-BIT REGSTER:

- ***Module Functionality:*** The 8-bit register is used to store and synchronize multi-bit data within the CPU datapath. Its primary role is to capture the output of the ALU on a clock edge and provide a stable, glitch-free output to subsequent blocks. By registering the ALU outputs, transient glitches produced by combinational logic are eliminated, ensuring clean and reliable data transfer across clock cycles.

- ***Architectural Specification:*** The 8-bit register consists of eight identical 1-bit D flip-flop cells connected in parallel. Each flip-flop receives a corresponding input bit from the data bus D<7:0> and produces the registered output Q<7:0>. All flip-flops share a common clock signal (CLK), ensuring simultaneous data capture across all bits. The register has no internal combinational logic between stages, providing uniform timing behavior for all bits.

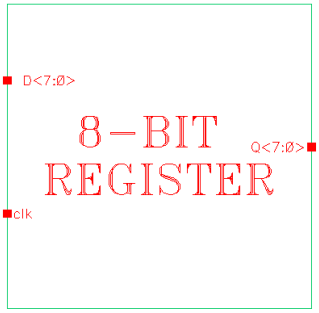
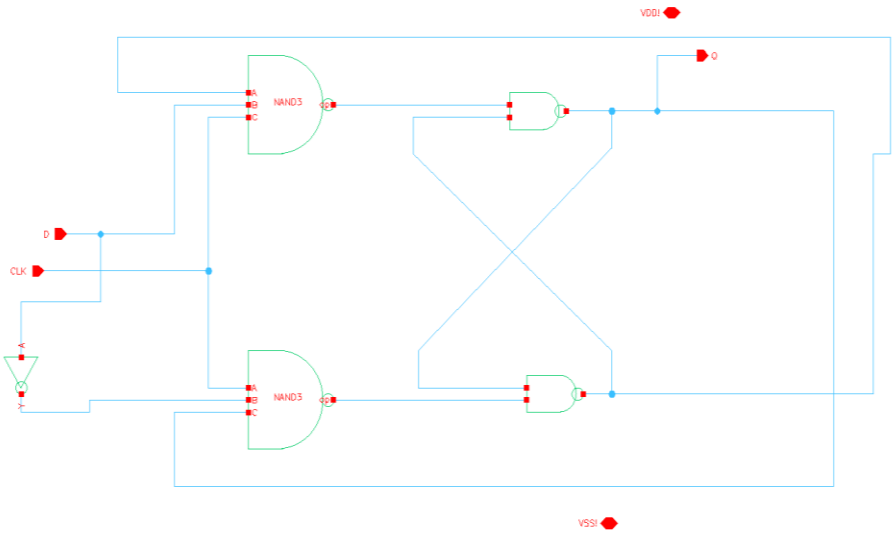


Figure 23. Symbol representation of 8-bit Register

- ***Implementation Methodology:*** The design was carried out using a hierarchical, bottom-up approach. First, a 1-bit NAND-based D flip-flop was implemented at the transistor level using FreePDK45 (45 nm) CMOS technology in Cadence Virtuoso. This flip-flop was verified through schematic inspection, symbol creation, and transient simulation. After successful verification, eight instances of the same 1-bit flip-flop were instantiated in parallel to construct the 8-bit register. This modular replication ensured design consistency, simplified debugging, and enabled reuse of a verified storage element.



CLK	D	Q (Next)
↑	0	0
↑	1	1
0/1 (no edge)	X	Q (previous)

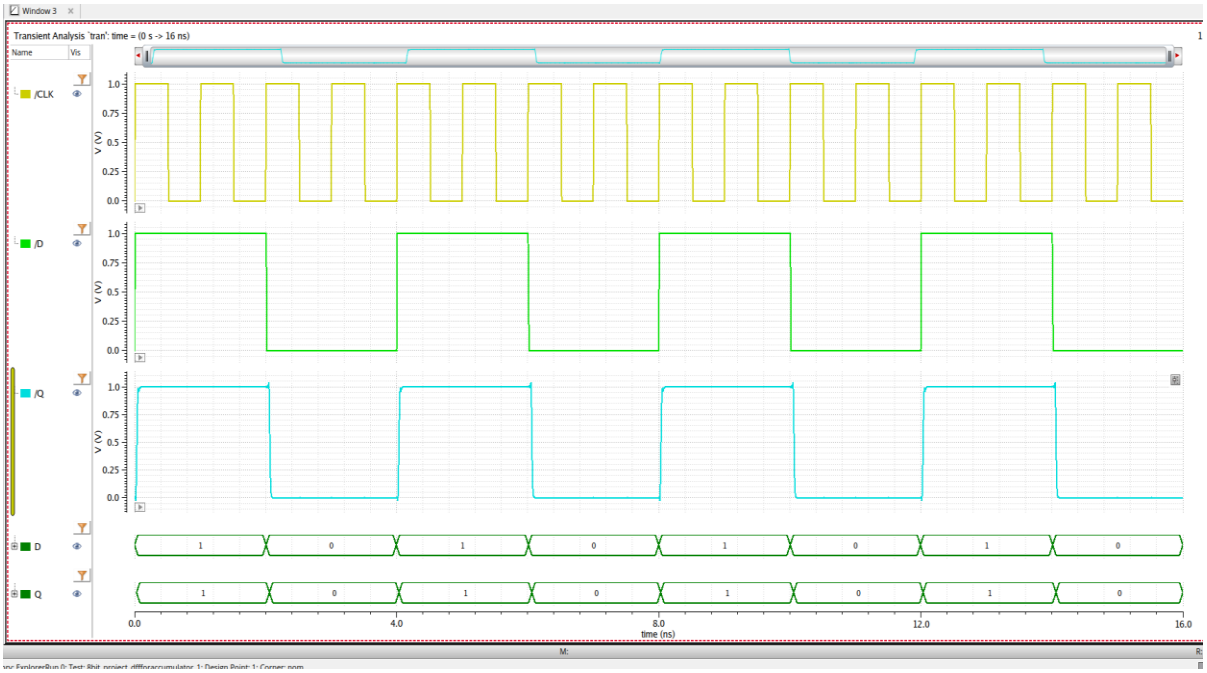


Figure 24. Symbol representation, transistor-level schematic, truth table, and transient simulation waveform of the 1-bit D-Flip Flop.

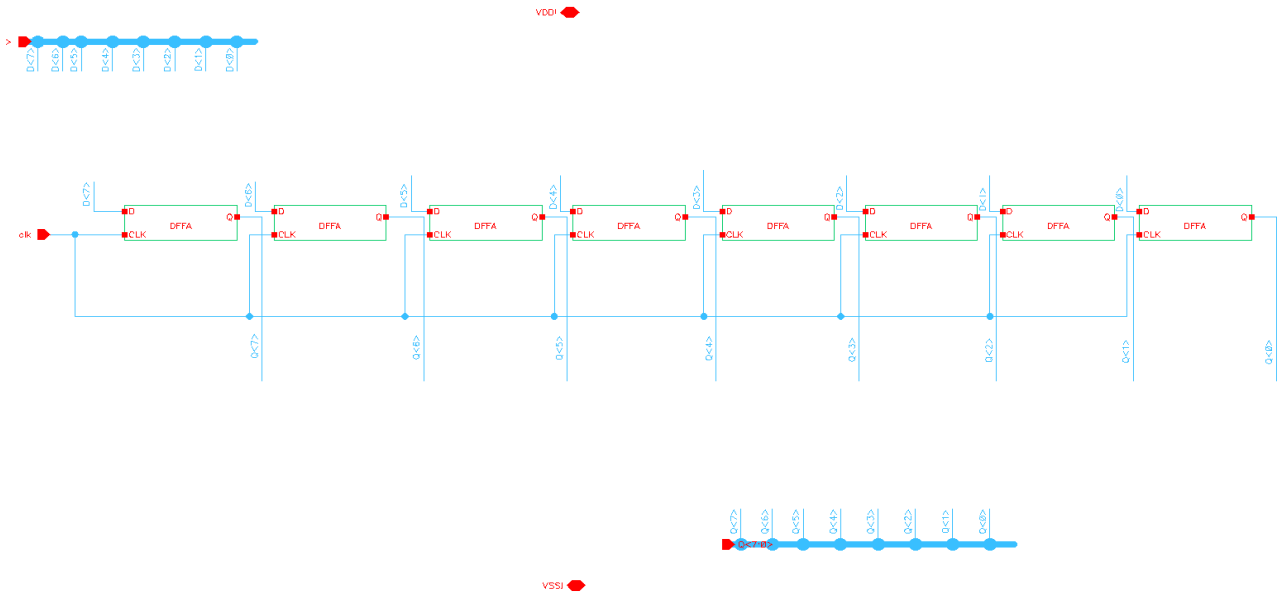


Figure 25. Schematic of the 8-bit register constructed using eight D flip-flop instances

- ***Functional Behavior:*** During operation, the 8-bit register samples the input data D<7:0> on the active clock edge and transfers it to the output Q<7:0>. Between clock edges, the stored data remains unchanged, providing stable outputs. Because the register isolates the ALU outputs from downstream logic, any intermediate switching activity at the ALU output does not propagate forward, effectively removing glitches from the datapath.

- ***Verification and Simulation Analysis:***

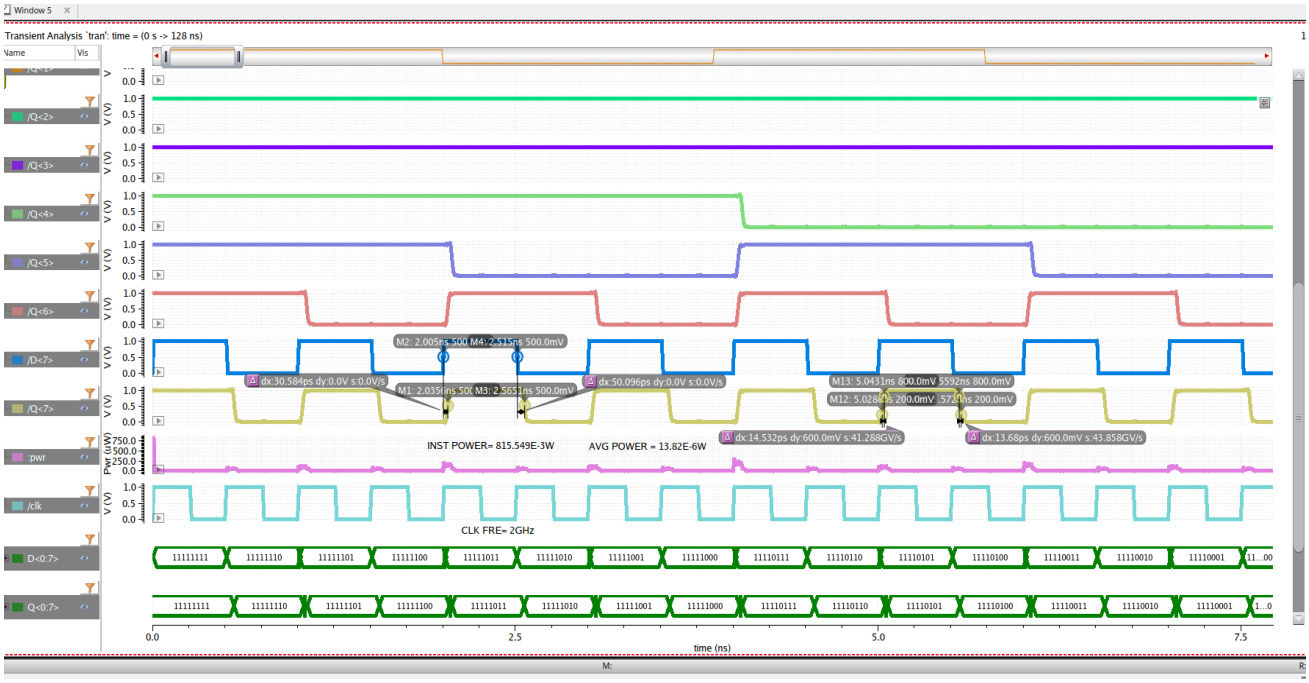


Figure 26. Simulation waveform verifying correct 8-bit register operation at 2 GHz

Functional verification of the 8-bit register was performed using transient simulations in the Cadence Spectre simulator with a 2 GHz clock. During simulation, different data patterns were applied to the input bus D<7:0>, and the corresponding outputs Q<7:0> were observed. When the clock is high, an input of D<7:0> = 11111111 results in Q<7:0> = 11111111 at the clock

edge. Similarly, applying D<7:0> = 11111110 produces Q<7:0> = 11111110, and D<7:0> = 11111011 results in Q<7:0> = 11111011. The simulation waveforms confirm that each output bit updates only on clock edges and retains its value between transitions. This behavior verifies correct multi-bit registration, synchronous operation, and effective suppression of combinational glitches originating from the ALU.

- ***Performance Analysis:*** The 8-bit register achieves reliable high-speed operation by sampling input data strictly on clock edges and isolating downstream logic from combinational switching. Balanced transition behavior across all bits confirms uniform timing due to parallel flip-flop instantiation. Power consumption remains controlled, as internal nodes switch only during clock-triggered data capture. The compact transistor count and favorable energy–delay characteristics indicate an efficient storage element suitable for stabilizing ALU outputs in the CPU datapath.

Table 6. Performance summary of 8-bit register

Frequency	Rise Time	Fall Time	Rising Propagation Delay	Falling Propagation Delay	Average Propagation Delay	Average Power	Instantaneous Power	No. of Transistors	EDP
2GHz	14.532ps	13.68ps	30.58ps	50.09ps	40.33ps	13.82×10^{-6} W	815.549×10^{-3} W	176	0.0225 pJ·ps

F. Multiplexer (MUX):

- ***Module Functionality:*** The multiplexer is used to select one of multiple address sources and route it to the memory block input based on a control signal. In this design, the MUX selects between the instruction memory address lines and the instruction pointer address lines. The selected output address is forwarded to the memory block for instruction fetch or data access.

- ***Architectural Specification:***

The implemented multiplexer is a 5-bit wide, 2-to-1 multiplexer, operating on address buses.

- Inputs:
  - Instruction Memory Address: A0<4:0>
  - Instruction Pointer Address: A1<4:0>
- Select Line:
  - Opcode / Select Signal: S
- Output:
  - Memory Address Output: OP<4:0>

Each bit of the output bus is generated by an identical 2-to-1 multiplexer cell, ensuring uniform timing behavior across all five bits.

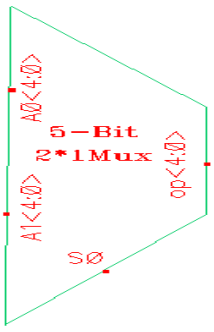
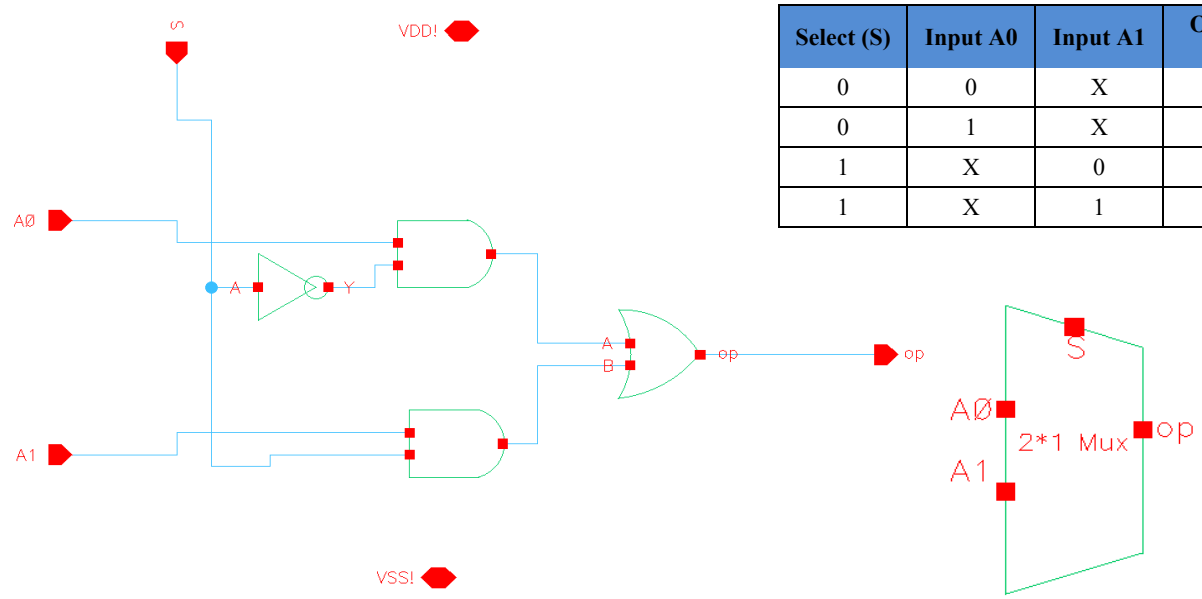


Figure 27. Symbol representation of 5-bit 2 X 1 Multiplexer

- ***Implementation Methodology:*** The multiplexer design followed a bottom-up hierarchical approach. First, a 2-to-1 multiplexer was implemented at the transistor level using basic CMOS logic gates, including NOT, AND, and OR gates, in FreePDK45 (45 nm) technology. The functionality of the 2-to-1 MUX was verified using transient simulations.

Select (S)	Input A0	Input A1	Output (OP)
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1



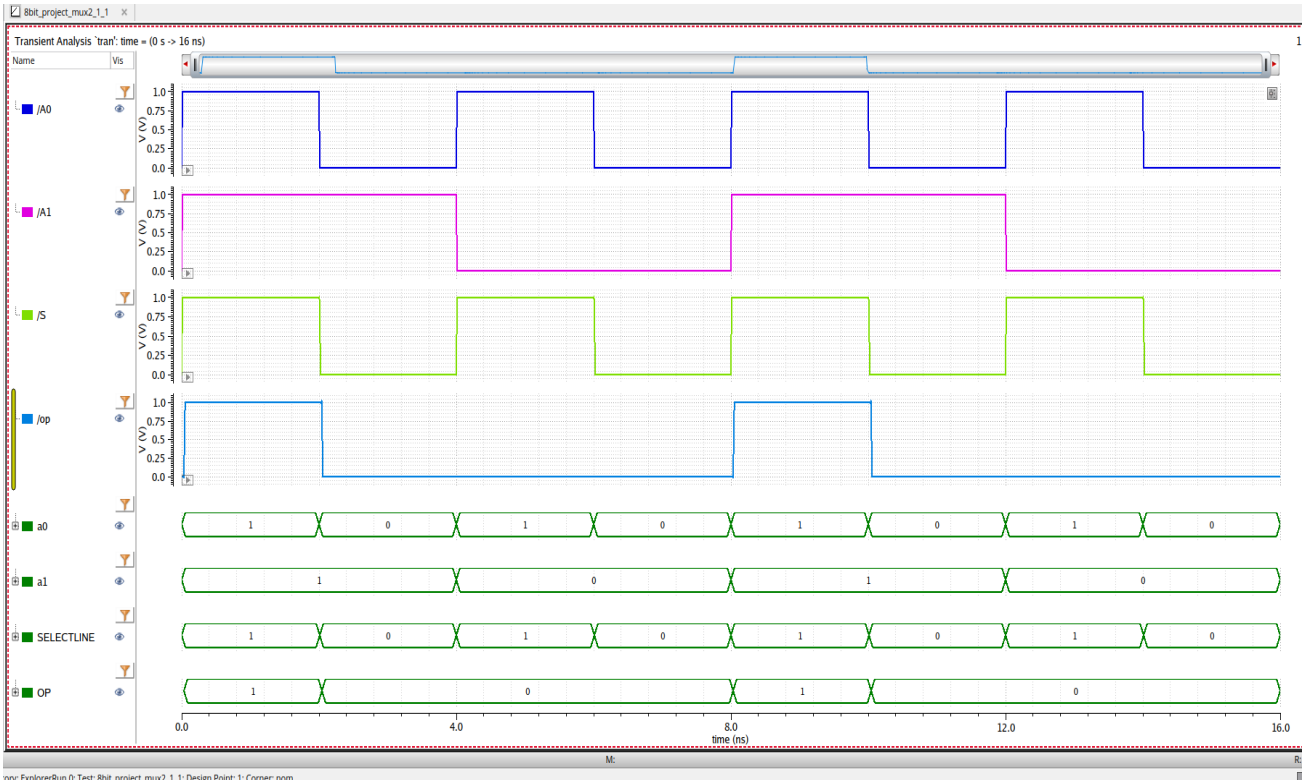


Figure 28. Symbol representation, transistor-level schematic, truth table, and transient simulation waveform of the 2 X 1 Multiplexer.

After successful verification, five identical 2-to-1 MUX instances were placed in parallel to construct the 5-bit wide multiplexer. The select line is shared across all bits, while each bit processes its corresponding input signals independently. This modular approach simplifies verification and ensures scalability.

Select (S)	A0<4:0>	A1<4:0>	OP<4:0> (Output)
0	0	XXXXX	0
0	10101	XXXXX	10101
0	11111	XXXXX	11111
1	XXXXX	0	0
1	XXXXX	1010	1010
1	XXXXX	11111	11111

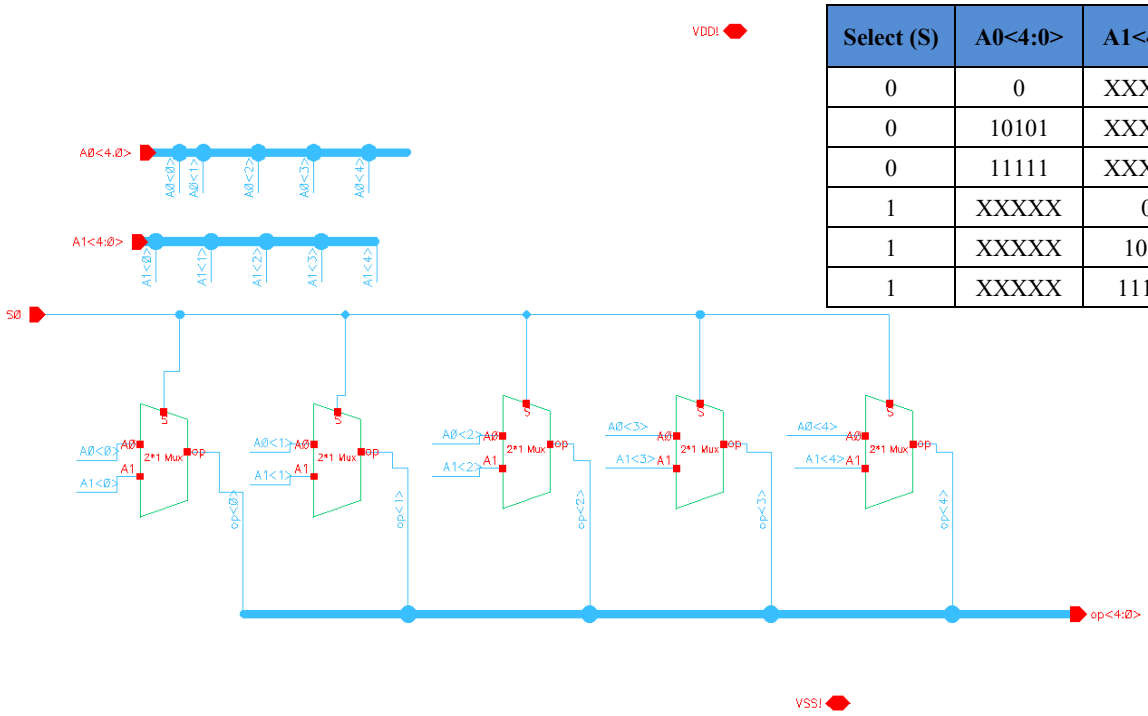


Figure 29. Schematic of the 5-bit 2 X 1 Multiplexer

- ***Functional Behavior:*** When the select signal S is low, the multiplexer forwards the instruction memory address A0<4:0> to the output OP<4:0>. When S is high, the instruction pointer address A1<4:0> is selected and routed to the output. The switching behavior occurs purely through combinational logic , and the output reflects the selected input without requiring a clock signal.

- ***Verification and Simulation Analysis:***

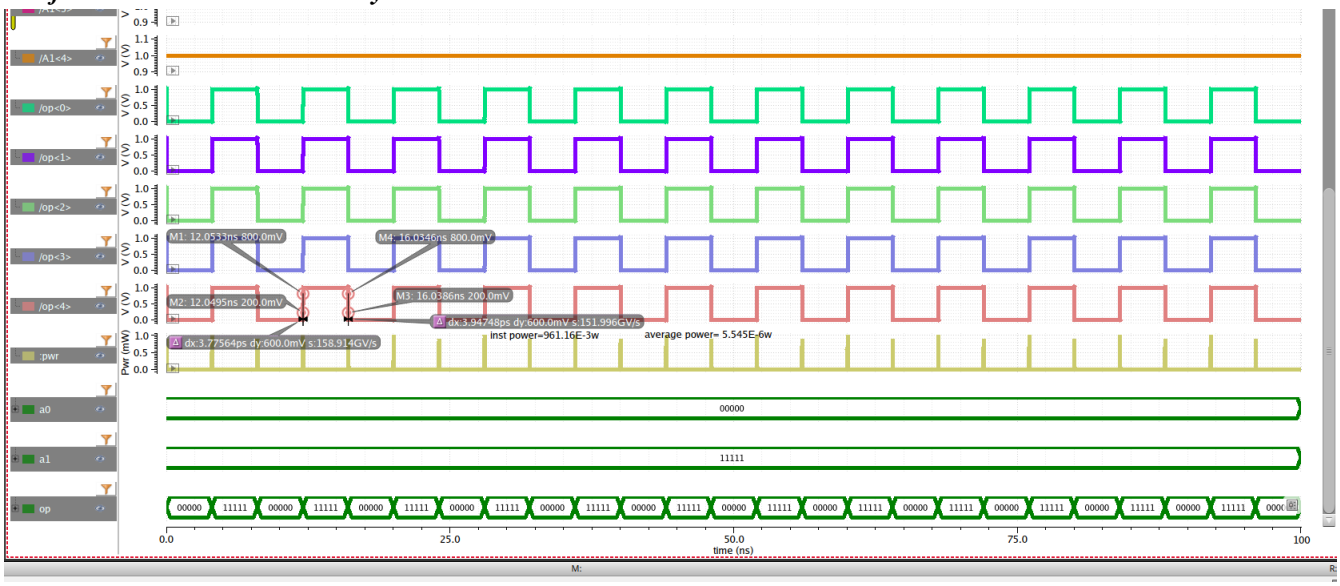


Figure 30. Simulation waveform verifying 5-bit 2 X 1 Multiplexer

Functional verification was performed using transient simulations in the Cadence Spectre simulator. Various combinations of the input buses A0<4:0>, A1<4:0>, and the select line S were applied to validate correct multiplexer operation. When the select line S is set to 0, the output OP<4:0> follows the A0<4:0> input. For example, with A0<4:0> = 00000000 and A1<4:0> = 11111111, the output remains OP<4:0> = 00000000. When the select line S is set to 1, the output switches to follow A1<4:0>, resulting in OP<4:0> = 11111111. The simulation waveforms confirm correct multi-bit selection and consistent behavior across all address bits.

- ***Performance Analysis***: The 5-bit 2×1 multiplexer exhibits very fast switching characteristics, with short rise and fall transitions suitable for high-speed digital operation. Power measurements show efficient behavior, with low average consumption during steady operation and brief peaks during switching events. The compact transistor count contributes to reduced area and minimized loading at the input nodes. Overall, the multiplexer meets the timing and power requirements for reliable address selection within the CPU architecture.

Table 7. Performance summary of 5-bit 2 X 1 Multiplexer

Rise Time	Fall Time	Average Power	Instantaneous Power	No of Transistors
3.77ps	3.94ps	5.54E× 10^{-6} W	9.61× 10^{-1} W	45

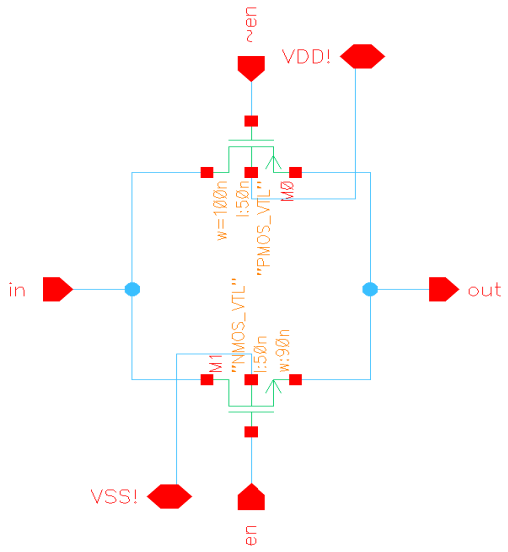
G. CLOCK GENERATOR - Ring Oscillator:

- ***Module Functionality:*** The ring oscillator generates a periodic waveform using an odd number of inverting stages connected in a closed loop. The continuous toggling of node voltages produces a free-running oscillation that demonstrates internal clock-generation behavior at the transistor level. In this project, the ring oscillator was implemented to study inverter-chain dynamics, startup conditions, and delay-dependent frequency characteristics in 45 nm CMOS. The design illustrates how device-level propagation delays directly determine oscillation frequency. Although functional for simulation, such oscillators naturally exhibit frequency variation; therefore, a PLL (Phase-Locked Loop) is typically required in practical ICs to stabilize and regulate the generated clock.

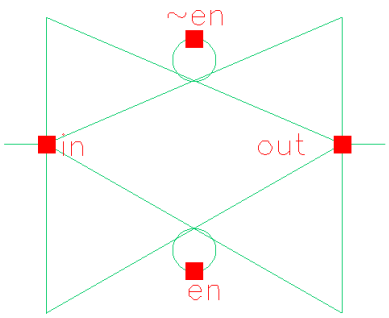
- ***Architectural Specification:*** The oscillator is organized as a 23-stage inverter loop, where the odd number of inversions ensures sustained oscillation once a disturbance is introduced. A CMOS transmission gate is placed at the loop input as an enable control. When EN = 1 and $E\overline{N}$ = 0, the feedback path is closed, completing the inverter chain and allowing oscillation. When EN = 0, the loop is interrupted and the circuit settles to a static state. Because ideal transistor-level simulations require a non-zero initial condition to break symmetry, a small seed pulse (≈1.8 ns) was applied at startup. After receiving this disturbance, the chain naturally develops a periodic output whose frequency is determined by the cumulative delay of the 23 inverters.

Figure 31. Symbol representation of Ring Oscillator

- ***Implementation Methodology:*** The design began with the construction of a CMOS transmission gate used to control the feedback path. The transmission gate consists of parallel NMOS and PMOS devices driven by complementary control signals (EN and $E\bar{N}$). When enabled, it passes the seed signal with full voltage swing, making it well-suited for initiating oscillation. Although pass-transistor logic can also be used, the CMOS transmission gate provides better noise margin and reliable logic-level transfer. Once the enable block was verified, the 23-stage inverter chain was implemented at the transistor level using FreePDK45 (45 nm) CMOS models. The inverters were arranged in a ring, and the transmission gate was placed at the entry point of the loop. Transient simulations were used to confirm correct startup, signal propagation, and sustained oscillation.



EN	EN′	TG State	Function
1	0	**ON (Conducting)**	Input is passed to output
0	1	**OFF (Isolated)**	Output is disconnected from input



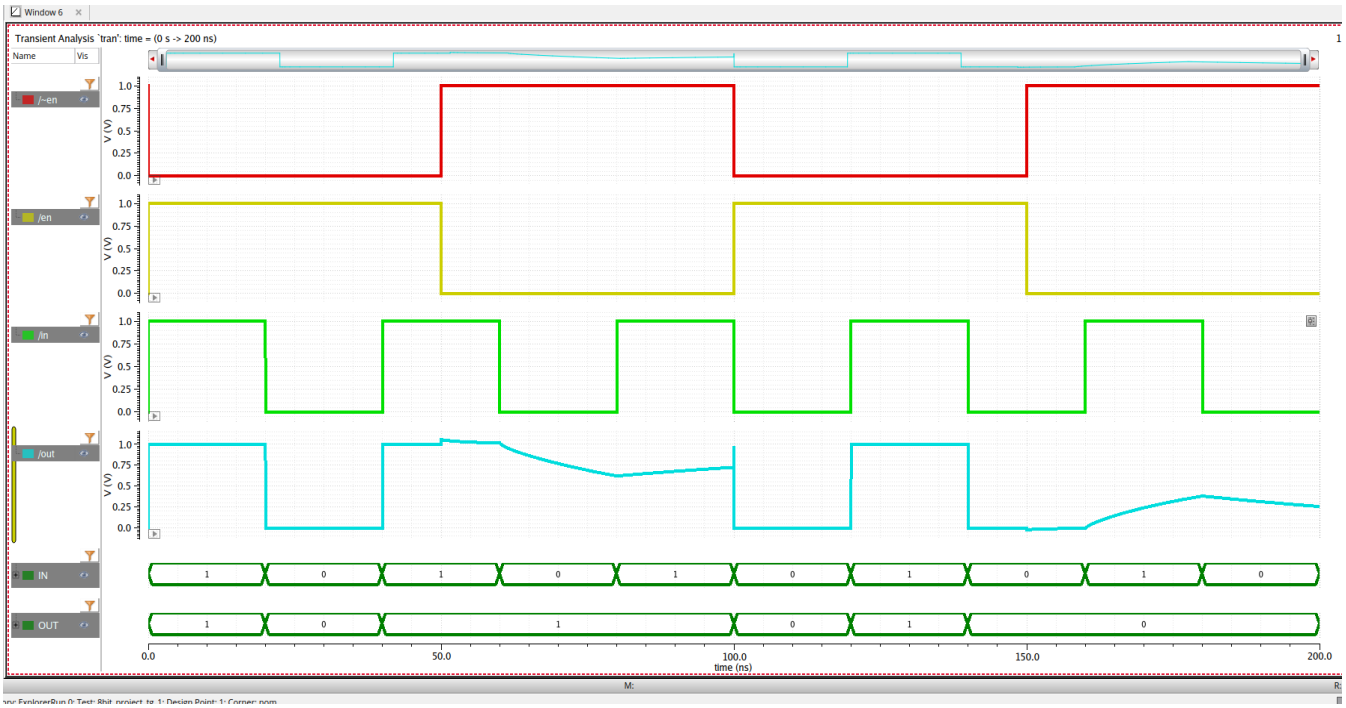


Figure 32. Symbol representation, transistor-level schematic, truth table, and transient simulation waveform of Transmission Gate

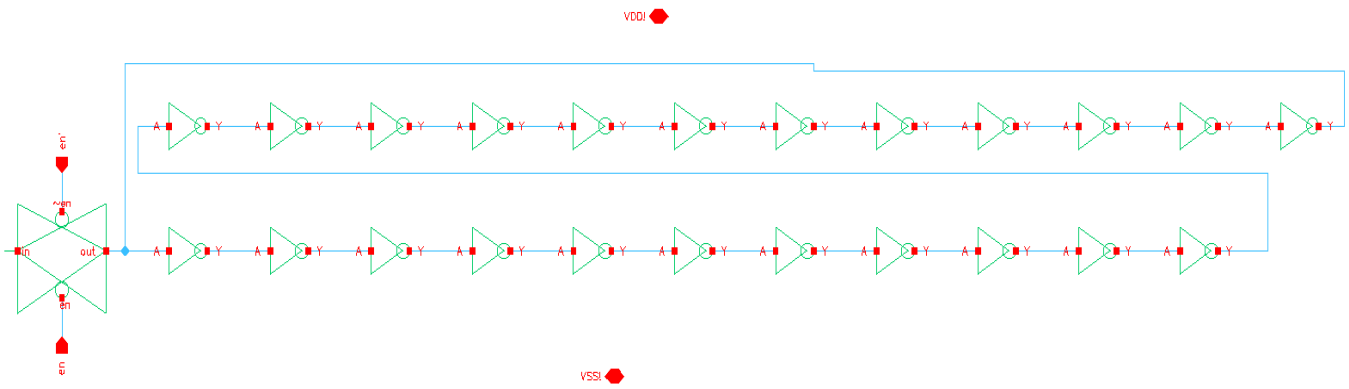


Figure 33. Schematic of Ring Oscillator

- ***Functional Behavior:*** The transmission gate functions as the enable mechanism for the oscillator by controlling whether the inverter loop is electrically closed or open. When **EN = 1** and **E$\bar{N}$ = 0**, the transmission gate conducts and completes the feedback path, allowing the inverter chain to begin oscillating. When **EN = 0** and **E$\bar{N}$ = 1**, the gate turns off, the loop is interrupted, and the circuit settles into a static, non-oscillating state. Once enabled, the injected seed pulse propagates through the chain of inverters, and the alternating inversions at each stage gradually build into a full-swing periodic waveform. After the initial startup interval, the oscillator reaches steady state, where each stage exhibits consistent high-to-low and low-to-high transitions, demonstrating proper delay accumulation and sustained oscillatory behavior.

- ***Verification and Simulation Analysis:***

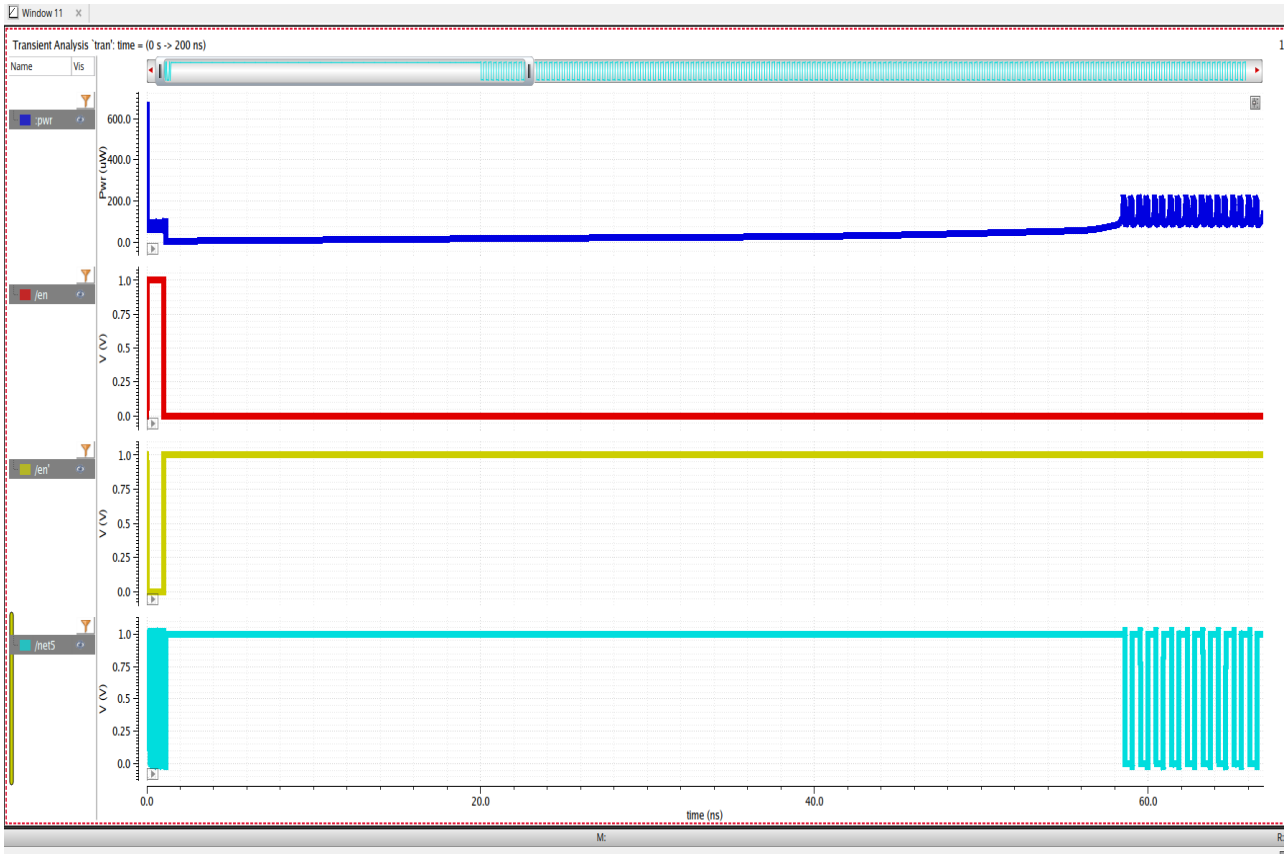


Figure 34. Simulation waveform of Ring Oscillator

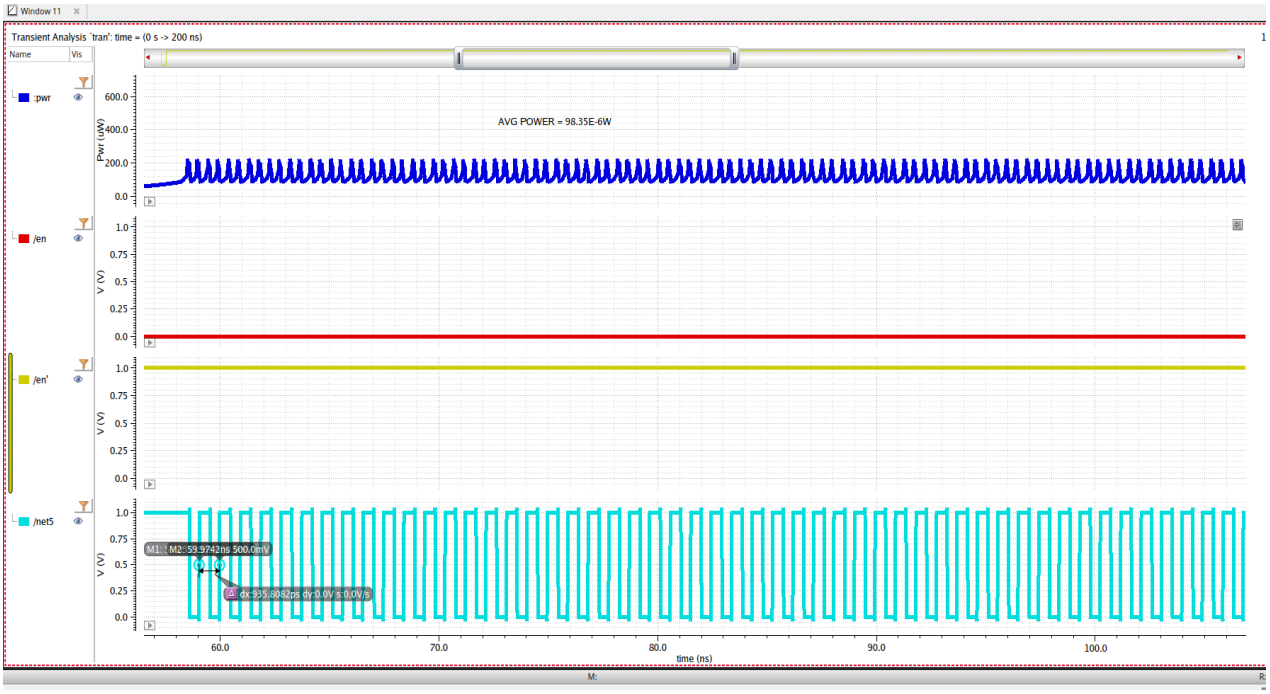


Figure 35. Simulation waveform of Ring Oscillator

Transient simulations were performed in Cadence Spectre to verify the operation of the 23-stage ring oscillator. During simulation, the enable signals were first set to EN = 1 and $E\bar{N}$ = 0, which turned on the transmission gate and allowed a small seed disturbance (a short 1.8 ns pulse) to enter the loop. This disturbance was necessary to break the initial steady state and start the oscillation. When the enable was low, the circuit remained completely static, confirming that the gating logic behaved correctly.

After enabling, the waveform showed a settling region of roughly 58 ns, where internal nodes gradually charged due to inverter delays and capacitive loading. Following this period, the output at *net5* developed into a stable periodic oscillation with full-swing transitions. The power waveform exhibited the same trend: low power during the idle phase and higher dynamic power once oscillation began. The measured average power was approximately 98.35 µW, consistent with the switching activity of a 23-stage inverter chain. Overall, the simulation verified proper oscillator startup, oscillation formation, and steady-state behavior. Small variations in frequency were also observed, which is typical for a free-running ring oscillator and is one reason why practical designs often include a PLL for frequency stabilization.

- ***Performance Analysis:*** The 23-stage ring oscillator was analyzed using transient simulations in 45 nm CMOS technology. The measured oscillation period is approximately 0.936 ns, corresponding to an operating frequency of 1.07 GHz. Once the enable signal is activated, the circuit enters steady oscillation after a brief start-up transient. The waveform shows consistent high-to-low and low-to-high transitions across all stages, confirming proper propagation through the inverter chain. Overall, the ring oscillator achieves full-swing periodic operation and generates a reliable internal clock for driving synchronous blocks during simulation.

Table 8. Performance summary of Ring Oscillator

Parameter	Value
Technology	45 nm CMOS
Stages (N)	23
Period (T)	0.936 ns
Frequency (f)	1.07 GHz

H. ARITHEMTIC LOGICAL UNIT (ALU BLOCK):

- ***Module Functionality:*** The Arithmetic Logic Unit (ALU) is the primary computational block of the CPU and performs all arithmetic, logical, and shift operations defined by the instruction set. The ALU receives two 8-bit operands (A<7:0> and B<7:0>) and a 3-bit operation code (op<2:0>) that determines which function is executed. Only one internal functional block is activated for each opcode, and its output is routed to the 8-bit ALU output bus (aluout<7:0>) through a final 8×1 multiplexer. The ALU supports eight operations: decrement, increment, multiplication, addition/subtraction, logical shift, XOR (odd-parity detection), NOT and AND.

- ***Architectural Specification:*** The ALU follows a modular and hierarchical architecture in which each operation is implemented as an independent functional block. These blocks include:

  1. Decrement (DECB)
  2. Increment (INCB)
  3. Multiplier
  4. Adder/Subtractor
  5. Logical Shifter
  6. XOR / Odd Parity Detector
  7. Inverter (NOT A)
  8. AND Block

  Each block produces an 8-bit output. All results feed into an 8-bit 8×1 multiplexer, which selects one output based on the 3-bit opcode. The opcode mapping used in the design is:

Table 9. ALU Opcode Table

Opcode (OP<2:0>)	Operation
000	Decrement (DECB)
001	Increment (INCB)
010	Multiplication
011	Addition / Subtraction
100	Shifter
101	XOR Operation / **Odd Parity Detection**
110	NOT Operation
111	AND Operation

The ALU symbol abstracts the internal complexity and exposes only operand inputs, opcode, and output, supporting clean hierarchical integration at the CPU top level.

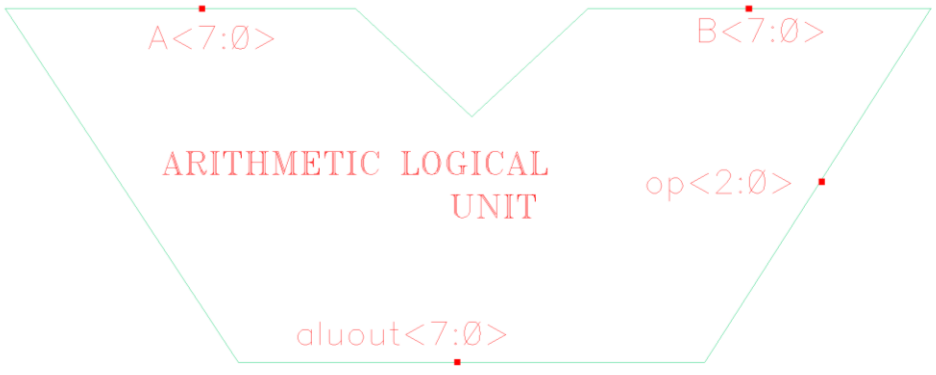


Figure 36. Symbolic representation of the Arithmetic Logical Unit (ALU) showing operand inputs A<7:0> and B<7:0>, opcode input op<2:0>, and output bus aluout<7:0>.

- ***Implementation Methodology:*** The ALU was implemented using a hierarchical transistor-level methodology in Cadence Virtuoso. Each internal function was first built and verified as a separate block using CMOS primitives from the FreePDK45 library. Fundamental gates (INV, NAND2, AND2, OR2, XOR2) were instantiated to construct larger arithmetic structures including the 1-bit full adder, half adder, 8-bit ripple-carry adder/subtractor, binary incrementer, and binary decrementer. The 4-bit array multiplier was developed using partial-product AND gates followed by a network of half adders and full adders. Logical functions such as XOR parity detection, bitwise AND, and NOT were implemented as dedicated byte-wide blocks. The shifter block was realized using a cascaded multiplexer network, which later formed the critical path of the ALU. All sub-block outputs were routed into a hierarchical 8×1 multiplexer that selects the final ALU result based on the opcode. After individual verification, the modules were integrated to form the complete ALU and validated through full transient simulation.

➢ ***1-BIT HALF ADDER:***

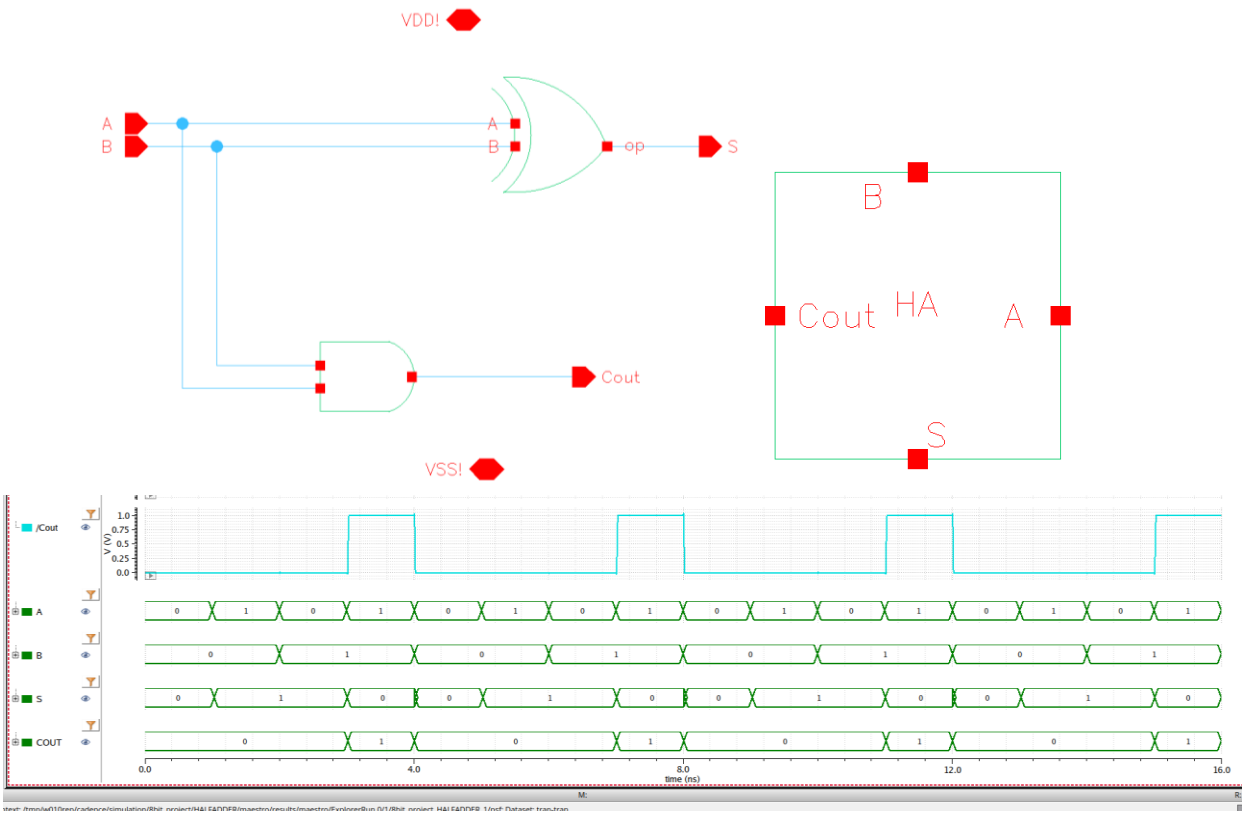
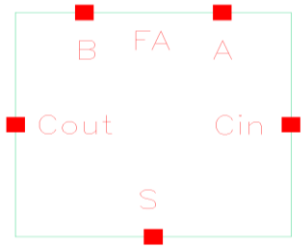


Figure 37. Half Adder - Schematic, Symbol, and Simulation

The Half Adder is implemented using a CMOS XOR gate for the SUM output and an AND gate for the CARRY output in the FreePDK45 technology. The symbolic view simplifies the block to inputs A, B and outputs SUM, CARRY for easy hierarchical integration. The transient waveform verifies correct functionality, showing SUM = A ⊕ B and CARRY = A ∧ B for all input combinations.

➢ ***1-BIT FULL ADDER:***



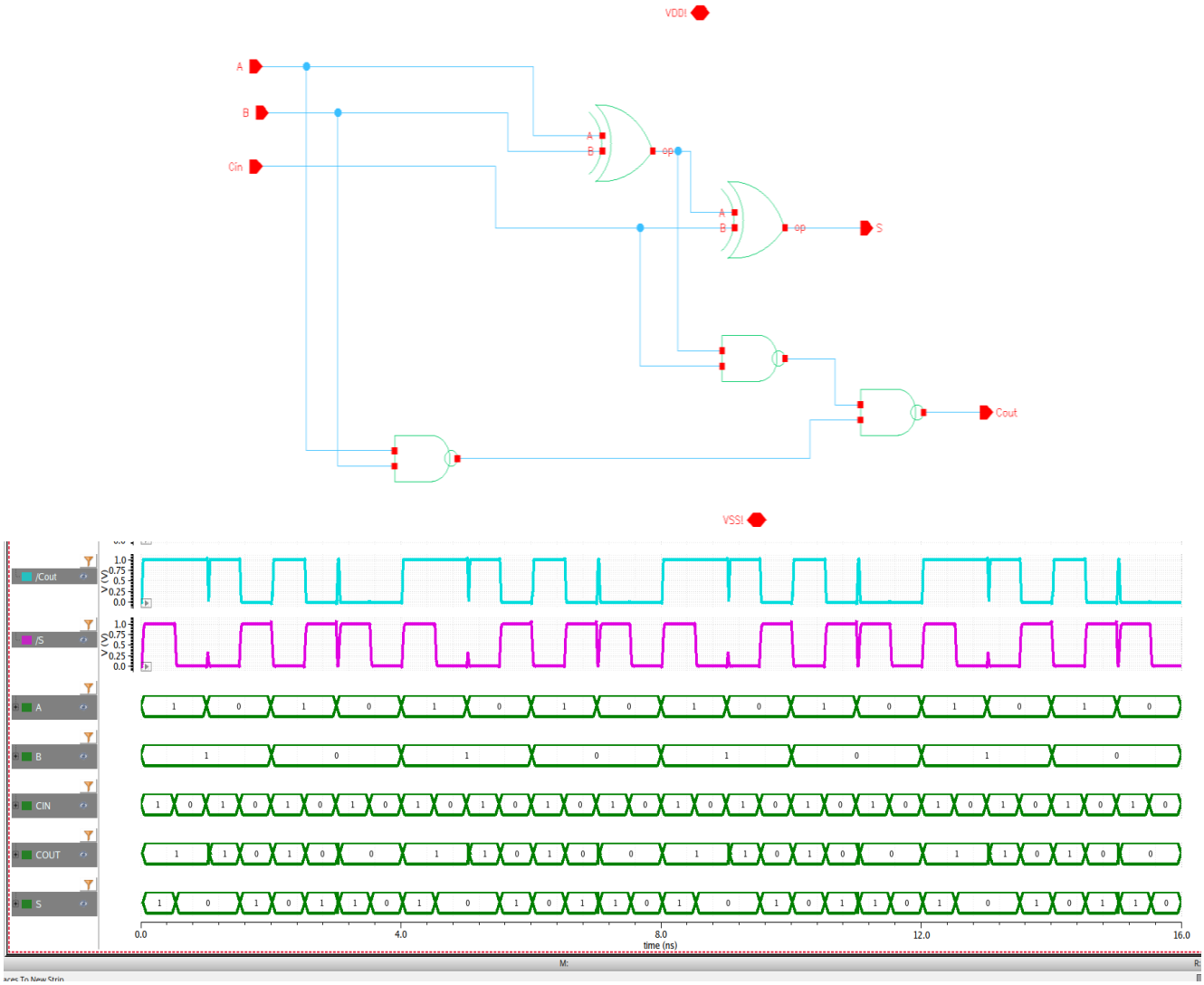
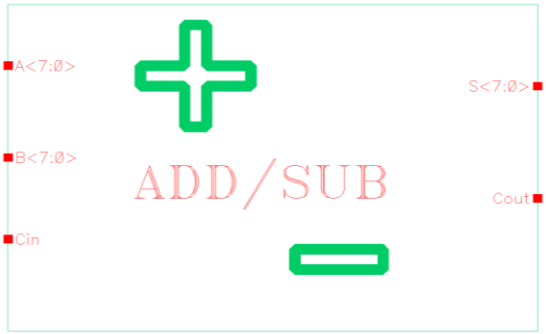


Figure 38. Full Adder – Schematic, Symbol, and Simulation

The Full Adder is implemented using a transistor-level CMOS XOR, AND, and OR gate combination to generate the SUM and CARRY outputs in FreePDK45 technology. The symbolic view abstracts the design into inputs A, B, Cin and outputs SUM, Cout for hierarchical integration. The transient simulation confirms correct arithmetic behavior, where SUM = A ⊕ B ⊕ Cin and Cout reflects proper carry-generation for all input cases.

➢ ***8-BIT ADDER/SUBTRACTOR WITHOUT REGISTER:***



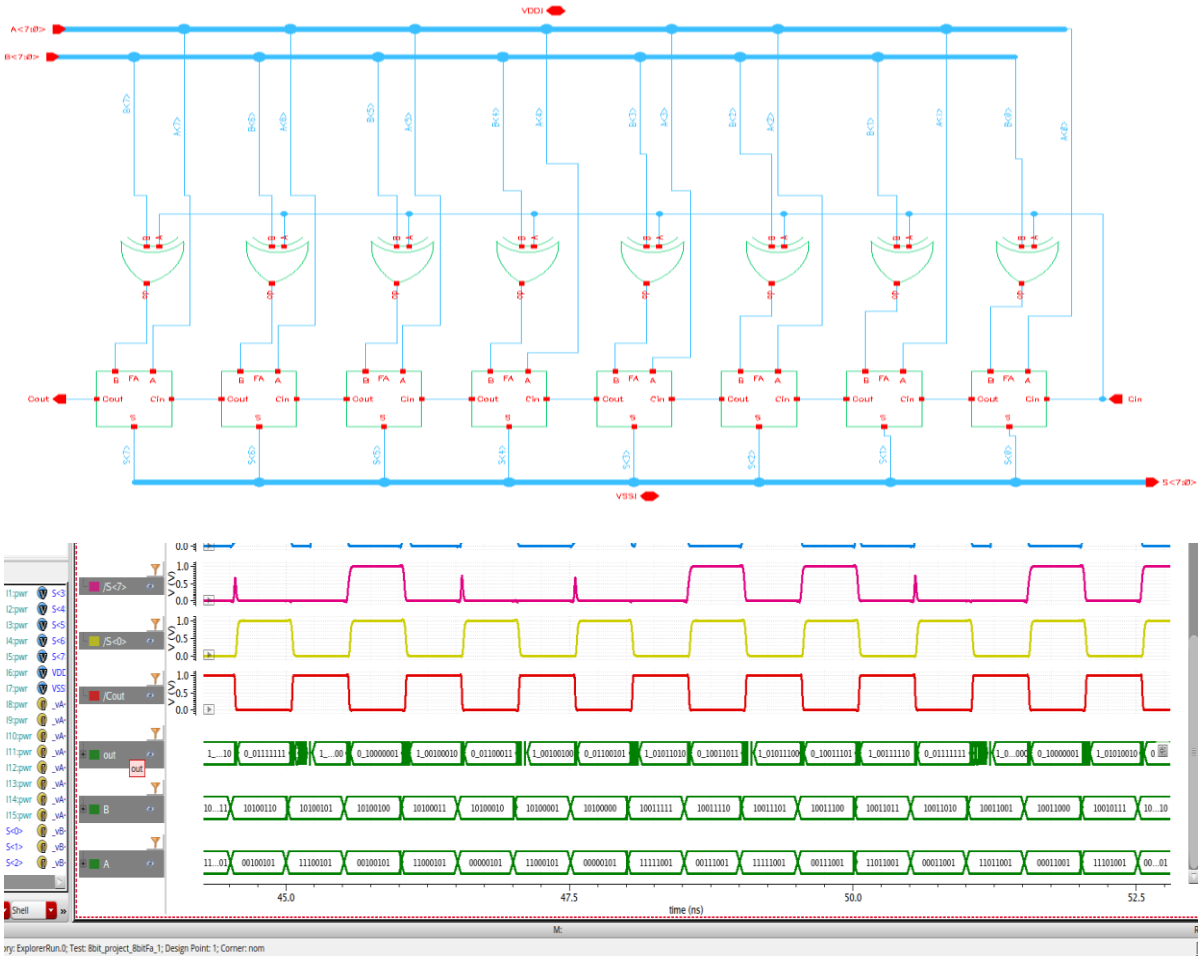
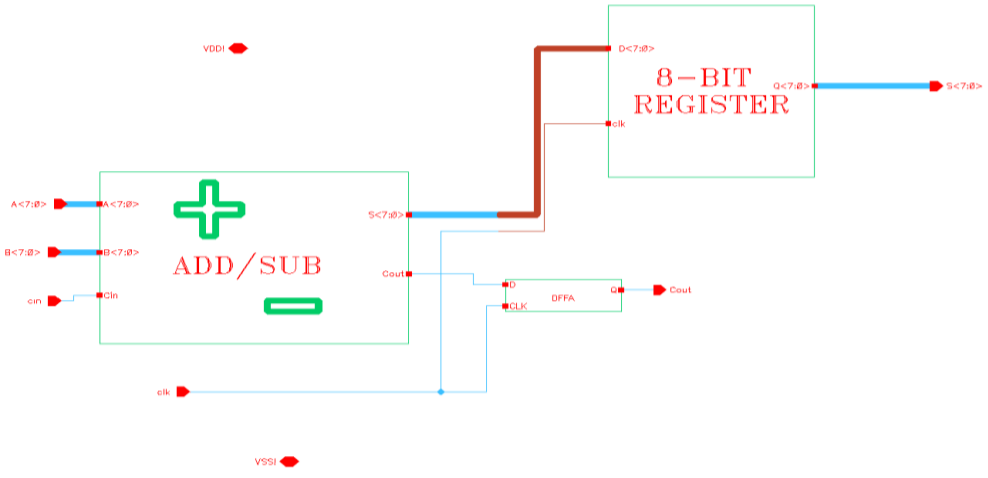


Figure 39. 8-bit Adder/subtractor (without register) - Schematic, symbol and Simulation

The unregistered 8-bit ripple-carry adder is formed by cascading eight transistor-level full adders, with each FA passing its carry to the next stage. The circuit performs addition when Cin = 0 and two's-complement subtraction when Cin = 1, where B is internally XORed with Cin. Because the design is fully combinational, the SUM<7:0> outputs exhibit small glitches during carry propagation. These transients are expected in ripple structures and settle to the correct final value once all carries resolve.

- ***8-BIT ADDER/SUBTRACTOR WITH REGISTER:***



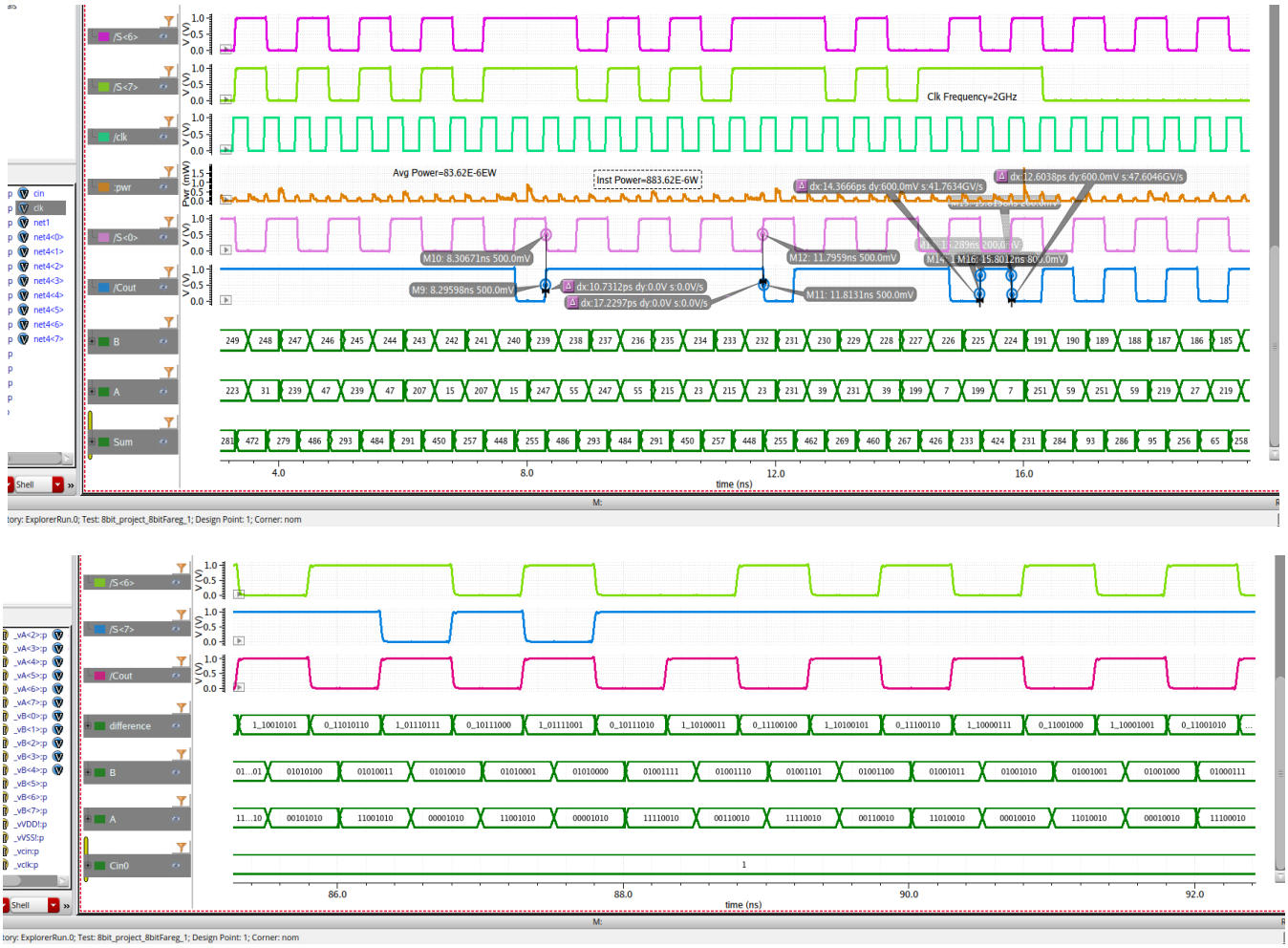
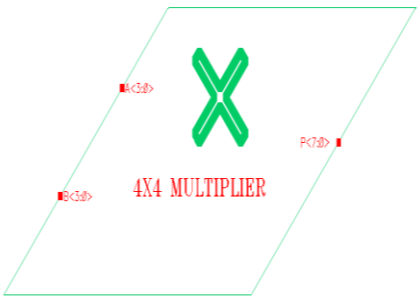


Figure 40. 8-bit Adder/subtractor (with register) - Schematic, symbol and Simulation

To eliminate the combinational glitches produced by the ripple-carry propagation, a register stage was added only at the output of the 8-bit adder/subtractor. In this architecture, the adder computes the SUM<7:0> using the combinational logic as before, and the resulting output is captured by an 8-bit D-flip-flop on the rising clock edge. This ensures that the value delivered to the CPU datapath is fully synchronized and glitch-free. The transient waveform verifies clean, stable timing and correct arithmetic results for both addition (Cin = 0) and subtraction (Cin = 1).

- ***4-BIT MULTIPLEXER:***



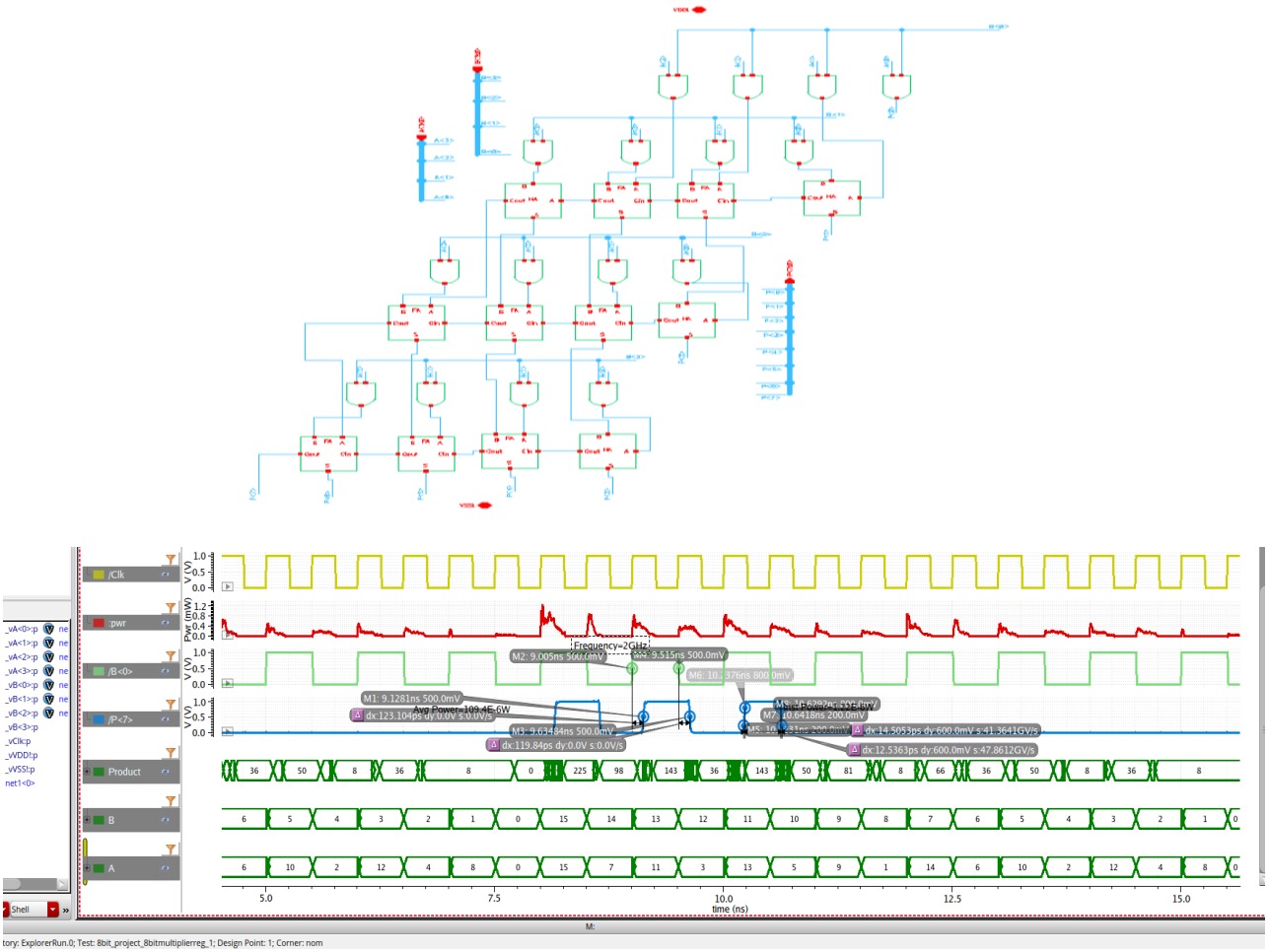
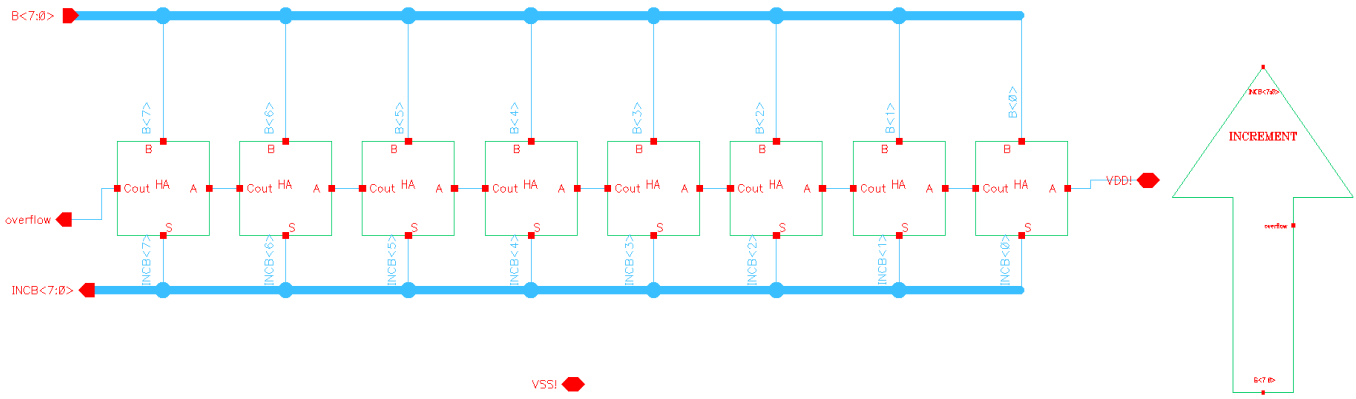


Figure 41. Transistor-level Implementation and Simulation of the 4-bit Multiplier

The 4-bit multiplier is implemented using an array-multiplier architecture, where partial products are generated using AND gates and summed column-wise through a structured network of half adders (HA) and full adders (FA). The schematic shows the systematic arrangement of HA/FA cells that propagate carries diagonally and produce the 8-bit product output. The transient waveform confirms correct multiplication results for several input combinations, such as 6 × 6 = 36 and 15 × 15 = 225, demonstrating accurate partial-product accumulation. The clocked waveform also shows clean transitions and stable product bits, verifying proper arithmetic functionality and timing behavior of the multiplier.

- ***8-BIT INCREMENT WITHOUT REGISTER:***



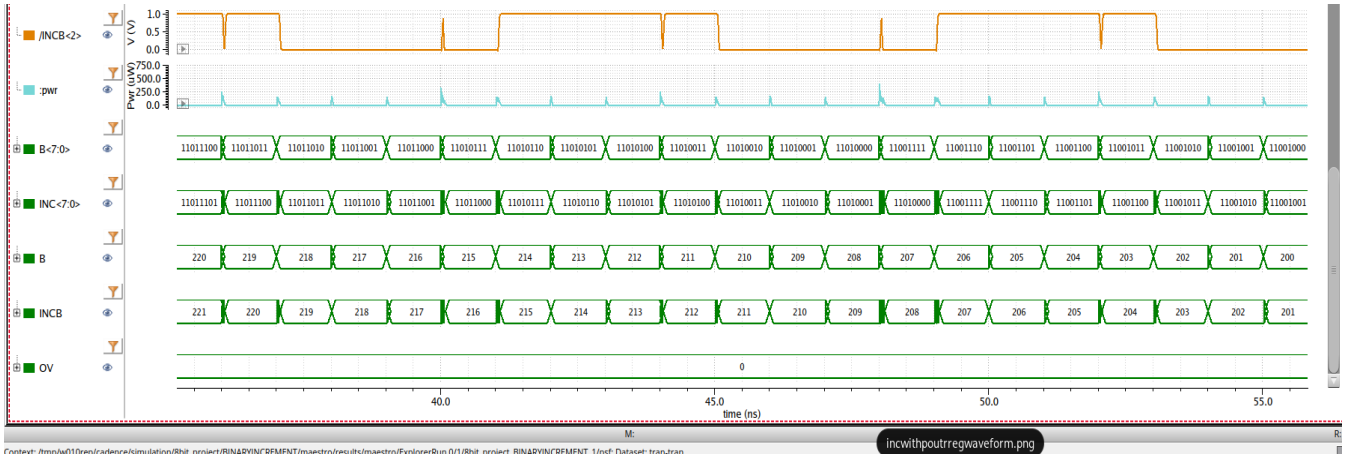
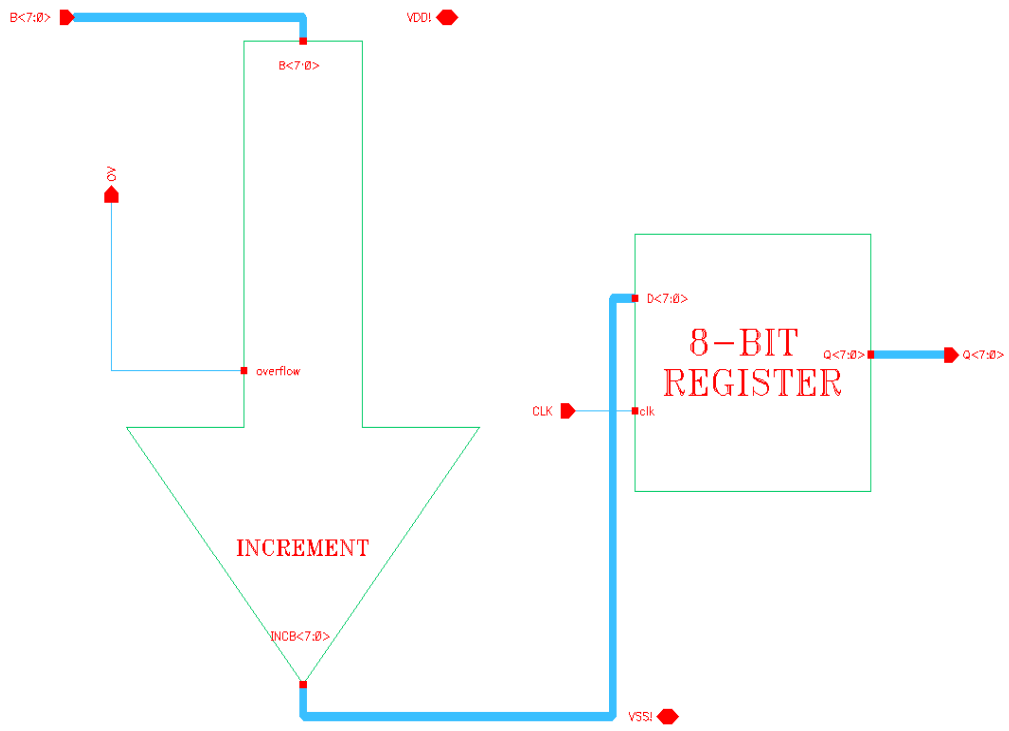


Figure 42. Transistor-Level Schematic, Symbol, and Transient Waveform of the 8-bit Incrementer (Without Register)

The 8-bit incrementer is implemented using a chain of eight cascaded Half Adders, where each bit adds a constant '1' through ripple carry propagation. The schematic shows the series HA structure that generates the INCB<7:0> output and an overflow flag. The symbol abstracts this into an 8-bit combinational block. The transient waveform confirms correct +1 behavior across all test vectors - for example, B = 220 → 221, 219 → 220, 210 → 211 - while also showing small combinational glitches during carry transitions due to the absence of output registers.

- ***8-BIT INCREMENT WITH REGISTER:***



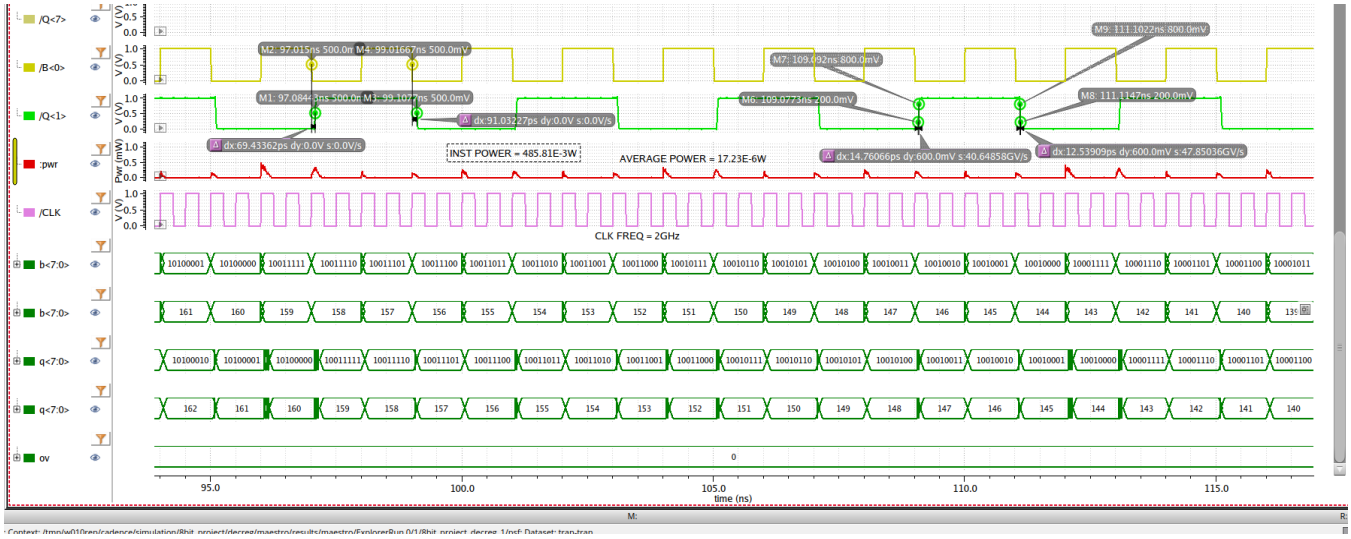


Figure 43. Schematic and Transient Simulation of the 8-bit Incrementer with Output Register Demonstrating Glitch-Free Operation.

The final incrementer design integrates an 8-bit synchronous register at the output to eliminate ripple-carry glitches present in the purely combinational version. As shown in the schematic, the incremented value INCB<7:0> is captured on the rising clock edge, ensuring stable and time-aligned outputs for the CPU datapath. The simulation waveform demonstrates clean sequential increments (e.g., 162 → 163 → 164 …) without any intermediate switching spikes, confirming that the register effectively suppresses all combinational glitches. The block operates reliably at high speed, with instantaneous switching peaks during transitions and an average power of approximately 17.23 µW at a 2 GHz clock frequency.

- ***8-BIT DECREMENT WITHOUT REGISTER:***

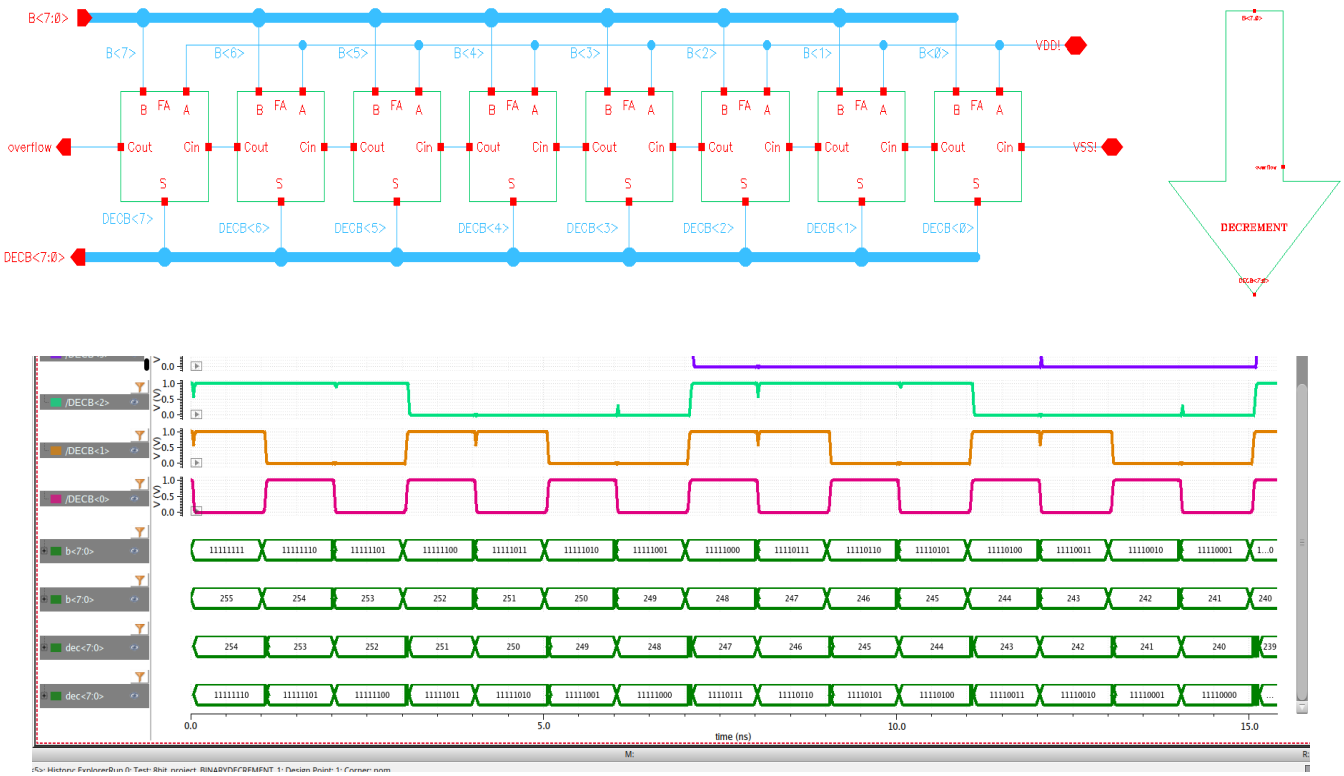


Figure 44. Schematic, Symbol and Transient Simulation of the 8-bit Decrementer Without Output Register.

The figure shows the operation of the 8-bit ripple-borrow decrementer, where each stage subtracts one by propagating the borrow through a chain of full adders. Since no output register is used, small combinational glitches appear during transitions as the borrow ripples across bits, but the output stabilizes correctly each cycle (e.g., 255 → 254 → 253 → 252 …). The waveform clearly demonstrates the expected decrement sequence along with brief intermediate switching activity typical of an unregistered ripple structure.

- ***8-BIT DECREMENT WITH REGISTER:***

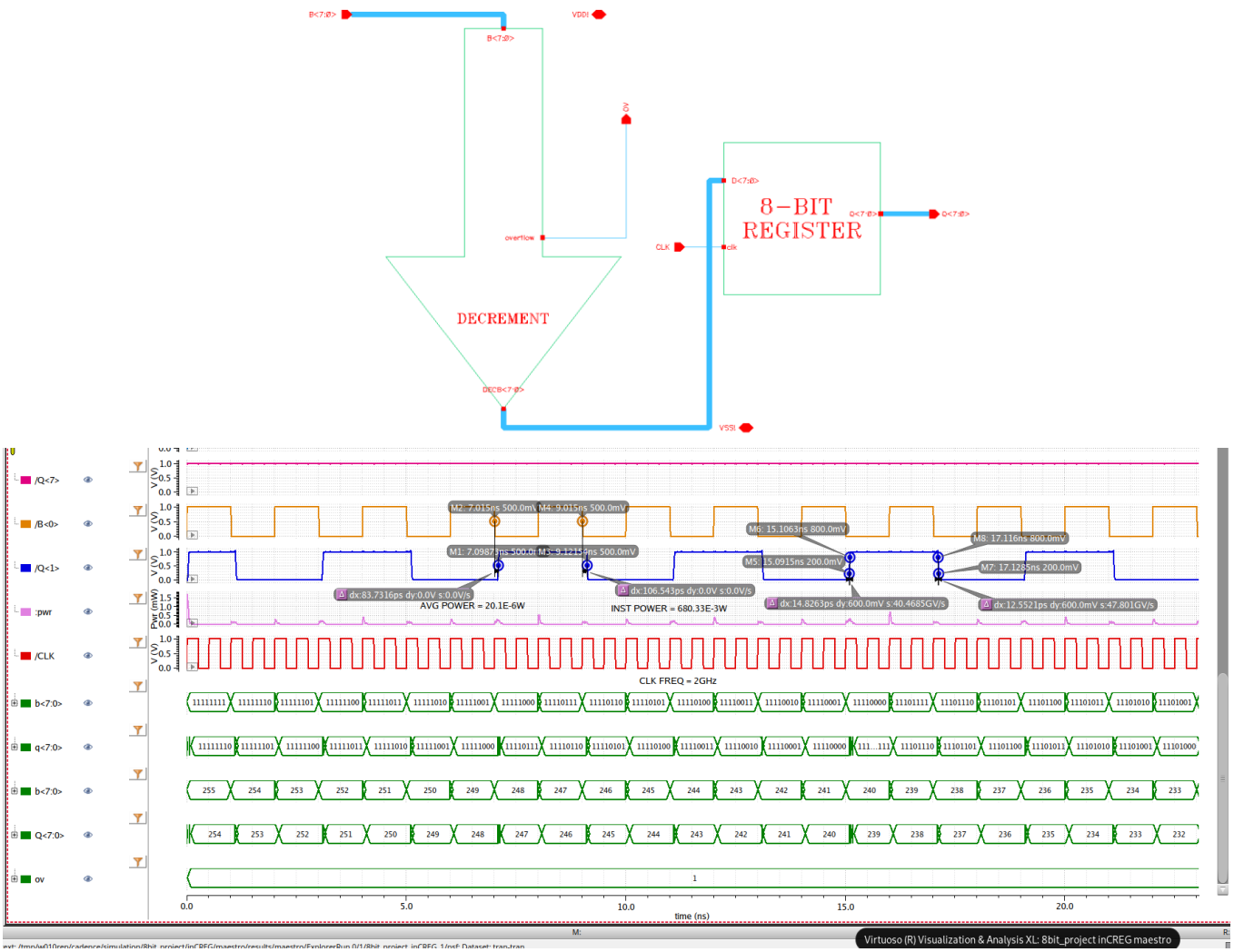


Figure 45. Schematic and Transient Simulation of the 8-bit Decrementer With Output Register.

The figure shows the complete 8-bit decrementer integrated with an output register, where the combinational decrement logic generates the next value and the 8-bit D-flip-flop stage captures the final output on each rising clock edge. The transient simulation confirms clean, glitch-free transitions as the register removes the intermediate ripple-borrow switching activity seen in the unregistered version. The output sequence (254 → 253 → …) appears fully synchronized with the clock, demonstrating stable behavior suitable for CPU datapath integration and high-frequency operation.

➢ ***8-BIT BARREL SHIFTER:***

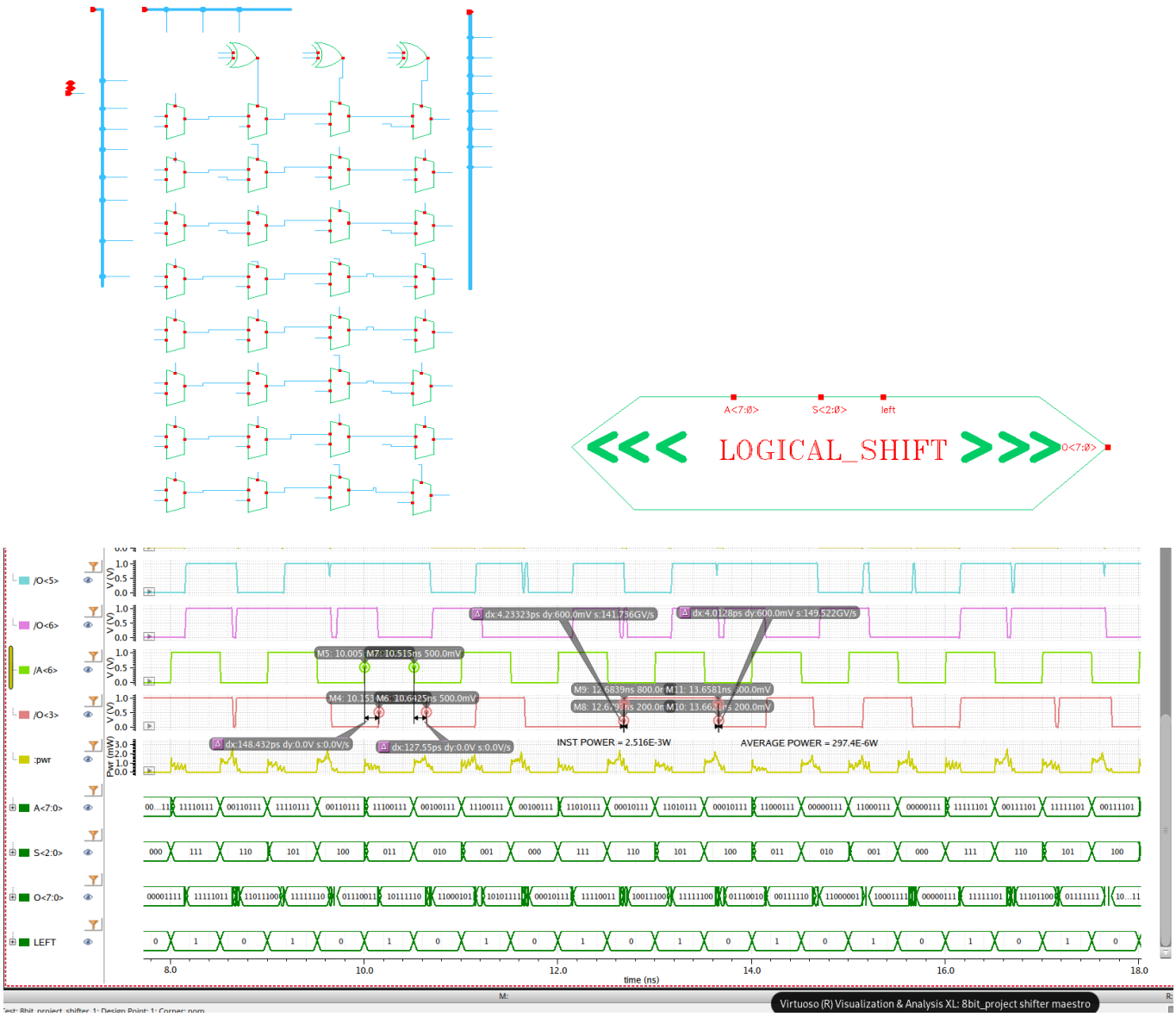


Figure 46. Schematic and Transient Simulation of the 8-bit Barrel Shifter.

The 8-bit barrel shifter is implemented using three hierarchical stages of cascaded 2:1 multiplexers, enabling shift amounts of 1 bit, 2 bits, and 4 bits, which combine to support any logical shift from 0 to 7 positions. The select inputs S2, S1, S0 control the effective shift amount, while the input word A<7:0> is routed through the multiplexer network to produce the shifted output Q<7:0>. The transient simulation waveform confirms correct shifting behavior for all select-line combinations. For S2S1S0 = 000, the output directly matches the input, and as the select lines toggle (e.g., 001, 010, 011, … 111), the output bits shift in a parallel and fully synchronized manner. The observed results show clean transitions, stable output values, and correct logical shifting without glitches, validating the functional correctness of the barrel shifter architecture.

➢ ***8-BIT LOGICAL AND:***

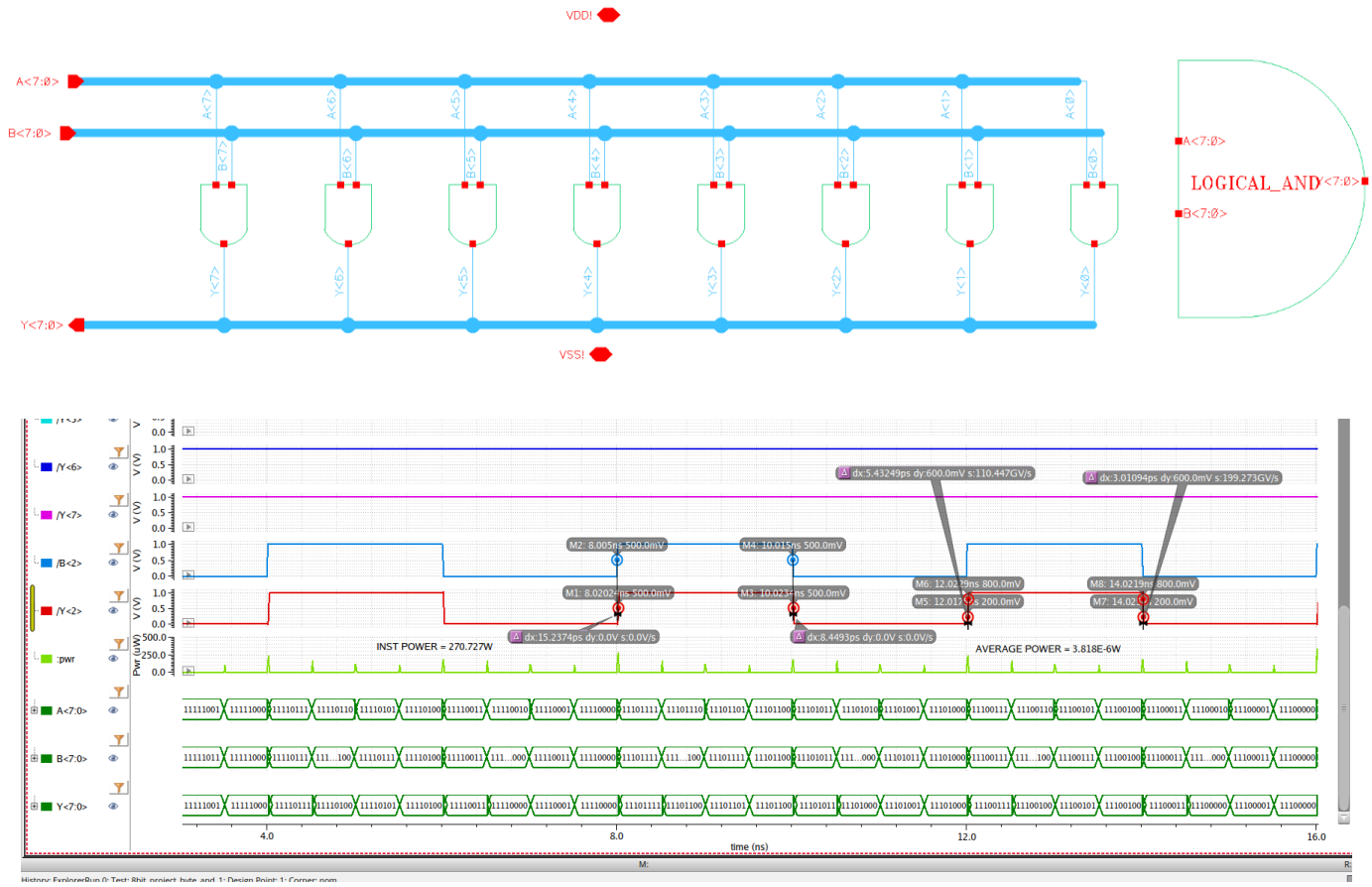
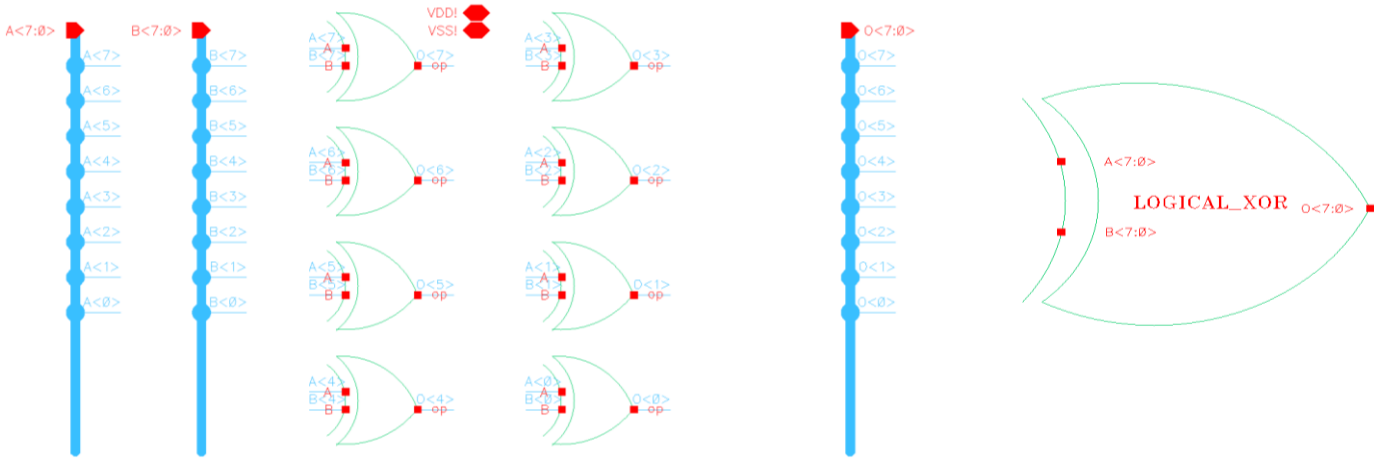


Figure 47. Symbol, Schematic, and Transient Simulation of the 8-bit Logical AND Block.

The 8-bit Logical AND block performs a bitwise AND operation between the two 8-bit operands A<7:0> and B<7:0>. The circuit is implemented using eight parallel 2-input CMOS AND gates, each generating one output bit of Y<7:0>. The transistor-level design ensures clean switching with no hazards, as each bit is computed independently. The transient simulation confirms correct functionality for all input combinations: the output bit is '1' only when both corresponding input bits are '1'. This block is used as the logical operation for opcode 111 in the ALU.

➢ ***8-BIT LOGICAL XOR:***



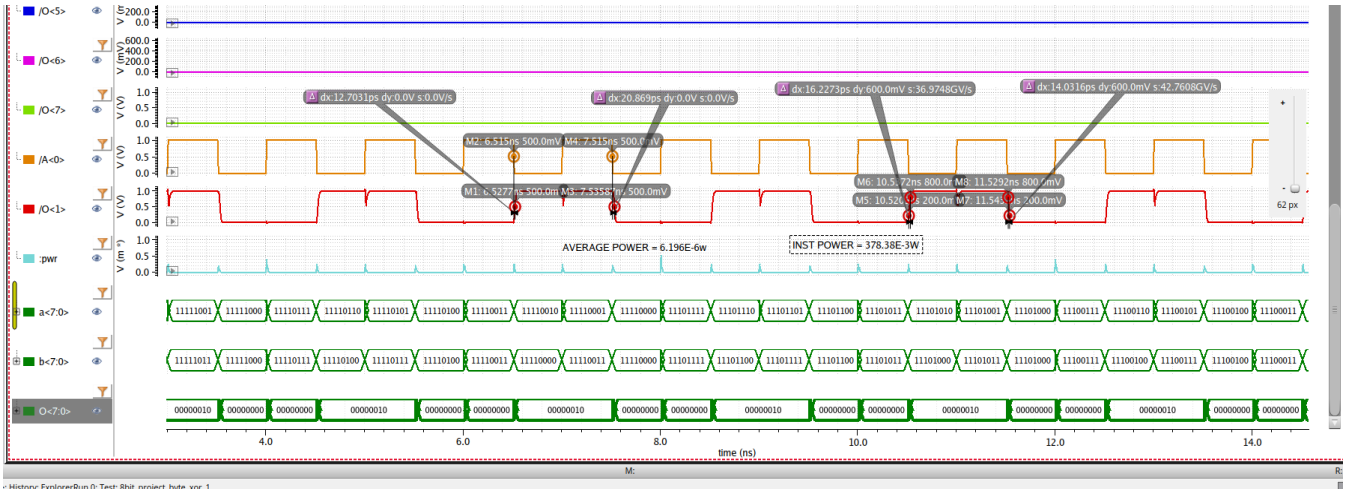
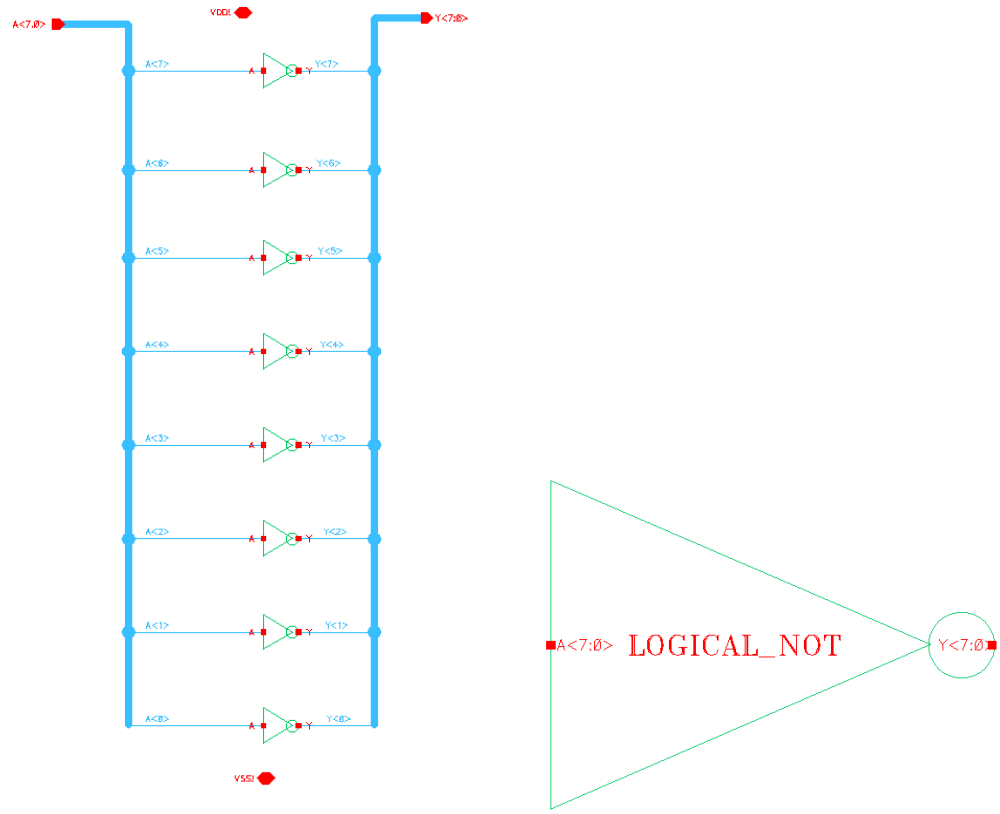


Figure 48. Symbol, Schematic, and Transient Simulation of the 8-bit Logical XOR Block.

The 8-bit Logical XOR block performs a bitwise exclusive-OR operation between operands A<7:0> and B<7:0>, generating output D<7:0>. The circuit is constructed using eight parallel CMOS XOR gates, each producing one output bit independently. Since XOR outputs '1' only when the two input bits differ, this block is also useful for parity generation in the ALU. The transistor-level implementation ensures clean switching and symmetric rise/fall behavior across all bits. Transient simulation confirms correct functionality: output transitions occur exactly when the corresponding input bits toggle, with no hazards or glitches, demonstrating stable and accurate odd-parity behavior for opcode 101.

- ***8-BIT LOGICAL INVERTER:***



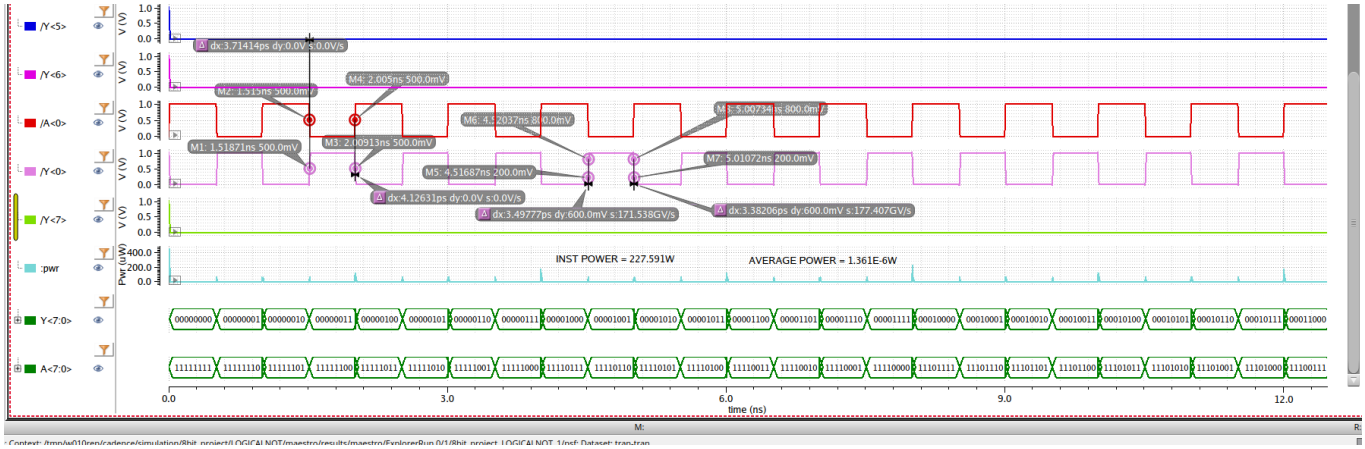
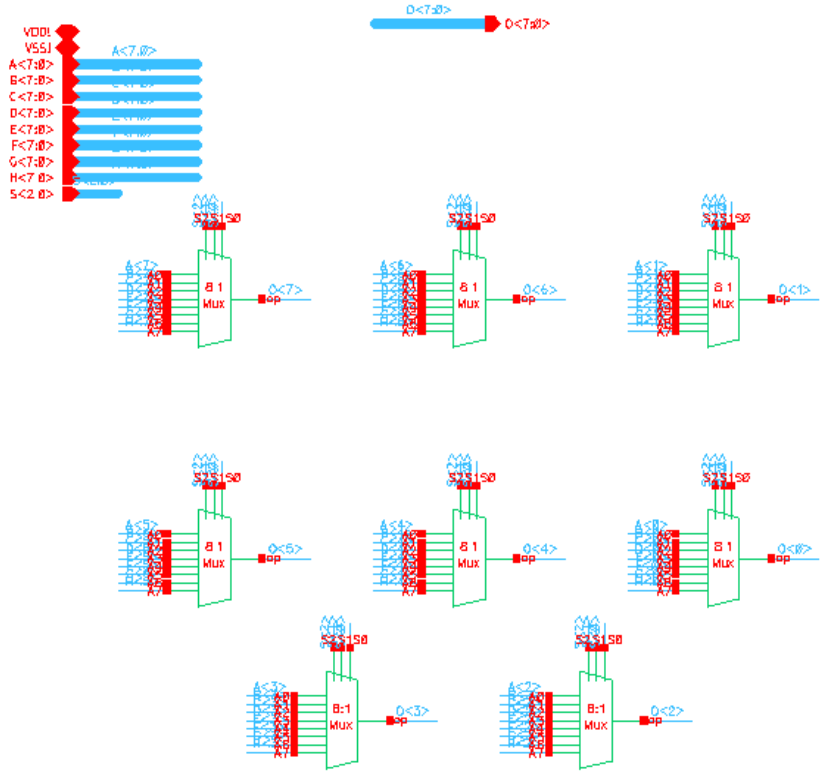


Figure 49. Symbol, Schematic, and Transient Simulation of the 8-bit Logical NOT Block.

The 8-bit Logical NOT unit performs bitwise inversion of the input vector A<7:0> using eight parallel CMOS inverter stages. Each bit of the 8-bit input is independently inverted to produce the corresponding output bit, ensuring fully combinational and delay-matched behavior. The schematic illustrates the uniform inverter structure repeated across all eight bits, while the symbol provides a simplified representation for hierarchical integration. Transient simulation results confirm correct inversion across all input patterns - for example, 11111111 → 00000000 and 11001011 → 00110100 - showing clean transitions, no glitches, and stable output behavior during switching.

- ***8 x 1 MULTIPLEXER:***



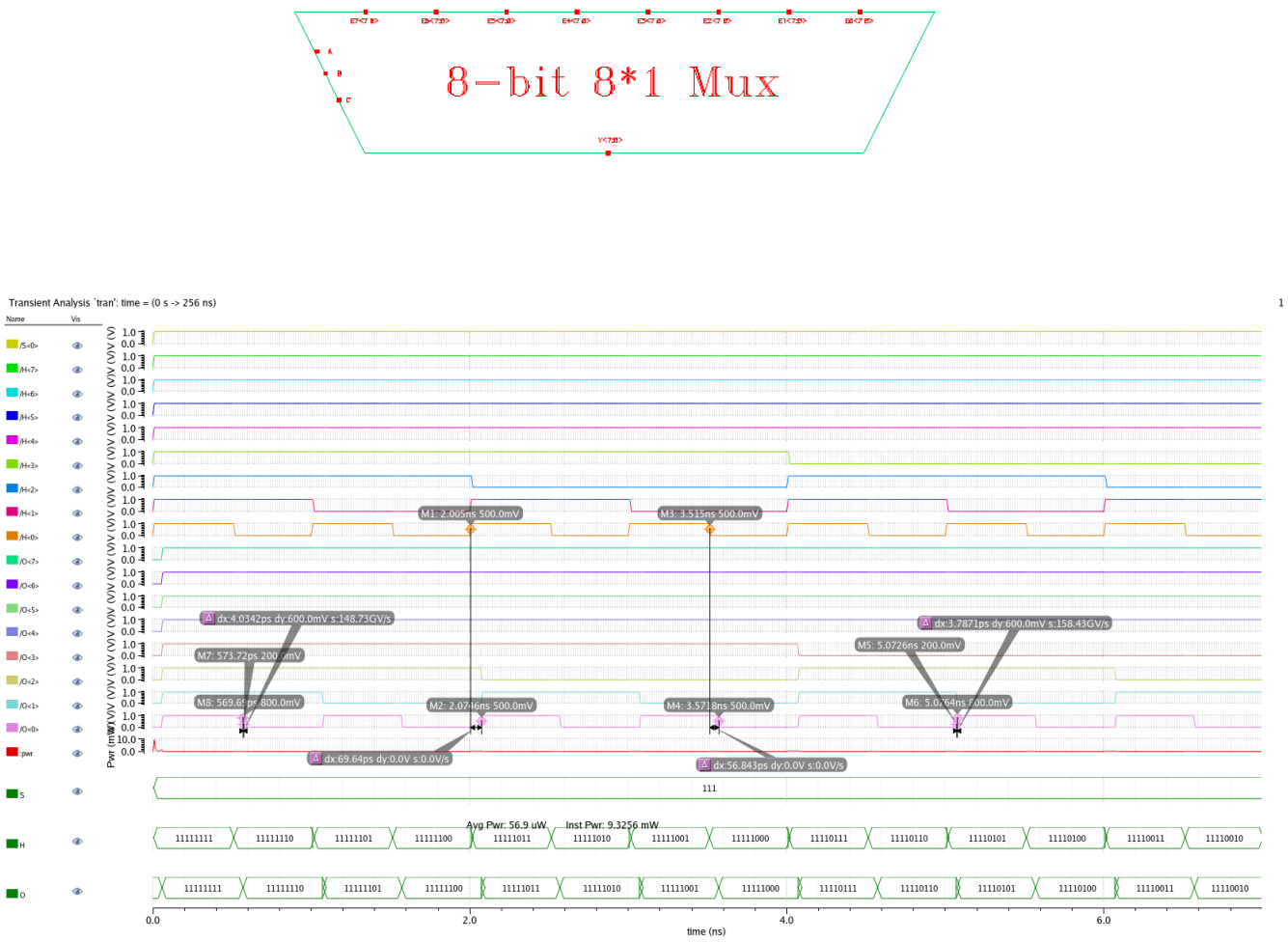


Figure 50. Symbol, Schematic, and Transient Simulation of the *8 x 1 MUX*

The 8-bit 8×1 multiplexer in the ALU serves as the final selection stage that chooses one of the eight operation results generated by the internal functional units (AND, OR, XOR, Adder, Subtractor, Incrementer, Decrementer, and Pass-through). Each functional unit produces an 8-bit output, and the multiplexer routes the correct result to the ALU output based on the 3-bit select signal S<2:0>, which is driven by the instruction decoder. The design is implemented using eight parallel 8→1 MUX slices, where each slice selects the corresponding bit from all eight function outputs. This bit-sliced structure ensures uniform delay across all data bits and simplifies transistor-level construction. The multiplexer operates as the critical datapath element that ensures only the desired arithmetic or logical result propagates forward, enabling clean control of ALU behavior during CPU execution. Simulation waveforms confirm that for every change in select lines, the output updates immediately to the corresponding function block result, with all bits switching consistently.

➢ **8-BIT ALU TOP-LEVEL SCHEMATIC:**

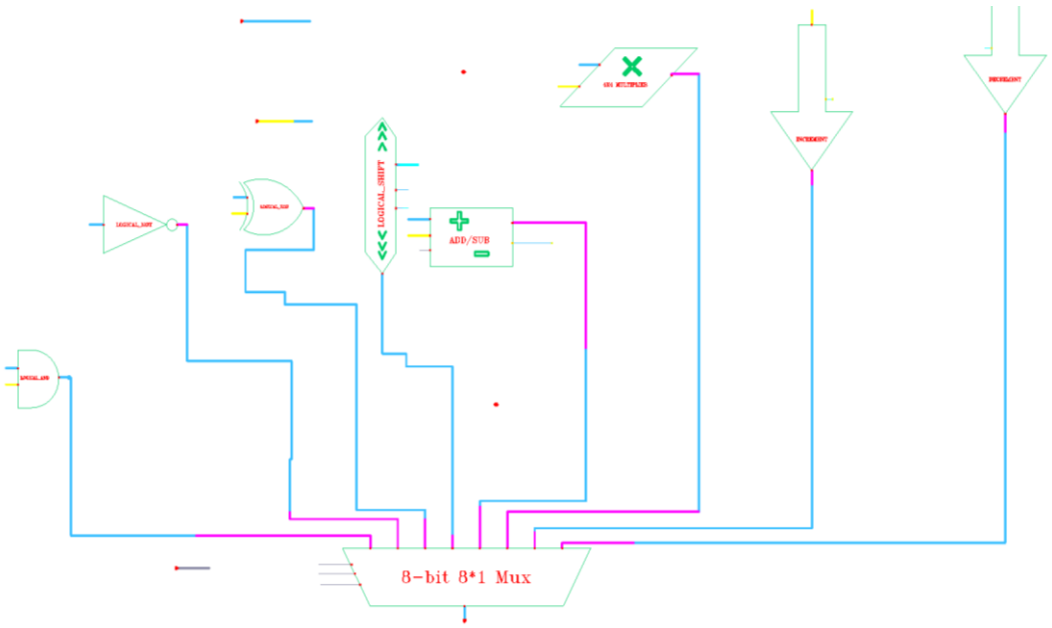


Figure 51. Top-Level ALU Schematic Showing Integration of All Functional Blocks

The figure illustrates the complete 8-bit ALU architecture, where each functional unit- incrementer, decrementer, adder/subtractor, multiplier, logical shifter, AND block, XOR block, and NOT block is implemented as an independent module. All block outputs are routed into a hierarchical 8×1 multiplexer, which selects the final ALUOUT<7:0> based on the 3-bit opcode. The wiring highlights the modular structure and clean signal flow between arithmetic, logical, and shift components. This integrated schematic forms the core computational datapath used in the CPU design.

- ***Functional Behavior:*** During each clock cycle, the ALU receives the operand inputs A and B along with the 3-bit opcode op<2:0>, which determines the specific operation to be executed. Only the functional block associated with the selected opcode becomes active, and its result is routed to the aluout<7:0> bus through the final 8×1 multiplexer. When LD_AC = 1, the accumulator captures the ALU output on the rising edge of the clock. The ALU operates as a combinational unit, and its output transitions follow the internal propagation delay of the active block. Among all operations, the logical shifter exhibits the longest delay, making it the critical path of the ALU due to its cascaded multiplexer structure and multi-level signal routing.

  Examples of functional behavior:

  000 - DECB: Produces B − 1 (e.g., 11111111 → 11111110)
  001 - INCB: Produces B + 1 (e.g., 00000001 → 00000010)
  010 - MUL: Outputs lower 4 bits of A × B
  011 - ADD/SUB: Performs A + B or A − B based on Cin
  100 - SHIFT: Logical shift right by 1 bit (A >> 1)
  101 - XOR: Outputs A ⊕ B; also indicates odd parity
  110 - NOT: Outputs the bitwise inversion of A
  111 - AND: Produces A ∧ B

- ***Verification and Simulation Analysis:*** The ALU was verified using transient simulations in Cadence Spectre during which all eight operations were exercised with multiple operand patterns. The simulation waveforms confirmed that each opcode correctly activated its corresponding functional block and that the ALUOUT signal transitioned cleanly. To eliminate internal glitches caused by combinational switching, 8-bit input registers were added before the A and B operand buses and an 8-bit output register was added after ALUOUT. These registers ensured that both operands and results remained fully synchronized with the system clock, producing stable, edge-aligned waveforms. Arithmetic operations showed consistent propagation behavior, while logical and shift operations responded immediately as expected. Overall, the simulated outputs matched the designed truth table, opcode mapping, and expected functional behavior for all test cases.

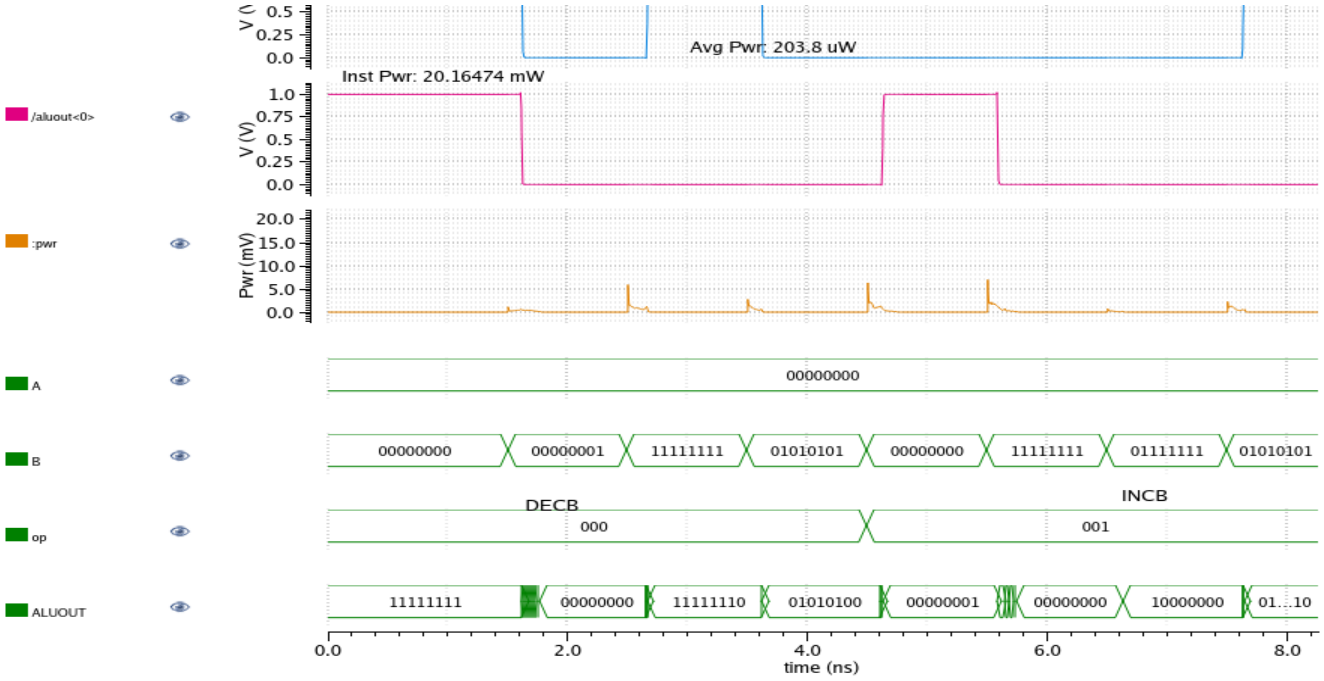


Figure 52. Transient Simulation of ALU Arithmetic Decrement (000) and Increment (001) Operations.

The waveform shows that the DECB operation correctly produces outputs such as 11111111 → 11111110, while the INCB operation generates 00000000 → 00000001, confirming proper subtract-by-one and add-by-one behavior.

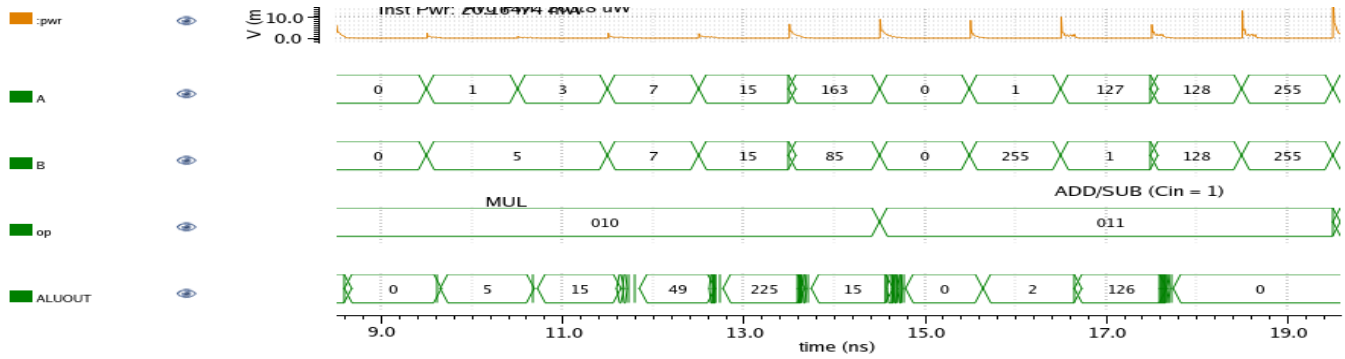


Figure 53. Transient simulation of the ALU performing Arithmetic multiplication (010) and addition/subtraction (011) operations.

The waveform shows that the MUL block correctly computes products such as 1 × 5 = 5, 3 × 5 = 15, and 7 × 7 = 49, while the ADD/SUB operation produces expected arithmetic results 128 - 128 = 0, 127 − 1 = 126, confirming correct ALU functionality.

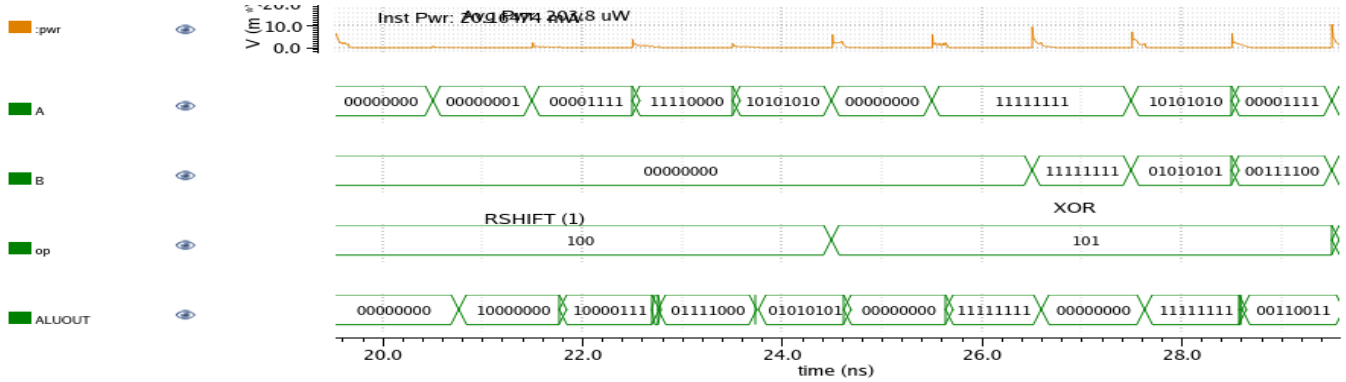


Figure 54. Transient simulation of the ALU performing Logical shift (100) and XOR (101) operations.

The waveform shows that the SHIFT block correctly shifts right values such as 00001111 → 10000111, while the XOR operation produces outputs like 11111111 ⊕ 11111111 = 00000000 and 10101010 ⊕ 01010101 = 11111111, confirming correct shift and odd-parity behavior.

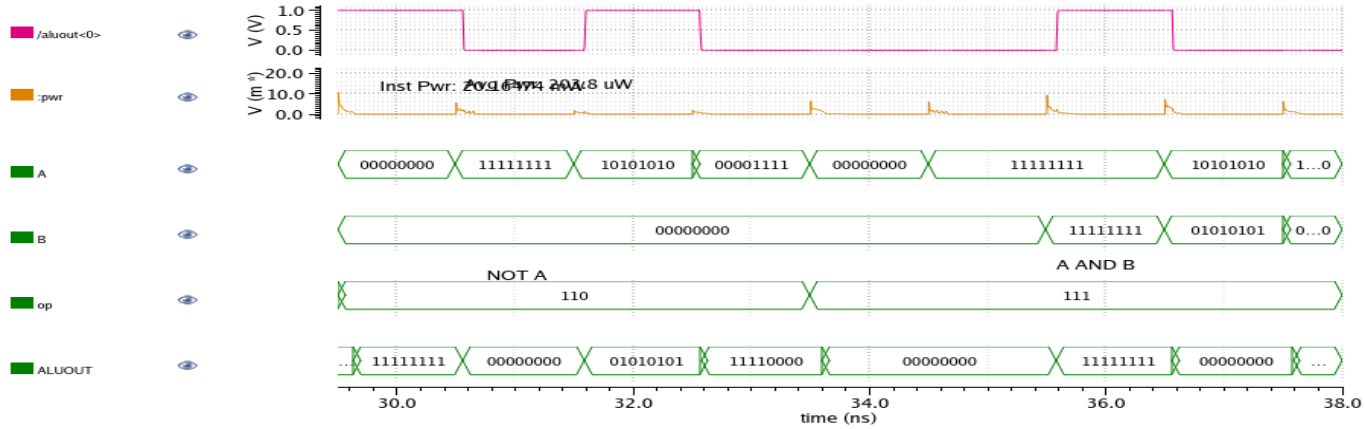


Figure 55. Transient simulation of the ALU performing Logical NOT (110) and AND (111) operations.

The waveform shows that the NOT block correctly inverts operand A 11111111 → 00000000, 00001111 → 11110000, while the AND operation produces expected bitwise results such as 11111111 ∧ 11111111 = 11111111, 10101010 ∧ 01010101 = 0000000 demonstrating correct logical functionality.

- ***Performance Analysis:*** Spectre-based power and delay simulations show that the ALU consumes an average power of 203.8 µW, with instantaneous switching peaks reaching 20.16 mW during opcode transitions. Most operations exhibit short propagation delays; however, the logical shifter forms the critical path because of its cascaded multiplexer structure, giving it the highest delay in the ALU. This path ultimately limits the maximum achievable frequency of operation. The full ALU implementation consists of 2,748 transistors, representing the combined complexity of its arithmetic, logical, and multiplexing circuitry.

Table 10. Performance summary of ALU Block

Parameter	Value
Average Power (Pavg)	203.8 µW
Instantaneous Peak Power (Pinst)	20.16474 mW
Total Transistor Count	2,748 transistors

I. INSTRUCTION MEMORY:

- ***Module Functionality:*** The Instruction Memory block stores the 8-bit instruction coming from the program data input and separates it into a 3-bit OPCODE and a 5-bit ADDRESS. When the Load (LD) and Reset (RST) signals are high, the block captures the 8-bit DATA input on the rising edge of the clock. The upper three bits become OPCODE<2:0>, which go to the control unit, and the lower five bits become ADDRESS<4:0>, which are sent to the address-selection multiplexer. This block acts as the bridge between the instruction input and the internal CPU control signals.

- ***Architectural Specification:***

  The Instruction Memory is built using eight D-flip-flops, one for each DATA bit.
  - DATA<7:5> → OPCODE<2:0>
  - DATA<4:0> → ADDRESS<4:0>

  The control signals are:
  - CLK – clock input for synchronous updates
  - LD – loads the instruction during the active clock edge
  - RST – resets the stored instruction to a known state

  The symbolic view combines all bit registers into a single block labeled Instruction Memory, with clear ports for DATA, OPCODE, ADDRESS, CLK, LD, and RST.

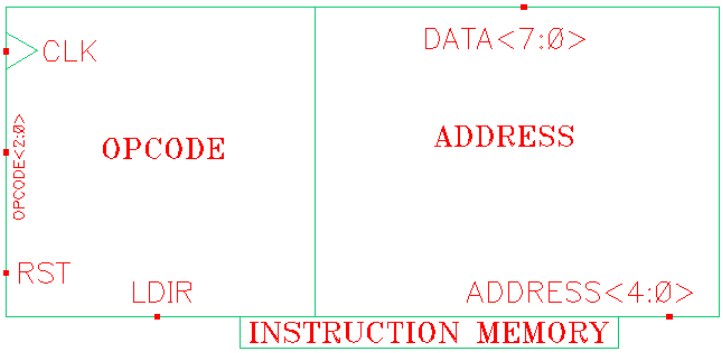


Figure 56. Symbol representation of instruction memory

- ***Implementation Methodology:*** The block was implemented in Cadence Virtuoso using FreePDK45 transistor-level DFF cells. Each DATA bit is directly connected to a D input of its flip-flop. The outputs of the flip-flops are grouped to form the OPCODE and ADDRESS buses. All flip-flops share the same CLK, LD, and RST signals so that the entire 8-bit instruction word updates together. The schematic was abstracted into a clean symbol for easy hierarchical integration in the CPU.

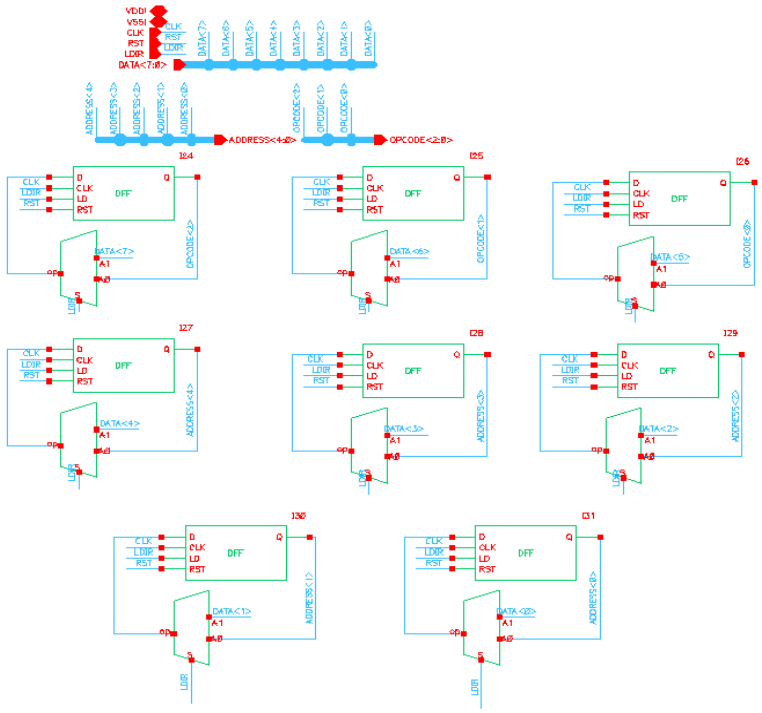


Figure 57. Transistor-Level Schematic of the Instruction Memory Block

- ***Functional Behavior:***

The Instruction Memory behaves like a simple synchronous instruction register:

- LD = 1 and RST = 1:
  On the rising clock edge, the current 8-bit DATA value is loaded and immediately separated into OPCODE and ADDRESS.
  Example: DATA = 11111101 → OPCODE = 111, ADDRESS = 11101.
- LD = 1 and RST = 0:
  The DATA word is also loaded on the clock edge, updating the OPCODE and ADDRESS outputs.
- LD = 0:
  The previously stored instruction remains unchanged, even if DATA continues switching.

The outputs only change on the clock edge, so the CPU receives stable OPCODE and ADDRESS signals during the cycle. This ensures correct execution-stage control sequencing.

- ***Verification and Simulation Analysis:*** The Instruction Memory was verified using transient simulation by applying sequential 8-bit DATA patterns while controlling the LD and RST signals. When LD = 1 and RST = 1, the memory correctly captured the incoming instruction and separated it into the 3-bit OPCODE and 5-bit ADDRESS on the next rising clock edge. The waveform shows the DATA inputs transitioning through values such as 11111111, 11111110, 11111101, 11111100, and the corresponding outputs updating correctly-for example, OPCODE = 111 and ADDRESS = 11111, followed by OPCODE = 111 and ADDRESS = 11101, and then OPCODE = 111 and ADDRESS = 11100. All updates occur only at clock boundaries, and no unwanted intermediate switching is observed. The simulation confirms that the Instruction Memory reliably stores each 8-bit instruction and outputs the correct opcode and address fields exactly as intended for the CPU fetch stage.

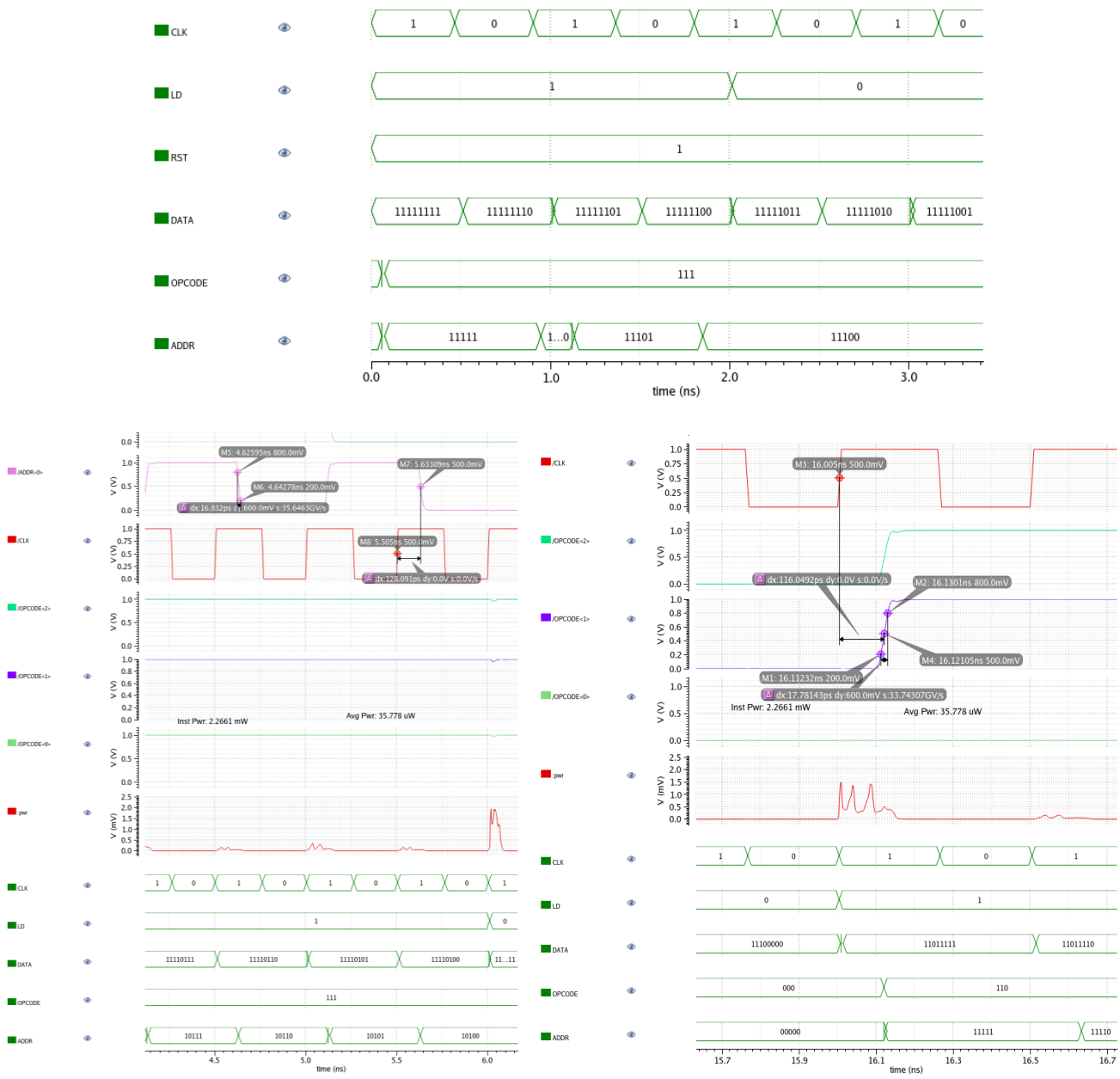


Figure 58. Transient Simulation of the Instruction Memory Demonstrating Proper Opcode and Address Generation.

- ***Performance Analysis:*** The performance of the Instruction Memory was evaluated using transient simulation at the nominal 45 nm operating conditions. The waveform shows that the memory block responds cleanly to the LD and RST control inputs with no observable glitches on either the OPCODE or ADDRESS outputs. Timing behavior remains stable across all switching intervals, with the internal registers correctly capturing new values only during LD = 1. The circuit demonstrates fast settling, with output transitions completing well within the clock period, confirming that the memory system can reliably operate at high frequencies used in the CPU datapath. Power measurements indicate an instantaneous switching peak of approximately 2.2661 mW, occurring during simultaneous bit transitions, while the average power consumption is extremely low at around 35.778 µW, demonstrating efficient static operation between load events. The clean timing edges, consistent bit-field extraction, and low power profile collectively indicate that the Instruction Memory meets the required performance for stable and energy-efficient integration within the CPU's instruction fetch stage.

Table 11. Performance summary of Instruction Memory Block

Frequency	Rise Time	Fall Time	Rising Propagation Delay	Falling Propagation Delay	Average Propagation Delay	Average Power	Instantaneous Power	No. of Transistors	EDP
2 GHz	17.781 ps	16.823 ps	116.049 ps	128.091 ps	122.07 ps	35.778 uW	2.2661 mW	528	0.0533 pJ·ps

J. SRAM CELL:

- ***Module Functionality:*** The 6-transistor (6T) SRAM cell provides a single-bit storage element used for building larger memory arrays in the CPU design. It stores a logic '0' or '1' using two cross-coupled CMOS inverters that maintain the state as long as power is supplied. Read and write operations are enabled through the wordline (WL), which activates two access transistors that connect the internal storage nodes (Q and QB) to the bitlines (BL and BLB). During a write operation, BL/BLB drive the cell to the new value, while during a read operation, the cell modulates the bitlines without disturbing its stored state.

- ***Architectural Specification:***

  The SRAM cell follows a standard 6T topology consisting of:
  - Two cross-coupled inverters (M2–M5) forming a bistable latch that holds one bit.
  - Two NMOS access transistors (M0, M1) controlled by WL for read/write access.
  - Differential bit lines BL and BLB used for high-speed and noise-tolerant sensing.

  Key signal roles:
  - BL / BLB → Carries data into/out of the cell during write and read cycles.
  - WL → Enables access transistors for memory operations.
  - Q → Primary storage node used for reading the stored bit.

  This architecture provides strong static noise margin (SNM), low leakage, and reliable read stability suitable for integration into an 8-bit CPU memory subsystem.

Figure 59. Symbol representation of 6T SRAM CELL

- ***Implementation Methodology:***

  The SRAM cell was implemented at the transistor level in Cadence Virtuoso using FreePDK45 CMOS technology. Steps followed:

  1. Design Sizing:
     - Pull-down NMOS devices were sized stronger (W/L = 180/50 nm) to ensure read stability.
     - Pull-up PMOS devices used smaller sizing (W/L = 90/50 nm) for balanced write margin.

- Access transistors were sized (W/L = 120/50 nm) to allow reliable bitline coupling during write and read.

2. Latch Creation:
   The two cross-coupled inverters were connected to form a positive-feedback loop that maintains the bit.
3. Bitline Interface:
   Access transistors connect Q/QB to BL/BLB, ensuring differential operation.
4. Symbol Abstraction:
   A simplified SRAM Cell symbol was created with pins BL, BLB, WL, Q for top-level integration.
5. Transient analysis setup:
   BL, BLB, and WL waveforms were applied to simulate write and read cycles over a 100 ns time window.

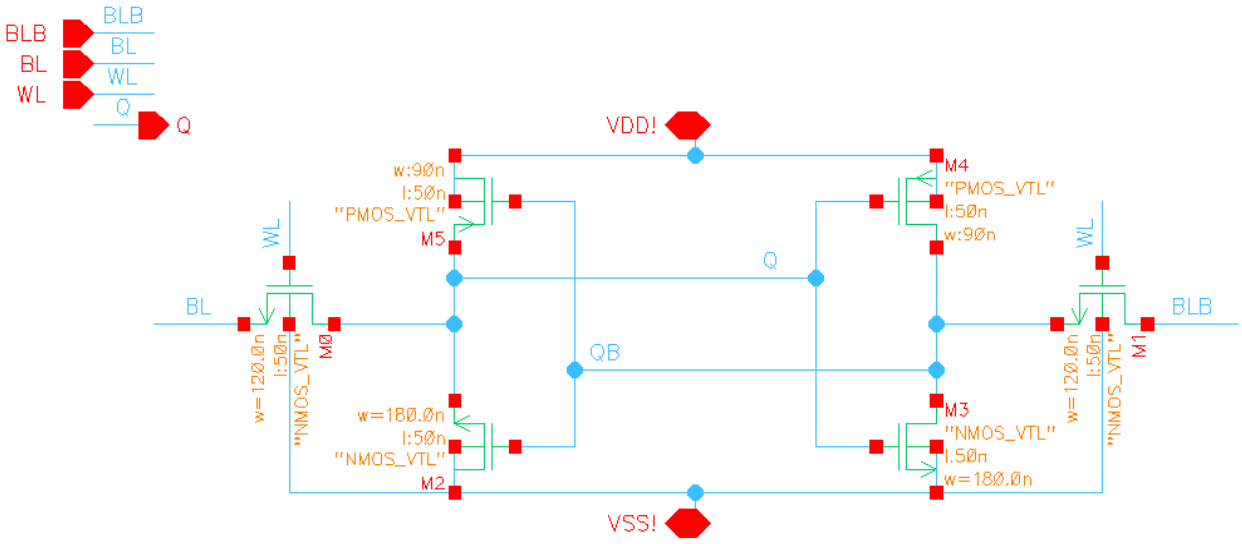


Figure 60. Transistor-Level Schematic of 6T SRAM CELL

- ***Functional Behavior:***

Write Operation:

- WL = 1 enables the access transistors.
- BL/BLB are driven externally (for example, BL = 1, BLB = 0 writes a logical '1').
- The stronger bitline drivers overpower internal inverter feedback and force the cell into the new state.
- After WL returns to 0, the cross-coupled inverters latch the new value.

Read Operation:

- BL and BLB start precharged high.
- WL = 1 connects the storage node to the bitlines.
- The stored bit causes one bitline to discharge slightly, allowing a sense amplifier (externally) to detect the bit.
- The cell maintains stored data without being disturbed due to proper transistor sizing.

The stored state (Q or QB) remains stable as long as VDD is supplied.

- ***Verification and Simulation Analysis:***

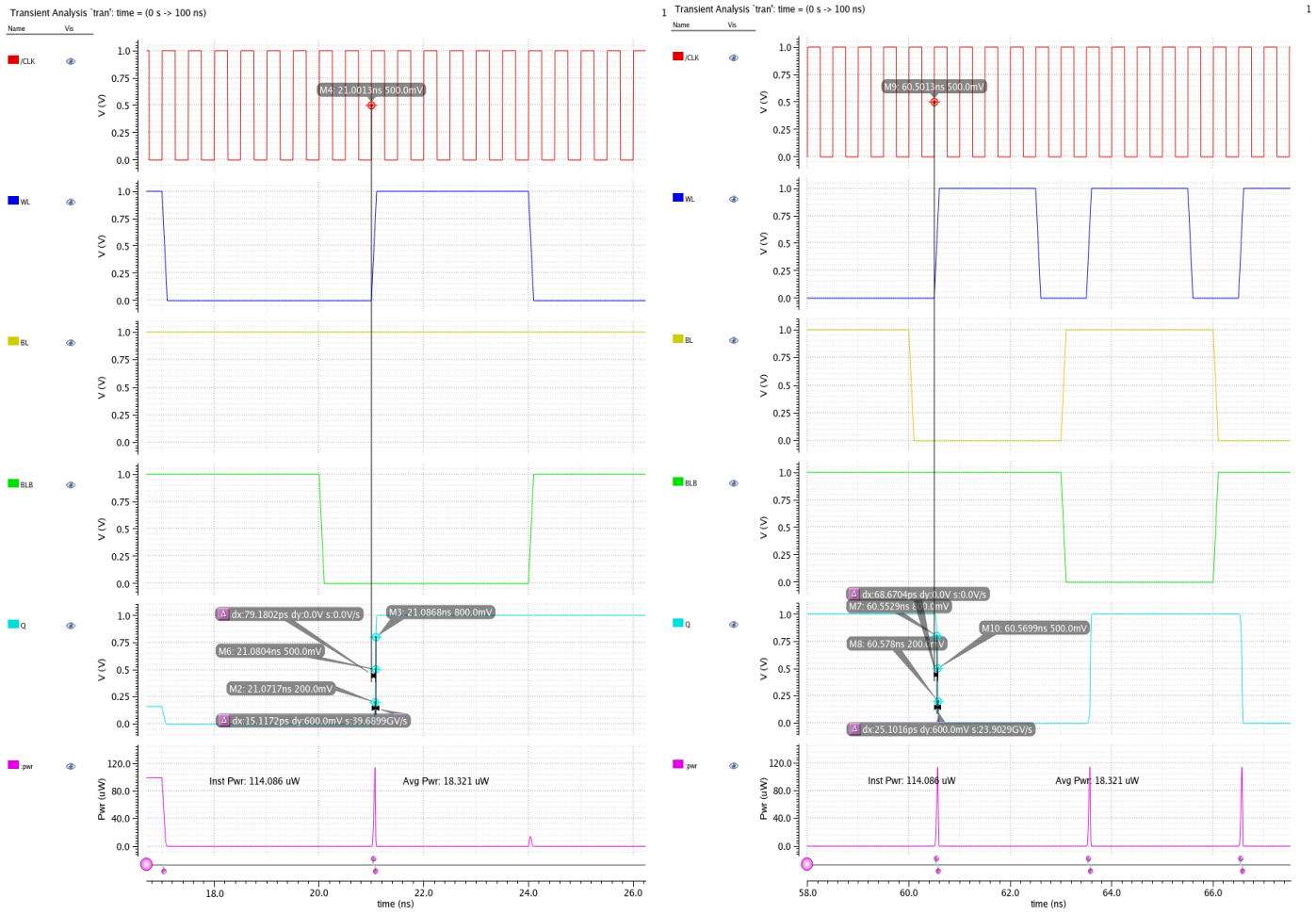


Figure 61. Transient waveform plots of the 6T SRAM cell demonstrating correct write and read behavior, including wordline control, bitline transitions, and stable retention of the stored node Q during memory operations.

Transient simulations were performed to verify write, hold, and read functionality.
Write Cycle Observation:

- When WL = 1 and BL = 1, BLB = 0, the cell successfully transitions and stores logic '1'.
- Similarly, applying BL = 0, BLB = 1 writes logic '0'.
- Q node clearly shifts to 1 or 0 in the waveform around 60.50 ns, confirming successful overwriting.

Read Cycle Observation:

- With WL = 1 and bitlines precharged, Q correctly transfers a small differential voltage to BL/BLB.
- Read operation does not corrupt stored data, validating cell stability.

Hold Condition:

- When WL returns to 0, Q remains static despite BL/BLB transitions, showing strong static retention.

These simulations confirm correct read/write/hold performance and proper transistor sizing for stability.

- ***Performance Analysis:*** The performance of the 6T CMOS SRAM cell was evaluated through transient simulations focusing on its read and write behavior, stability, and switching characteristics. During the read operation, the cell demonstrated reliable access behavior with a clear separation between the high and low storage levels, confirming adequate static noise margin and proper functioning of the cross-coupled inverter pair. The write operation successfully overpowered the stored state when the wordline was enabled, indicating that the chosen transistor sizing provides sufficient write ability without compromising read

stability. The bitline precharge scheme ensured consistent readout, while the differential bitline discharge pattern verified correct sensing behavior. Wordline activation and pulse timing allowed stable transitions without unintended disturbances to the stored value when inactive. Finally, the power profile showed short switching spikes associated with read/write events and low steady-state consumption when the cell was idle, validating the efficiency of the design for memory array integration.

Table 12. Performance summary of 1-BIT SRAM CELL

Parameter	Value
Rise Time	15.117ps
Fall Time	25.101ps
Average Propagation Delay	73.92ps
Instantaneous Power	114.086 µW
Average Power	18.321 µW
NMOS Pull-Down Size	W = 180 nm, L = 150 nm
No of Transistors	6
NMOS Access Size	W = 120 nm, L = 150 nm
PMOS Pull-Up Size	W = 90 nm, L = 150 nm

K. SRAM ARRAY:

- ***Module Functionality:*** The SRAM Array module provides an 8-bit wide memory structure built by replicating eight identical SRAM Column modules. Each column stores one bit across 32 possible word locations, allowing the array to store an entire 8-bit word at a selected address. The array supports three major operations: write, read, and precharge. During a write cycle, the selected column receives input data D<7:0> and the column enable signals S<31:0> determine which memory row is activated. During a read cycle, the outputs from each column are combined to form the 8-bit word O<7:0>. Precharge signals prepare all bitlines for the next read cycle, ensuring reliable sensing and correct differential operation.

- ***Architectural Specification:***
  The SRAM Array architecture consists of:
  - Eight SRAM column modules, each containing 32 SRAM cells arranged vertically.
  - Address decoder interface through S<31:0>, selecting which row in each column is active.
  - Global control signals:
    – PRE: Precharge control for bitlines.
    – WR: Write-enable signal for programming memory.
    – CLK: Used for timing alignment with CPU operations.
  - 8-bit data input bus D<7:0> used during write cycles.
  - 8-bit data output bus O<7:0> used during read cycles.

  Each SRAM column internally interfaces with its own bitline pair while sharing common control signals. This array organization enables scalable memory capacity, low power consumption, and uniform timing across all bit positions.

Figure 62. Symbol representation of SRAM ARRAY

- ***Implementation Methodology:***

The SRAM Array was constructed in Cadence Virtuoso using FreePDK45 technology by hierarchically instantiating 8 SRAM Column blocks.
Implementation steps included:

1. **Column Replication:**
   Each SRAM Column was duplicated eight times, mapped to bits O<7>…O<0>, and connected to the corresponding data bit inputs D<7:0>.

2. **Control Signal Distribution:**
   PRE, WR, and CLK were broadcast to all columns to maintain synchronous memory behavior across the array.

3. **Address Decode Mapping:**
   The 32-bit select line S<31:0> connects to each column, enabling exactly one row per column during read and write cycles.

4. **Symbol-Level Abstraction:**
   A simplified SRAM Array symbol was created to allow clean top-level CPU integration, exposing only the array-level pins without internal details.

5. **Transient Simulation Setup:**
   Multiple write and read cycles were applied for different addresses, data patterns (00h, FFh, alternating patterns), and precharge sequences to ensure functional correctness.

Figure 63. Schematic of SRAM ARRAY

- ***Functional Behavior:***

  Write Operation:
  - WR = 1 enables writing.
  - One of the address select lines S<31:0> goes high.
  - Corresponding row in each column is activated.
  - D<7:0> is written into all eight columns at that selected row.
  - After WR returns to 0, the stored values remain latched in each 6T cell.

  Read Operation:
  - PRE first drives the bitlines to equal voltage.
  - WR = 0 and selected address line S[i] = 1 activates one row.
  - Stored values are differentially sensed and propagated to array outputs O<7:0>.
  - Data remains stable as long as the row is selected.

  Hold Condition:
  - When no row is selected (all S = 0), every cell isolates its bitline pair.
  - Internal storage nodes retain their values indefinitely under VDD.

  This confirms correct multi-bit memory behavior and proper operation across all columns.

- ***Verification and Simulation Analysis:***

Figure 64. Transient simulation waveform of the 8-bit SRAM Array showing write cycles, precharge operation, address selection (S<31:0>), and correct readout on O<7:0>.

Simulation included sweeping through several address patterns and data values (00h, FFh, alternating patterns). The waveforms show:

- **Correct write behavior:** The output of the selected row updates to match the input data.
- **Stable precharge operation:** All bitlines return to a known state between operations.
- **Accurate read behavior:** Stored data is correctly retrieved at each addressed location without corruption.
- **No unintended switching:** Non-selected rows show no activity, confirming proper address decoding.

Across 1.4 µs of simulation, the array consistently outputs the expected memory values and demonstrates correct sequencing of PRE, WR, D<7:0>, and O<7:0> signals.

- ***Performance Analysis:*** Transient simulations of the 8-bit SRAM array verified correct read, write, and hold behavior across all columns, with stable data retention and no read-disturb effects. Waveforms showed proper bitline activity whenever the wordline was enabled and clean transitions during switching. Power results indicate brief peaks of about 12.6 mW during active cycles, while the average power remains low at 35.64 µW in steady state. The complete array contains 232 transistors and demonstrated uniform, reliable operation suitable for CPU memory integration.

Table 13. Performance Summary of 8-bit SRAM Array

Parameter	Value
Instantaneous Power	12.6 mW
Average Power	35.64 µW
Total Transistors in SRAM Array	232 Transistors

L. SRAM COLUMN:

- ***Module Functionality:*** The SRAM Column module implements a vertical chain of 32 individual 6T SRAM cells and provides storage for one bit across 32 address locations. It accepts a single-bit input $D$and writes this value into the selected cell when WR = 1. During a read cycle, the stored bit is accessed through the bitline pair and forwarded to the output O. The PRE signal precharges the bitlines (BL and BLB) prior to every read cycle, ensuring reliable differential sensing. Functionally, the column acts as the 1-bit slice of the complete 8-bit SRAM array.

- ***Architectural Specification:***
  The internal structure of the SRAM Column consists of:
  - 32 SRAM Cells, each implemented as a 6T structure.
  - Shared bitline pair (BL / BLB) connected vertically through all cells.
  - 32 wordline inputs (S<31:0>), with one active per operation.
  - Precharge network for equalizing BL and BLB before reads.
  - Write-driver stage for forcing D / DB onto BL / BLB during writes.
  - Sense inverter for amplifying the final output O.

  This architecture provides uniform timing, supports reliable memory access, and allows scalable integration within the full SRAM array.

Figure 65. Symbol of SRAM COLUMN

- ***Implementation Methodology:***
  The SRAM Column was developed in Cadence Virtuoso using FreePDK45 CMOS technology. Key steps:
  - Cell Integration:
    – The 6T SRAM cell was instantiated 32 times and connected to common BL/BLB lines.
  - Address Mapping:
    – Each cell's WL input was connected to one of the select signals $S < 31:0 >$.
  - Write Driver:
    – Complementary $D$and DBlines were routed to BL and BLB during write cycles.
  - Precharge Network:
    – PMOS equalization transistors were used to precharge BL and BLB when PRE = 1.
  - Output Buffer:
    – A sense inverter was added to generate the logic-level output O.
  - Transient Simulations:
    – BL, BLB, PRE, WR, D, and S<31:0> were stimulated using realistic timing for verification.

Figure 66. Schematic of SRAM COLUMN

- ***Functional Behavior:***

  Write Operation:
  PRE = 0, WR = 1
  D / DB drive BL / BLB
  Selected WL (S[i] = 1) enables write access
  Cell overwrites its internal value
  When WR returns to 0, retention is preserved by cross-coupled inverters

  Read Operation:
  PRE = 1 → BL and BLB precharged
  PRE → 0, WR = 0
  Selected wordline activates one cell
  Stored bit causes differential discharge
  Sense-inverter produces output O

  Hold Condition:
  All S = 0
  No cell is accessed
  Stored values remain stable indefinitely under VDD

- ***Verification and Simulation Analysis:***

  Waveform analysis confirms:
  - Correct precharge → read → write sequencing
  - Clean bitline discharge when reading the selected cell
  - Reliable write operations with stable switching on BL and BLB
  - No interference with unselected cells
  - Accurate output transitions at the sense inverter
  - Low power spikes only during active switching

  The timing markers show adequate sensing margin and read stability across the full column.

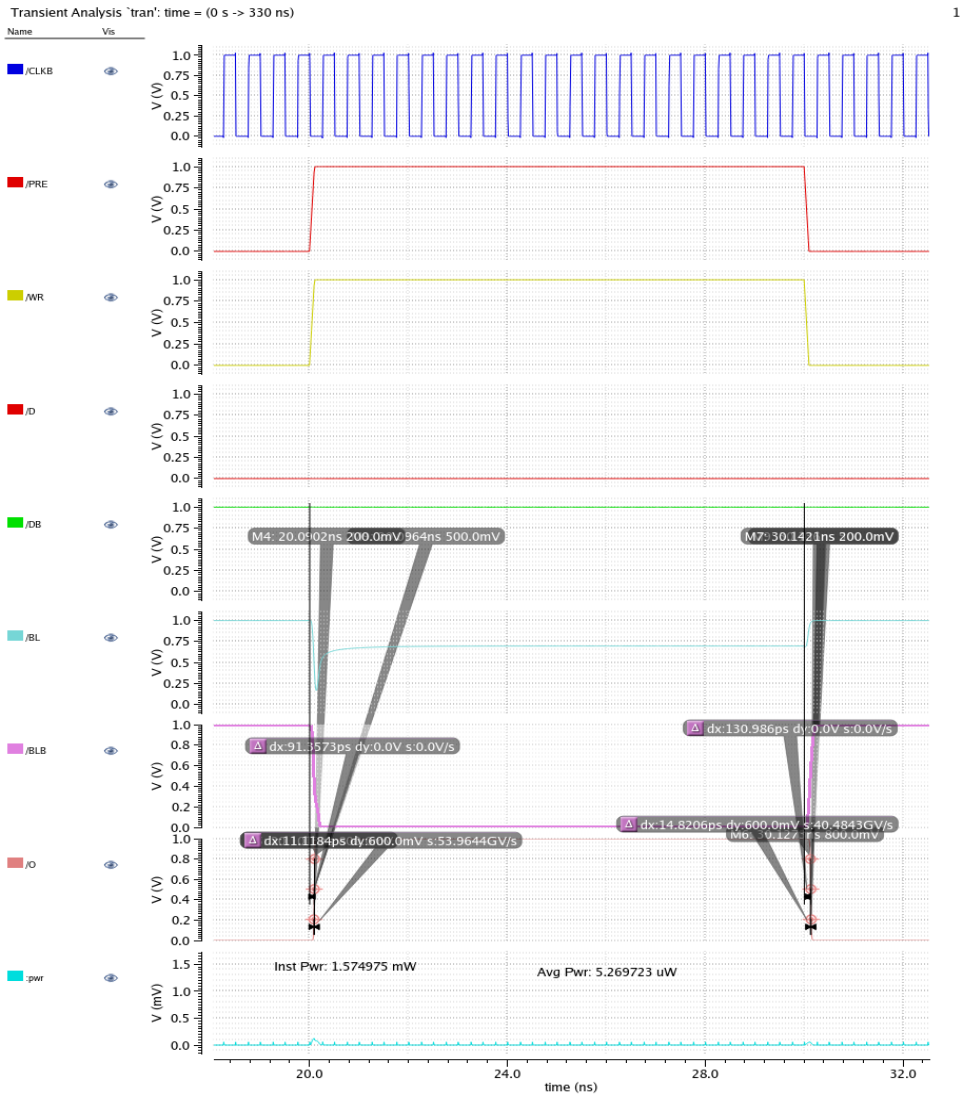


Figure 67. Transient simulation waveform of the 8-bit SRAM Column

- ***Performance Analysis:*** The SRAM Column was evaluated using transient simulations to verify its read, write, and precharge behavior. The bitlines showed clean transitions during write cycles and stable settling during precharge, ensuring reliable sensing during reads. The measured rise and fall times were 11.11 ps and 14.82 ps, and the average propagation delay was 111.184 ps, confirming fast and stable switching across all 32 cells. Power analysis showed short dynamic peaks, with instantaneous power of 1.574975 mW and a low average power of 5.269723 µW during steady operation. The complete column contains 200 transistors and demonstrated uniform performance across all rows, validating correct sizing of bitlines, wordlines, and access devices.

Table 14. Performance Summary of SRAM Column

Instantaneous Power	Average Power	Rise Time	Fall Time	Average Propagation Delay	Total Transistors in SRAM Column
1.574975 mW	5.269723 µW	11.11 ps	14.82 ps	111.184 ps	200 Transistors

V. RESULTS AND DISCUSSION

This work focuses on evaluating whether the transistor-level implementation of the 8-bit RISC CPU meets the intended architectural and performance goals defined at the beginning of the design. The discussion emphasizes system-level behavior, including datapath correctness, control signal coordination, timing alignment, and power trends across integrated modules. While individual blocks such as the ALU, accumulator, decoder, instruction pointer, and memory units operate correctly in isolation, full CPU integration highlights the importance of precise clock synchronization across the fetch, decode, and execute stages. These observations provide insight into the strengths of the hierarchical design approach as well as the challenges associated with integrating multiple synchronous and combinational subsystems at the transistor level.

Transient simulation results confirm that the CPU achieves stable operation at an effective clock frequency of approximately 2 GHz using FreePDK45 CMOS technology. The measured average power consumption of approximately 0.93 mW closely matches the initial design estimate, while total power consumption during full operation remains within the expected 13–14 mW range. The overall gate count of approximately 4500 gates is significantly lower than the initial upper-bound estimate, reflecting efficient logic implementation and reuse of optimized sub-blocks. All simulations were performed using Cadence Virtuoso and ADE Explorer, and the results verify correct functional behavior for arithmetic, logical, control, and memory-related operations under typical operating conditions.

Table 15 summarizes the target specifications and achieved performance metrics of the processor.

Table 15. Design Targets and Achieved Performance of the 8-bit RISC CPU

Parameter	Estimated Value	Calculated Value	Description
Technology Node	FreePDK45 (45 nm CMOS)	FreePDK45 (45 nm CMOS)	Implemented using open-source 45 nm standard cell library.
Operating Frequency	≈ 2GHz	≈ 2GHz	Achieved single-cycle execution speed with stable operation between 2GHz – 3GHz.
Average Power	≈ 1mW	≈ 0.93mW	The average power in all the instances
Gate Count	Up to 10,000 gates	Up to 4500 gates	Includes ALU, accumulator, program counter, decoder, instruction register, and control logic.
Total Power Consumption	≈ 13–14 mW	≈ 13–14 mW	Combined static and dynamic power consumption for full processor operation.
Design Tools	Cadence Virtuoso / ADE Explorer	Cadence Virtuoso / ADE Explorer	Used for schematic design, simulation, and verification.

VI. CONCLUSION

This project successfully demonstrated the transistor-level design and hierarchical integration of an 8-bit RISC CPU using FreePDK45 CMOS technology in Cadence Virtuoso. The processor architecture was organized around a standard fetch–decode–execute cycle, with coordinated interaction between the instruction memory, decoder, program counter, ALU, accumulator, and memory blocks. Each stage was designed to operate synchronously under a common clock while preserving modularity for verification and debugging.

The design process highlighted several important architectural insights. Within the ALU, the logical shifter was identified as the critical path due to its cascaded multiplexer structure, which dominates the overall propagation delay. Additionally, the inclusion of a multi-stage ring oscillator provided practical understanding of on-chip clock generation and delay modeling, reinforcing the relationship between inverter delay and system frequency. Overall, the project provided valuable experience in managing timing,

control synchronization, and transistor-level trade-offs in a complete CPU implementation, establishing a strong foundation for more advanced processor architectures in future work.

VII. FUTURE WORK

Future extensions of this project will focus on achieving full functional integration of the CPU, including stable synchronization across the instruction fetch, decode, execute, and memory stages. Additional refinement of timing across modules—particularly the ALU, program counter, and memory blocks—will help eliminate race conditions and ensure correct end-to-end operation. Architectural enhancements such as introducing a pipelined datapath, expanding the instruction set, and adding branch and jump support can significantly improve performance and flexibility. Furthermore, the current 8-bit architecture may be extended to 16-bit or 32-bit versions by scaling the datapath, ALU, memory, and control logic, enabling more complex computations. These improvements provide a clear roadmap for developing a more capable and efficient CPU in future work.

VIII.ACKNOWLEDGMENT

I would like to express my sincere gratitude to Dr. Saiyu Ren for her continuous guidance, support, and encouragement throughout the duration of this project. Her feedback and direction were instrumental in completing the design successfully. I, Keerthi Patil, contributed to the logical AND/NOT units, increment/decrement blocks, instruction pointer, decoder, ALU and CPU integration, and lower-level hierarchy circuits. I also acknowledge Rob, who worked on the shifter, XOR block, memory unit, instruction memory, lower-level hierarchy modules, and ALU integration, and Ramya, who contributed to the adder/subtractor, multiplier, accumulator, 8-bit register, multiplexers, ALU integration, CPU integration, and additional lower-level circuits.

IX. 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

[4] S. Narayanan and J. Selva Kumar*, "Performance Analysis of Ring Oscillator in Sub-Micron CMOS Technology,"* *International Journal of Applied Engineering Research*, vol. 10, no. 4, pp. 9601–9610, 2015. http://www.ripublication.com/ijaer10/ijaerv10n4_101.pdf

[5] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Boston, MA, Addison-Wesley, 2011.