1. OBJECTIVE

The arithmetic–logic unit (ALU) is the central computational component of any processor, responsible for executing numerical and logical operations that enable instruction-level functionality. In this project, an 8-bit ALU was designed and implemented at the transistor level to explore how arithmetic and logic operations can be realized using fundamental digital building blocks. The ALU supported four arithmetic functions - addition, subtraction, increment, and decrement and four logic functions - AND, OR, complement, and identity. These operations were controlled through a mode input *M* and select lines *S1* and *S0*, while full-adder stages formed the core computational structure. To meet synchronous design requirements, all inputs and outputs were driven through clocked D flip-flop registers, ensuring stable timing behavior across the entire circuit.

The primary goal of the project was to develop a fully functional 8-bit ALU that operated correctly and also demonstrated an efficient power–delay–area (PDA) trade-off. Transistor-level schematics, hierarchical bit-slice structures, and register integration were constructed to build the complete design. Simulation testbenches were used to verify correct operation, observe glitch behavior, determine worst-case delay, and measure power consumption under critical input conditions. This project provided practical experience with synchronous digital design, hierarchical circuit construction, and performance evaluation using industry-standard EDA tools, ultimately demonstrating a complete workflow from logic design to timing and power analysis.

2. DESIGN METHODOLOGY

The design process began by establishing the behavioral specifications of a one-bit ALU. Initially, the required circuits were developed using the fundamental logic gates constructed in previous laboratory assignments, ensuring consistency with earlier design methodologies. A complete truth table was formulated for all arithmetic and logic operations to determine how each input needed to be modified. Based on this truth table, Karnaugh maps were generated for the arithmetic extender (AE), logic extender (LE), and carry-out (CO) logic blocks. The simplified Boolean expressions obtained from K-map minimization were then implemented at the transistor level to create the AE, LE, and CO blocks. Each of these blocks was simulated individually to verify correct logical behavior. After confirming proper operation of these extenders, the full-adder stage was incorporated using previously designed compound-gate full adders, allowing arithmetic operations to be integrated seamlessly into the ALU structure.

Once the extender blocks and full adder were verified, the synchronous elements required by the project specification were designed. A one-bit D flip-flop register and an eight-bit register were created to support clocked input and output behavior. These registers were first validated independently and were then integrated into the ALU design. The next step involved constructing a complete one-bit ALU by combining the AE, LE, full adder, and register modules. The one-bit ALU was initially simulated without registers to verify pure combinational correctness, and was then tested again with input and output registers to confirm proper synchronous operation.

After confirming the correctness of the one-bit ALU, the design was expanded into an eight-bit ALU through hierarchical replication of the bit-slice architecture. The eight-bit ALU was first simulated without registers to verify functionality across all eight slices. Register stages were then added incrementally, beginning with output-only registers and later including both input and output registers to fully satisfy the synchronous design requirement. Each version of the eight-bit ALU was simulated to validate logical correctness, observe and eliminate glitches, and ensure stable timing behavior. Final simulations focused on determining the worst-case delay, measuring power consumption, and gathering area statistics, all of which were used to compute the overall Power–Delay–Area (PDA) efficiency of the final design.

FLOOR PLAN

The eight-bit ALU was organized using a regular bit-slice floorplan. Each slice contains an Arithmetic Extender (AE), a Logic Extender (LE), and a full-adder (FA) arranged horizontally in sequence. All eight slices were placed side-by-side to form the complete datapath, with the ripple-carry chain flowing from the least significant slice to the most significant slice. Input registers for A and B were positioned on the left of the datapath to feed all slices in parallel, while output registers for F[7:0], Cout, and OV were placed on the right to capture the final results synchronously. This floorplan minimizes routing complexity, shortens carry paths, and produces a compact schematic suitable for an 8-bit ALU.

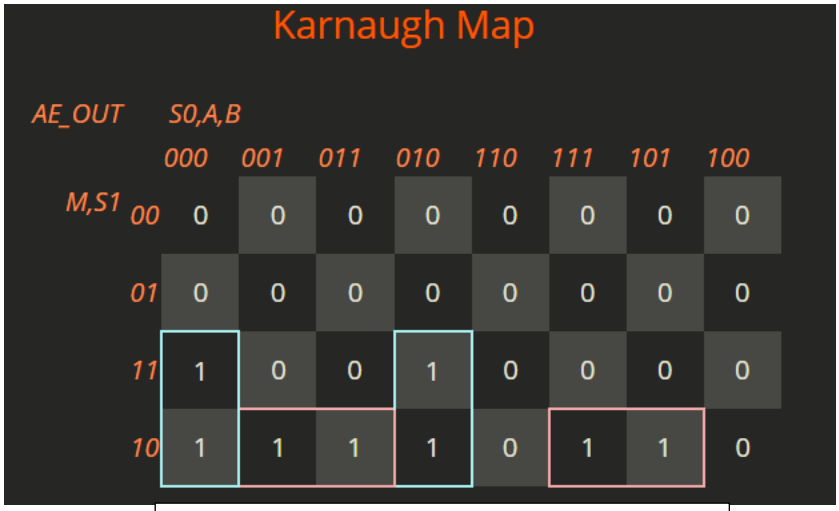
Table 1: The values of M, S1, and S0 for selecting different ALU operations

M	S1	S0	Function	F	X	Y	C0
1	0	0	Decrement	A-1	A	All 1's	0
1	0	1	Add	A+B	A	B	0
1	1	0	Subtract	A+B'+1	A	B'	1
1	1	1	Increment	A+1	A	All 0's	1
0	0	0	Complement	A'	A'	0	0
0	0	1	AND	A AND B	A AND B	0	0
0	1	0	Identity	A	A	0	0
0	1	1	OR	A OR B	A OR B	0	0

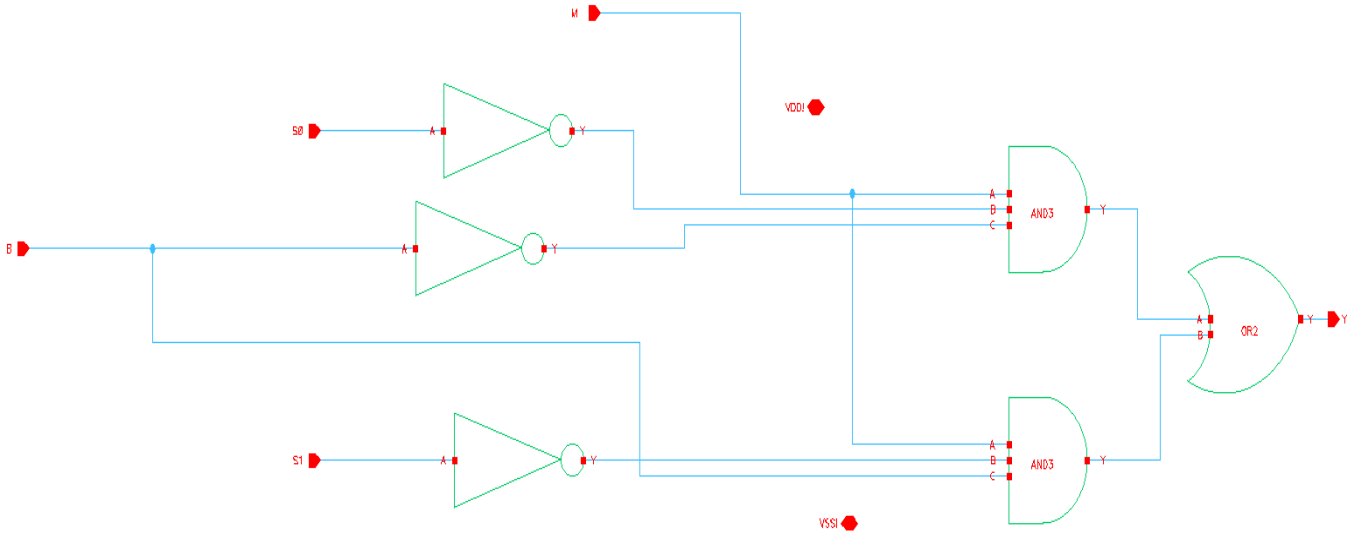
Table 2: 1-bit truth table for different ALU operations

M	S1	S0	A	B	X(LE OUT)	Y(AE OUT)	C0
0	0	0	0	0	1	0	0
0	0	0	0	1	1	0	0
0	0	0	1	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	1	0	0	0
0	0	1	1	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0
0	1	0	1	0	1	0	0
0	1	0	1	1	1	0	0
0	1	1	0	0	0	0	0
0	1	1	0	1	1	0	0
0	1	1	1	0	1	0	0
0	1	1	1	1	1	0	0
1	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0
1	0	0	1	0	1	1	0
1	0	0	1	1	1	1	0
1	0	1	0	0	0	0	0
1	0	1	0	1	0	1	0
1	0	1	1	0	1	0	0
1	0	1	1	1	1	1	0
1	1	0	0	0	0	1	1
1	1	0	0	1	0	0	1
1	1	0	1	0	1	1	1
1	1	0	1	1	1	0	1
1	1	1	0	0	0	0	1
1	1	1	0	1	0	0	1
1	1	1	1	0	1	0	1
1	1	1	1	1	1	0	1

AE (ARITHMETIC EXTENDER)- BLOCK IMPLEMENTATION



AE_OUT = (M & ~S1 & B) | (M & ~S0 & ~B);



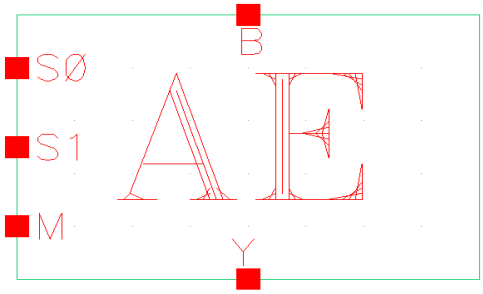


Figure 1: K-map, Boolean expression, schematic, symbol of AE block

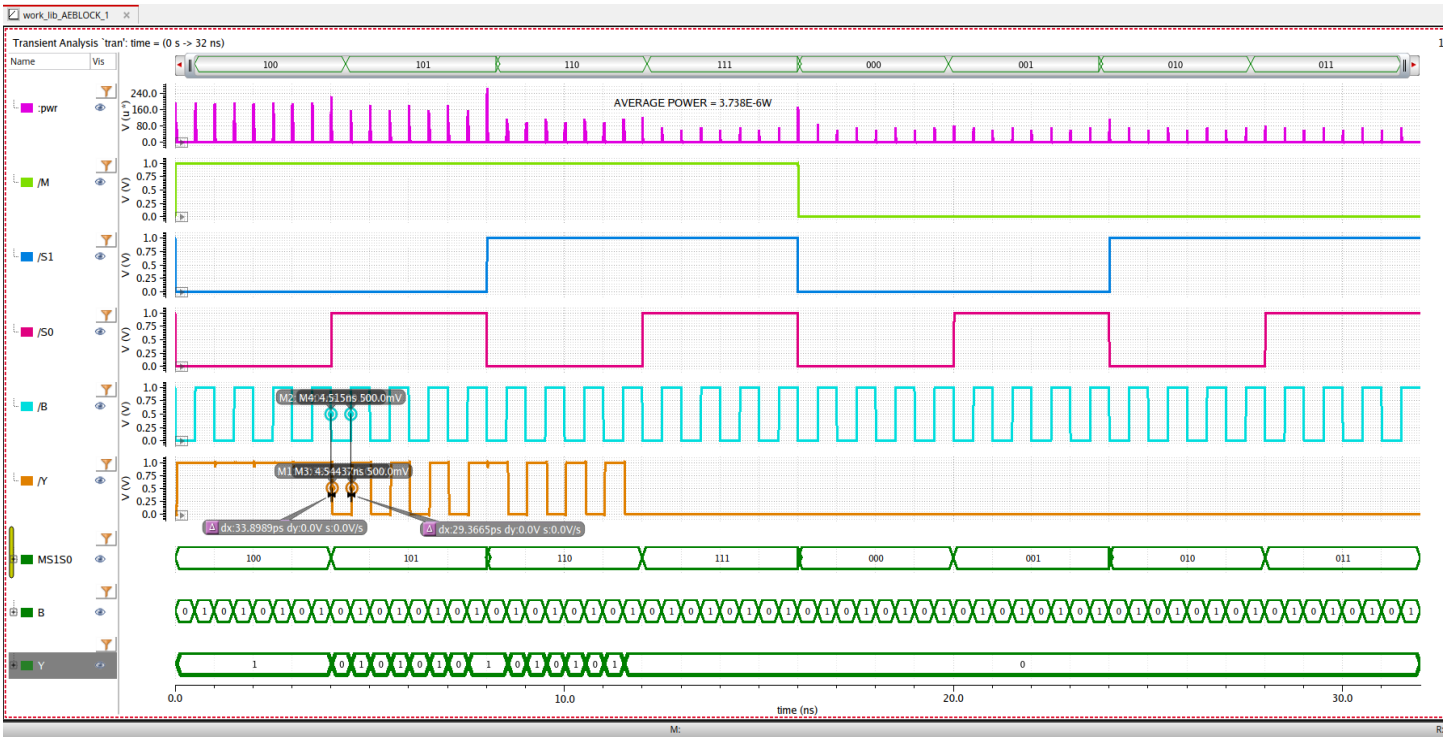
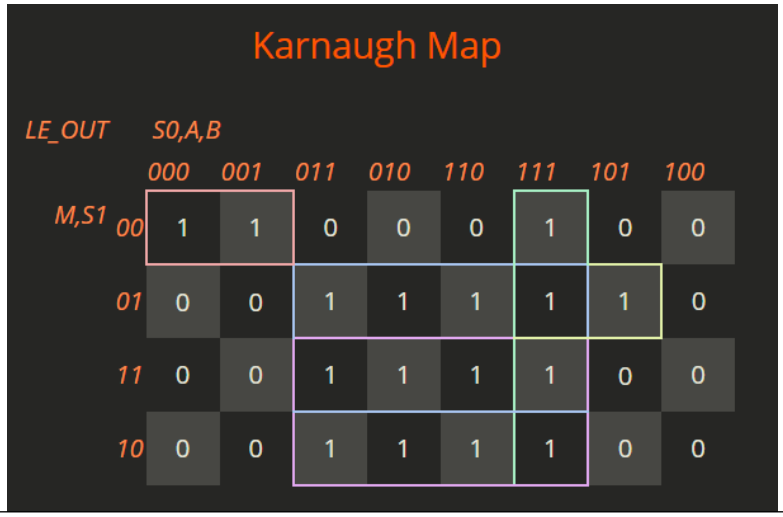


Figure 2: Waveform verifying AE block

The Arithmetic Extender (AE) block plays a critical role in the ALU by modifying the B-input before it enters the full-adder chain, directly influencing the overall power, delay, and area performance of the design. Because the AE is active during all arithmetic operations, its logic depth and switching activity affect both the propagation delay at the ALU output and the total power consumption. The AE logic was derived through K-map minimization to reduce unnecessary gate usage, which helped minimize area and internal transitions. After implementation in Cadence using transistor-level schematics, the functionality of the AE block was verified using transient simulations.

The waveform confirms that the AE produces the correct modified B-input for each arithmetic mode based on the control signals *M*, *S1*, and *S0*. When M = 1, the ALU enters arithmetic mode, and the AE output Y responds exactly according to the Boolean function implemented. The simulation clearly shows that the AE passes B unchanged during addition, generates the bitwise complement of B during subtraction, and produces constant logic values for increment and decrement operations depending on the select-line combination. These transitions appear clean and synchronized, with no observed glitches. Overall, the AE block operated correctly across all test cases and contributed minimal overhead to the critical path of the 8-bit ALU.

LE (LOGIC EXTENDER) - BLOCK IMPLEMENTATION



LE_OUT = (~M & ~S1 & ~S0 & ~A) | (S0 & A & B) | (S1 & A) | (~M & S1 & S0 & B) | (M & A);

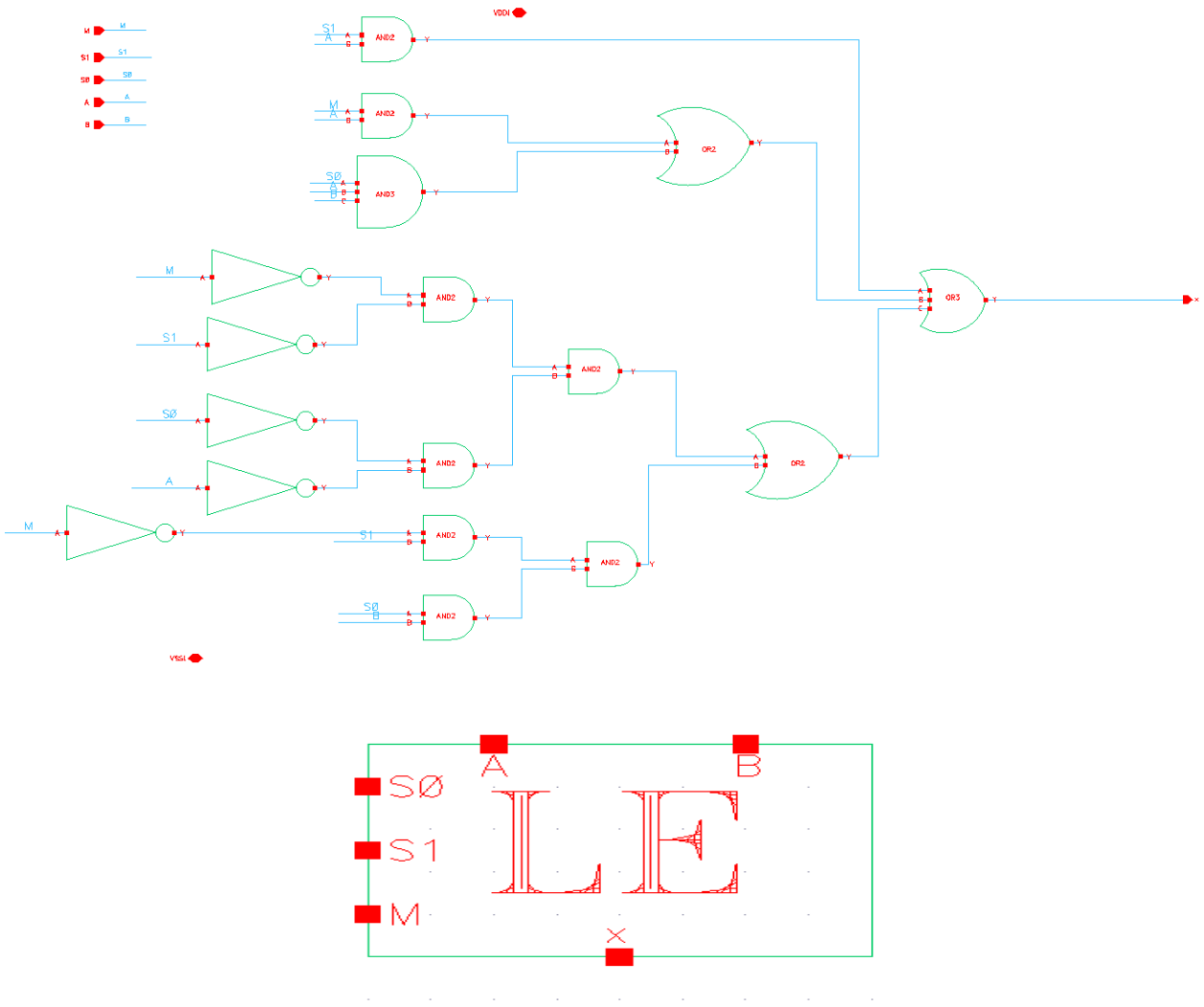


Figure 3: K-map, Boolean expression, schematic, symbol of LE block

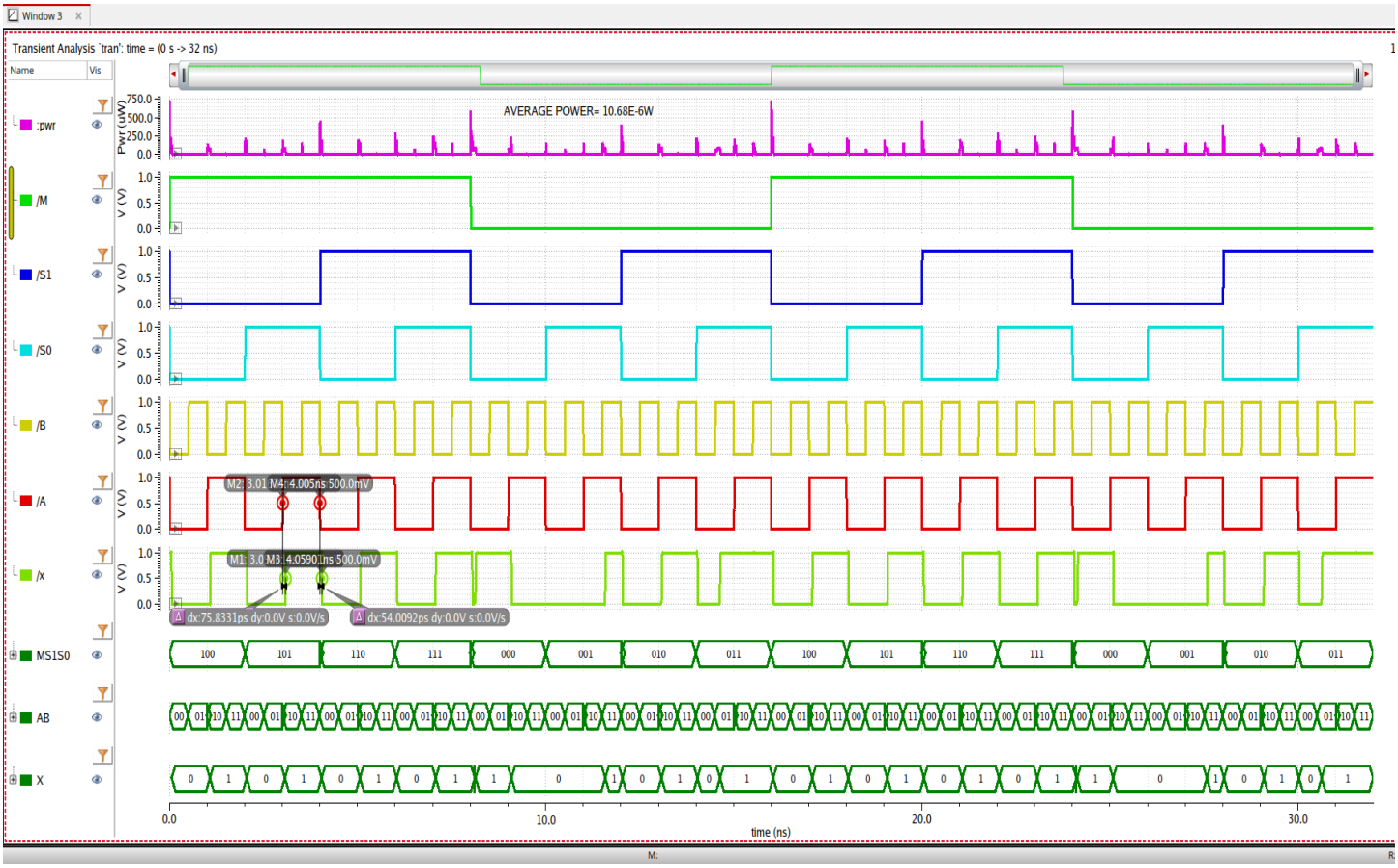
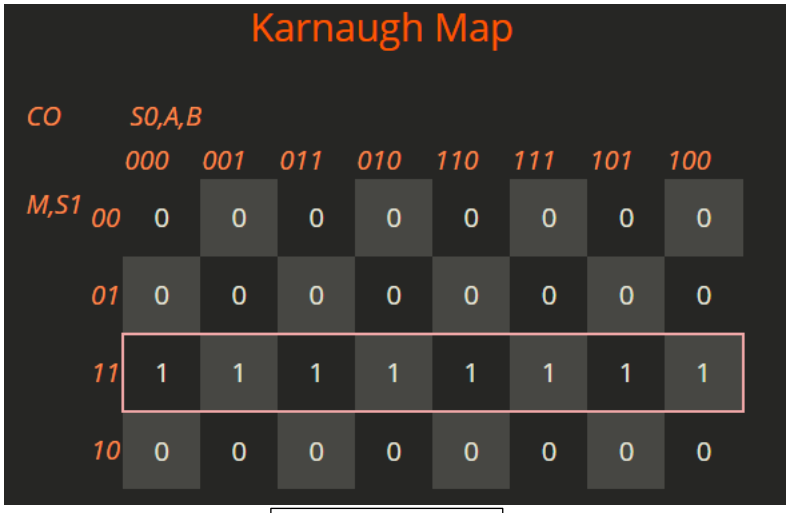


Figure 4: Waveform verifying LE block

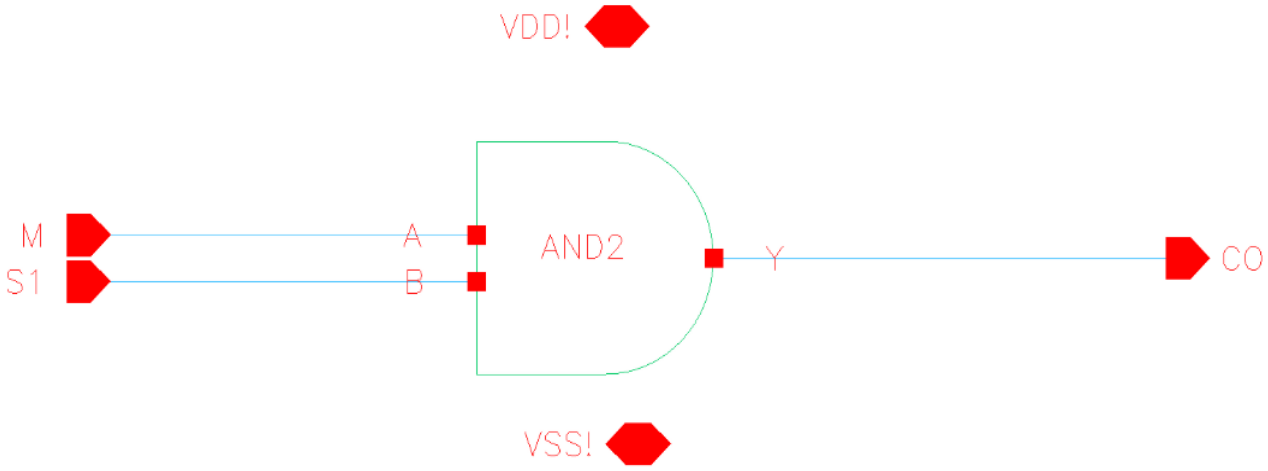
The Logic Extender (LE) block determines the logical output of the ALU when the mode bit M = 0. Based on the select lines S1 and S0, the LE generates the correct logic function such as identity (A), complement (~A), AND, or OR. The LE logic was derived using K-map minimization to reduce gate count and ensure efficient transistor-level implementation. This minimized structure helps reduce area, switching activity, and unnecessary internal transitions. The block was implemented in Cadence using previously established design techniques and verified independently before integration.

The simulation waveform confirms that the LE output X responds correctly to all logic modes. When the select lines change, X follows A for identity, inverts A for complement, and correctly produces A & B or A | B for AND and OR operations. The transitions are clean and free of glitches, demonstrating stable logic behavior. Overall, the LE block functioned exactly as intended and integrated smoothly into the 1-bit and 8-bit ALU designs.

CO (LOGIC BLOCK) – IMPLEMENTATION



CO = (M & S1);



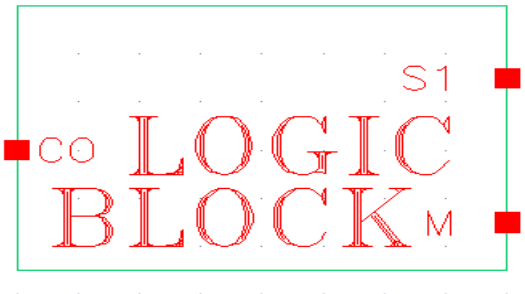


Figure 5: K-map, Boolean expression, schematic, symbol of co block

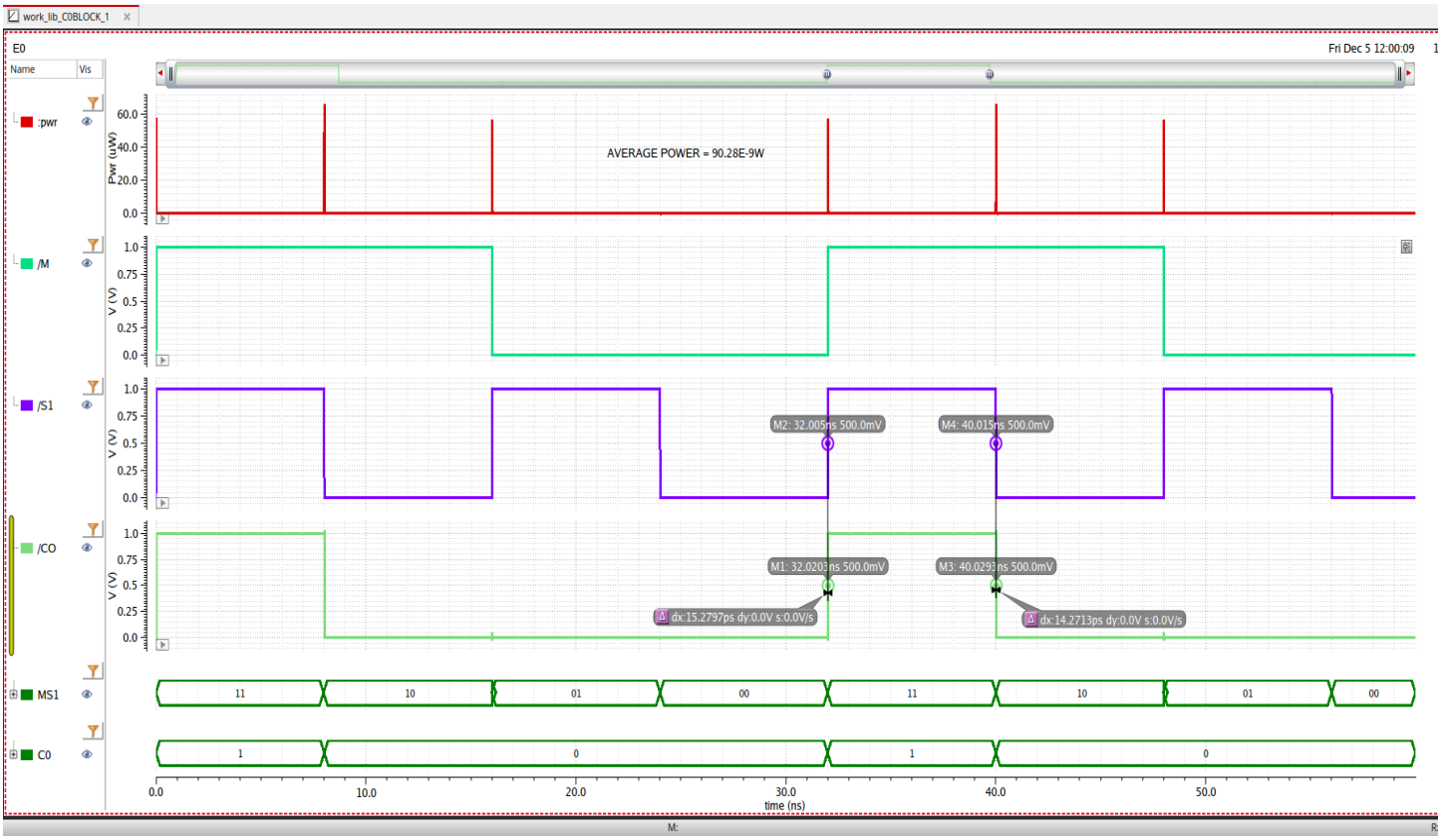


Figure 6: Waveform verifying co block

The Carry-Out (CO) block is responsible for generating the appropriate carry-in signal for the arithmetic operations performed by the ALU. When the mode bit M = 1, the ALU enters arithmetic mode, and the CO logic determines whether the operation requires an initial carry. For example, subtraction in two's-complement form requires a carry-in of 1, while addition and increment operations may not. The Boolean expression for the CO block was simplified using K-map minimization to ensure minimal logic depth and area consumption, allowing the CO signal to propagate quickly into the first full-adder stage. The block was implemented using Cadence transistor-level schematics and verified individually before full integration into the 1-bit and 8-bit ALU designs.

The waveform confirms that the CO output behaves exactly as expected for all combinations of the mode bit *M* and select line *S1*. When M = 1 and S1 = 1, the CO output transitions high, providing the required carry-in for subtraction-type operations. For all other arithmetic modes, the CO output remains low, ensuring correct adder behavior. The transitions are clean and glitch-free, demonstrating stable logical functionality.Overall, the CO block functions correctly and reliably triggers the proper arithmetic mode behavior within the ALU.

FULL ADDER (COMPOUND GATES)

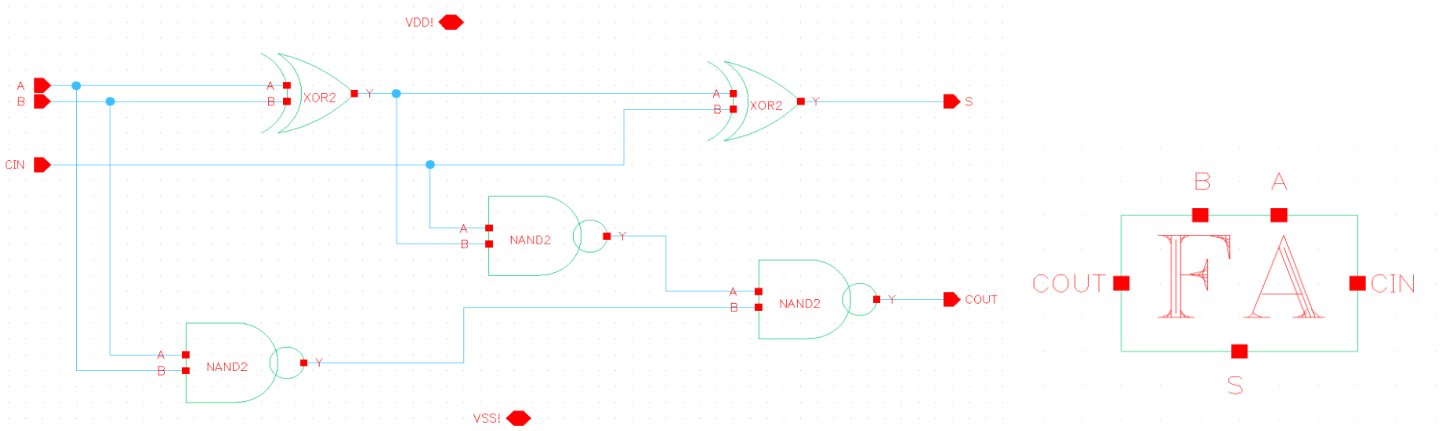


Figure 7: schematic, symbol of FA block

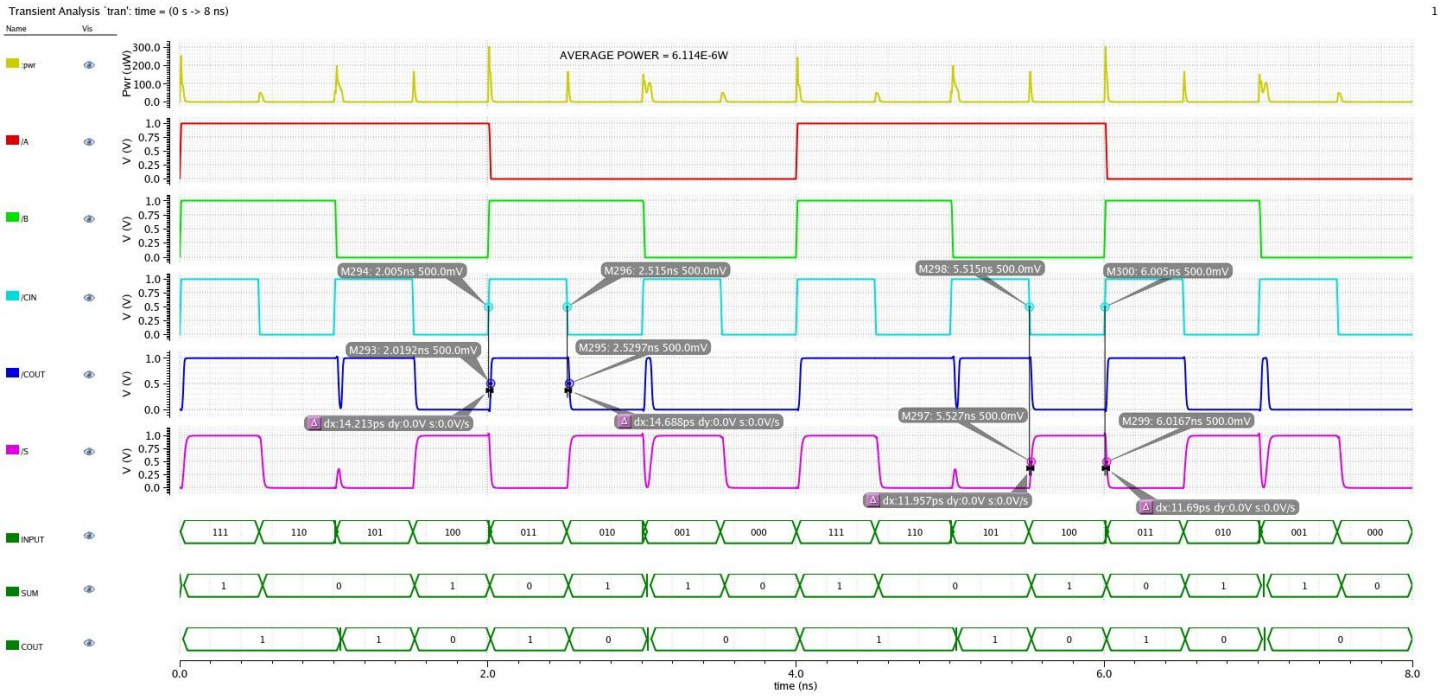


Figure 8: Waveform verifying FA block

The full adder was implemented using optimized compound gates to reduce logic depth, switching activity, and area compared to a traditional gate-level implementation. This FA forms the core arithmetic unit of each ALU bit-slice, producing correct SUM and COUT outputs for all combinations of A, B, and CIN. The waveform confirms accurate addition behavior, including proper carry generation and propagation. The transitions remain clean and stable across all test vectors, demonstrating reliable transistor-level functionality.

D-FF REGISTER (1-BIT)

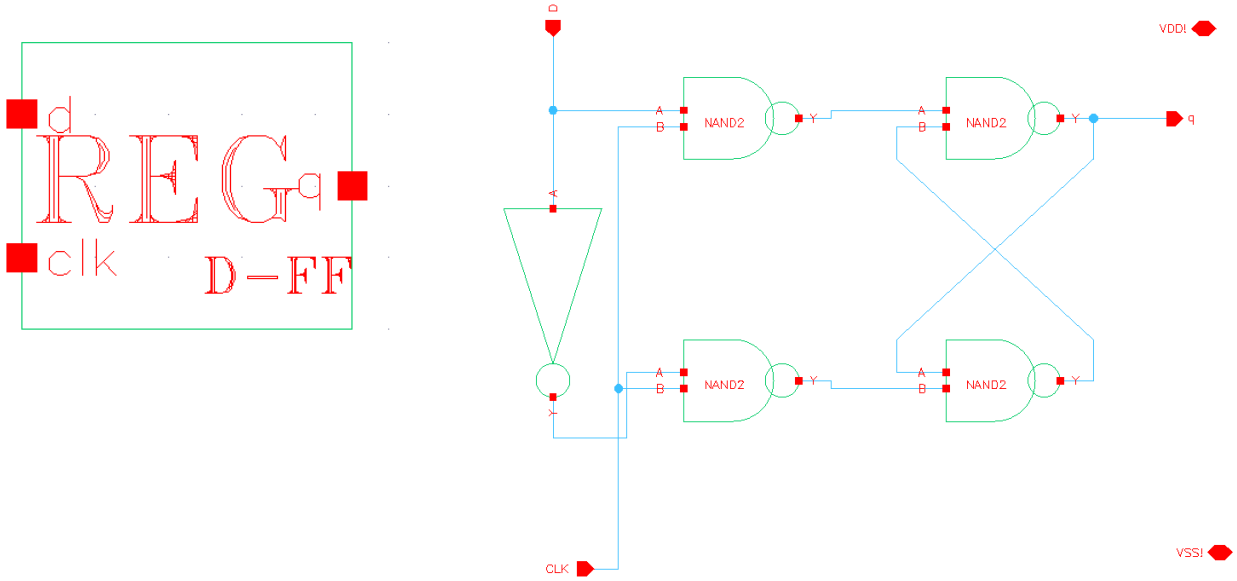


Figure 9: schematic, symbol of 1-bit DFF block

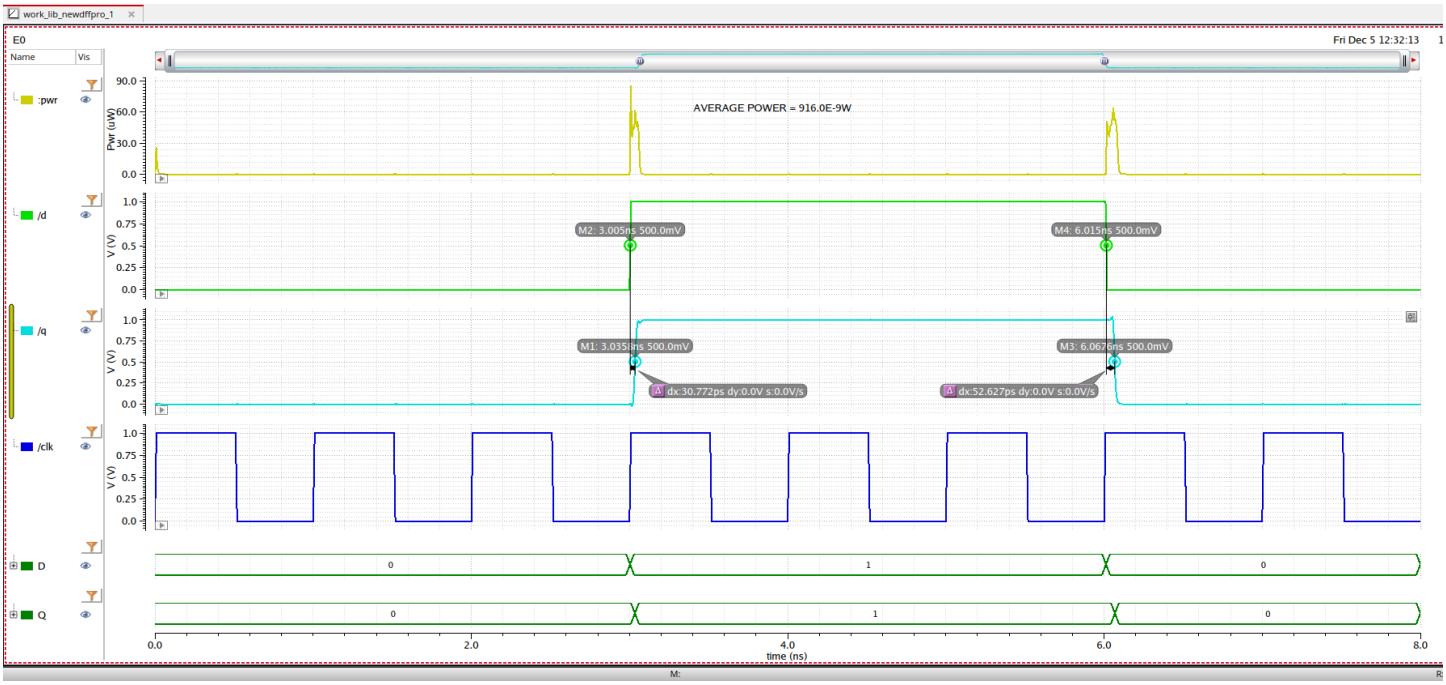


Figure 10: Waveform verifying 1-bit DFF block

The 1-bit D flip-flop is a fundamental storage element in this project, providing the ability to reliably store and hold a single bit of data across clock cycles. It captures the input D during the active clock transition and updates the output Q with a small propagation delay, enabling predictable and controlled data flow inside the system. This makes the DFF essential for building registers, synchronizing signals, and stabilizing the ALU outputs before they are used in the next stage. The simulation waveform clearly shows that Q changes only when the clock triggers confirming proper flip-flop behavior. This stable and clock-driven operation is critical for ensuring correct sequencing and timing throughout the entire 8-bit ALU design.

D-FF REGISTER (8-BIT)

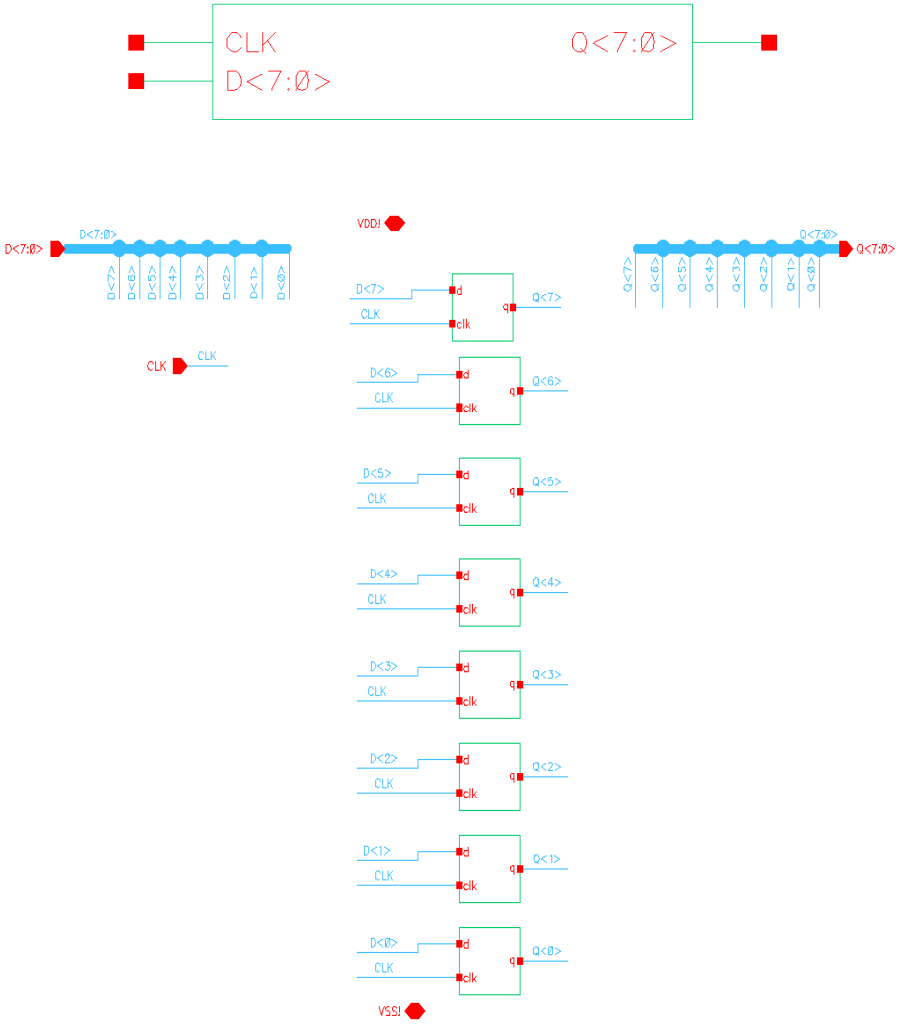


Figure 11: schematic, symbol of 8-bit DFF block

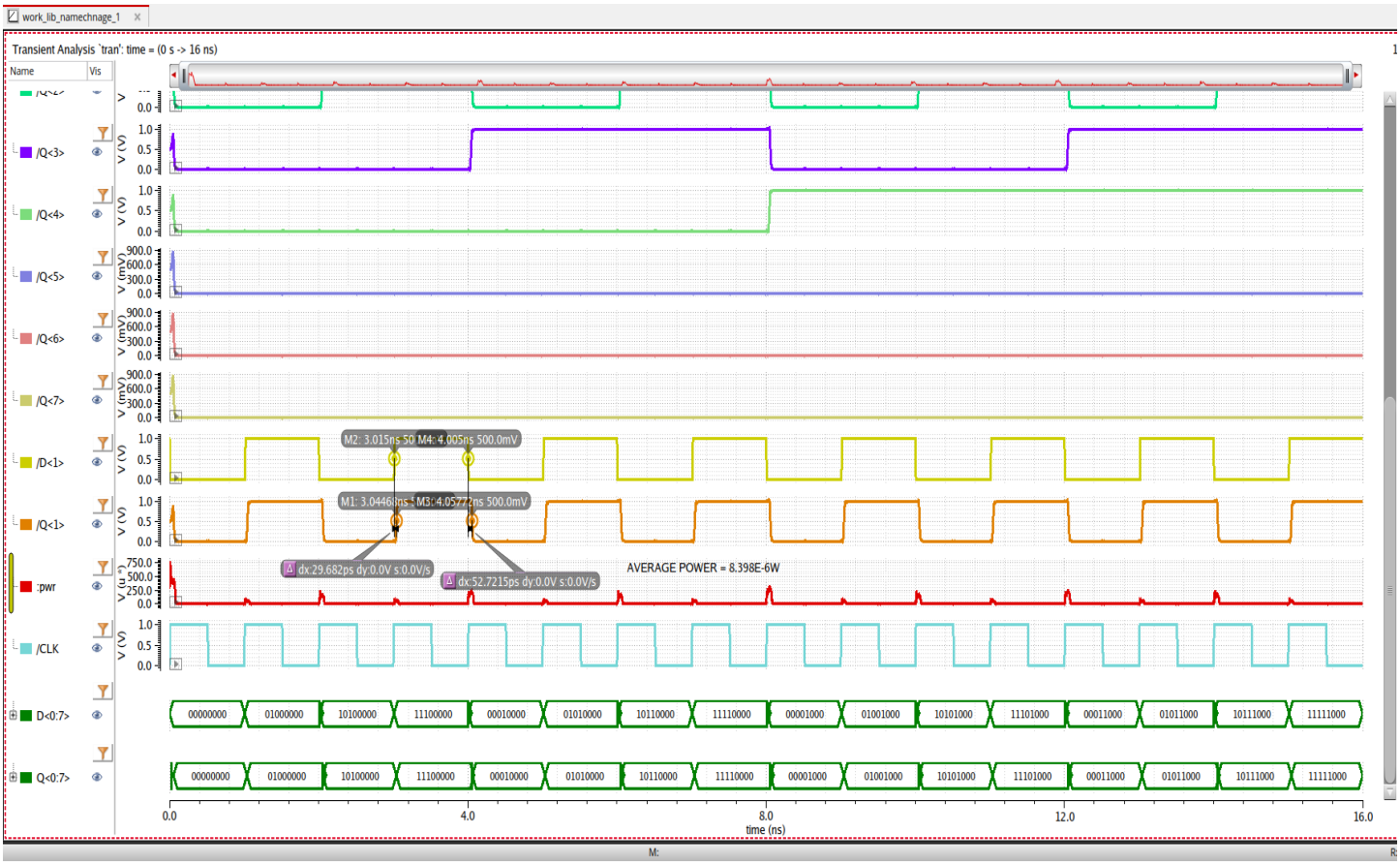


Figure 12: Waveform verifying 8-bit DFF block

The 8-bit register in this project is implemented by cascading eight identical 1-bit D-flip-flop elements, each responsible for storing one bit of the input vector D<7:0>. All flip-flops share a common clock, ensuring that the entire 8-bit word is captured synchronously on each rising clock edge. This makes the register function as a stable memory element that holds data between clock cycles and updates only when the clock triggers.

The waveform clearly demonstrates correct register behavior: whenever D<7:0> changes, the outputs Q<7:0> remain unchanged until the next rising edge of the clock. At every clock transition, all eight output bits update simultaneously to the new input value, confirming proper synchronous loading. The stored data remains constant even if the input changes mid-cycle, illustrating the register's role in providing clean, glitch-free, time-aligned storage for the system. This 8-bit register is essential in the project because it allows the ALU or other blocks to preserve intermediate values and maintain state for correct operation.

1-BIT ALU WITHOUT REGISTERS

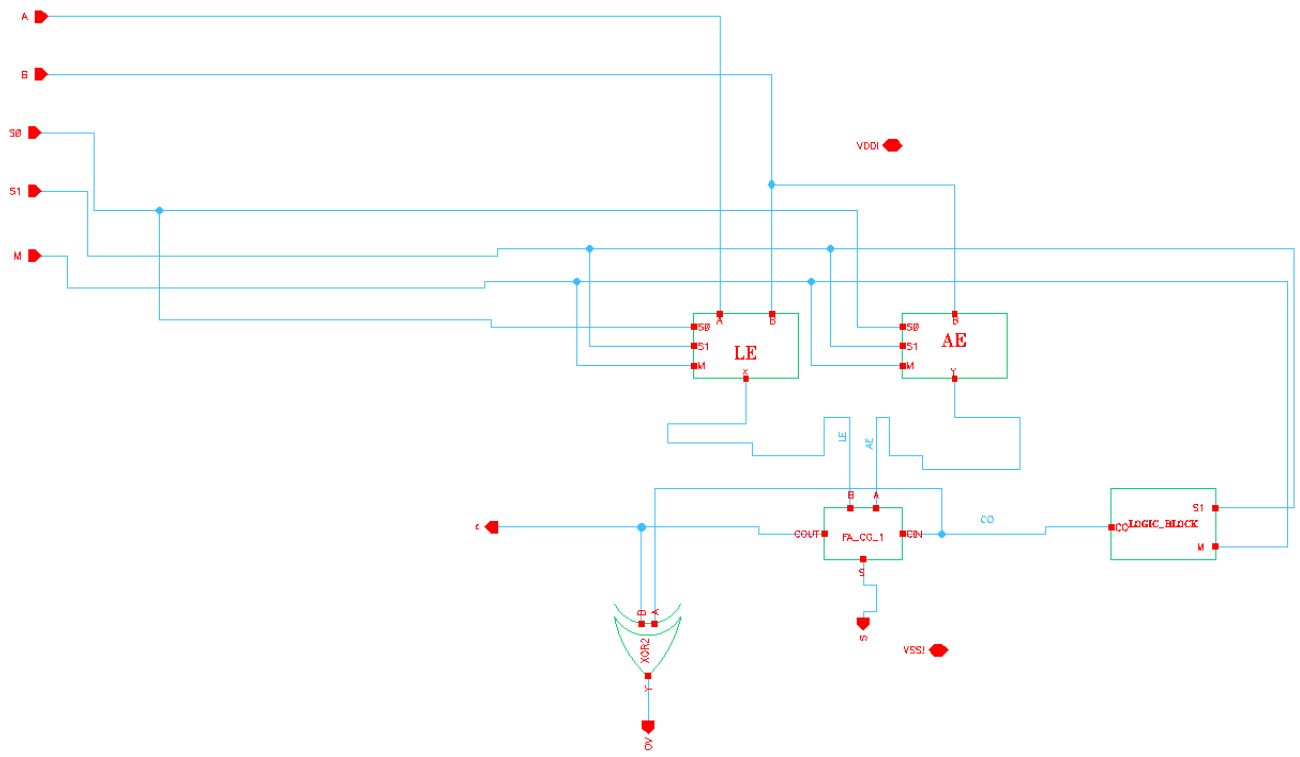


Figure 13: schematic of 1-bit ALU block

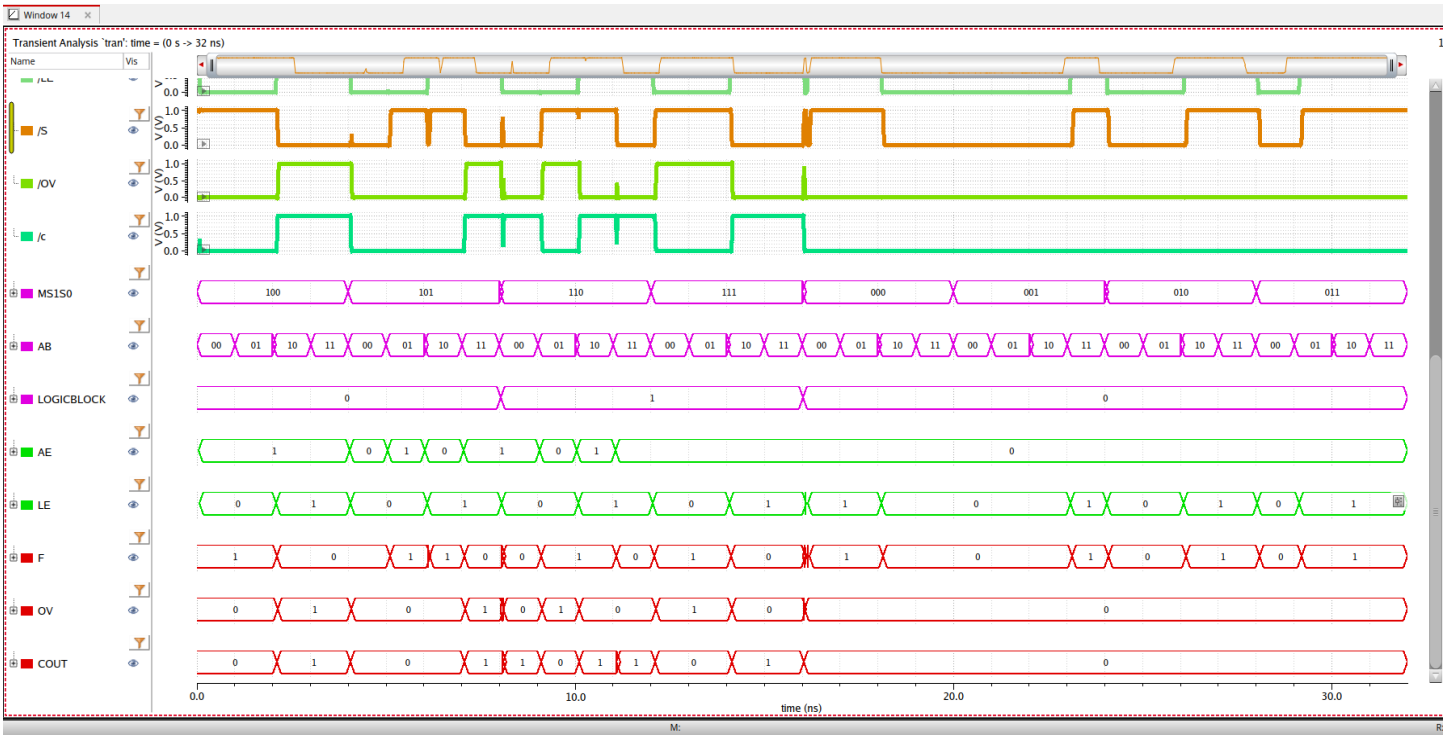


Figure 14: Waveform verifying 1-bit ALU block

The 1-bit ALU integrates the AE block, LE block, CO logic block, XOR selector, and full adder to support all arithmetic and logical functions determined by the mode bit M and select lines S1S0. When M = 1, the arithmetic path is enabled through the AE and full adder; when M = 0, the LE block generates the logical output. The XOR2 gate selects between the arithmetic and logic paths to produce the final output F, while COUT and OV are generated from the arithmetic section.

The waveform confirms correct functionality for all eight operations. Intermediate AE and LE signals follow their intended Boolean behavior, the logic block selects the proper path, and the outputs F, COUT, and OV match the expected values for each control combination. Glitches were observed within the combinational paths; however, the outputs always settle to the correct final values. The 1-bit ALU output follows the truth table and the waveform verifies correct operation for every control code.

Table 3: 1-bit truth table for different ALU operations

M	S1	S0	A	B	X(LE)	Y(AE)	C0	F	OV
0	0	0	0	0	1	0	0	1	0
0	0	0	0	1	1	0	0	1	0
0	0	0	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	0	1	1	1	1	0	0	1	0
0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0
0	1	0	1	0	1	0	0	1	0
0	1	0	1	1	1	0	0	1	0
0	1	1	0	0	0	0	0	0	0
0	1	1	0	1	1	0	0	1	0
0	1	1	1	0	1	0	0	1	0
0	1	1	1	1	1	0	0	1	0
1	0	0	0	0	0	1	0	1	0
1	0	0	0	1	0	1	0	1	0
1	0	0	1	0	1	1	0	0	0
1	0	0	1	1	1	1	0	0	0
1	0	1	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	1	0
1	0	1	1	0	1	1	0	0	1
1	0	1	1	1	1	0	0	0	0
1	1	0	0	0	0	1	1	1	0
1	1	0	0	1	0	0	1	0	0
1	1	0	1	0	1	0	1	1	0
1	1	0	1	1	1	1	1	0	0
1	1	1	0	0	0	1	1	1	0
1	1	1	0	1	0	1	1	1	0
1	1	1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1	1	0

1-BIT ALU WITH REGISTERS

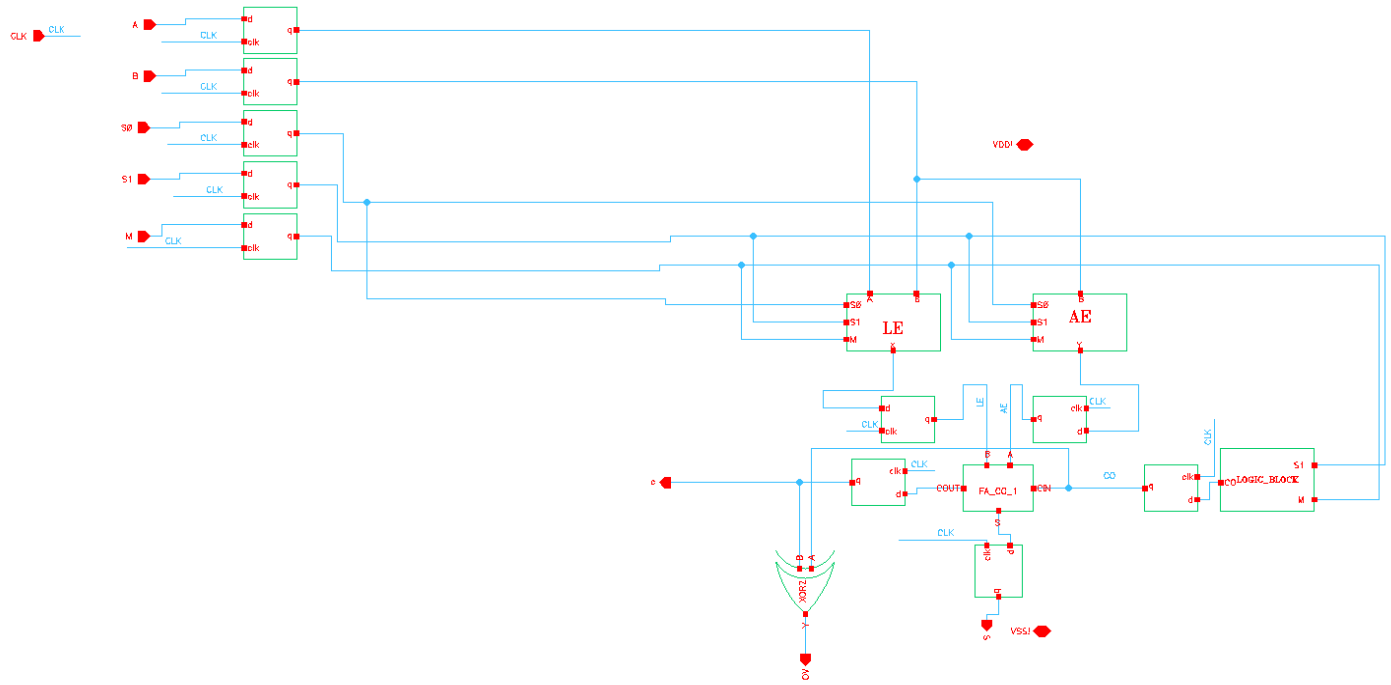


Figure 15: schematic of 1-bit ALU block with 1-bit registers

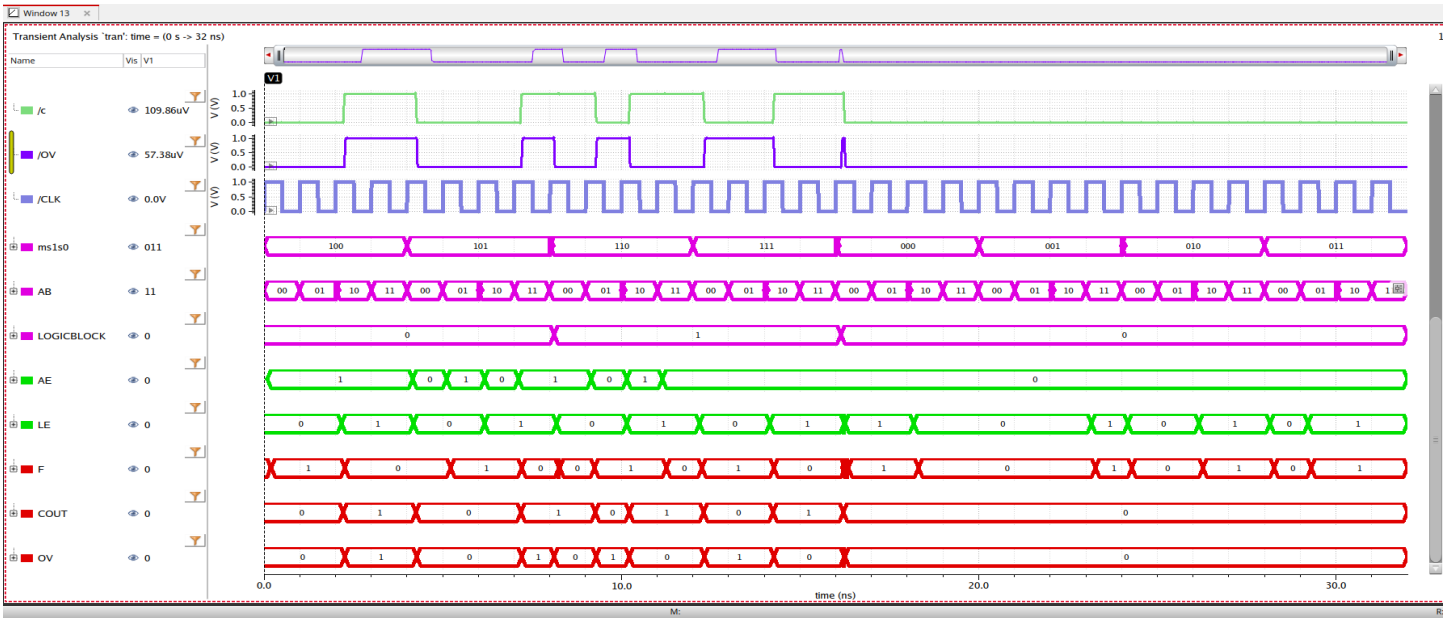


Figure 16: Waveform verifying 1-bit ALU block with 1-bit registers

The 1-bit ALU with registers is built by placing D-flip-flops at the inputs of the ALU (A, B, S0, S1, and M), ensuring that all control and data signals are sampled only on the rising edge of the clock before entering the combinational logic. Because each input is now stored and synchronized, the ALU receives stable values throughout the clock cycle, allowing it to perform the same logical and arithmetic functions verified earlier—addition, subtraction, increment, decrement, AND, OR, complement, and identity—without any mid-cycle transitions. The waveform confirms correct functionality: the registered inputs change only on clock edges, the LE (logic block) and AE (arithmetic block) outputs match the truth-table expectations, and the final outputs F, COUT, and OV are clean and time-aligned. Importantly, compared to the pure combinational 1-bit ALU, this registered version shows significantly fewer glitches because the flip-flops eliminate hazards caused by propagation delays inside the logic network. As a result, the ALU behaves more predictably, produces stable outputs, and is suitable for integration into larger synchronous systems like the full 8-bit ALU.

8-BIT ALU WITHOUT REGISTERS

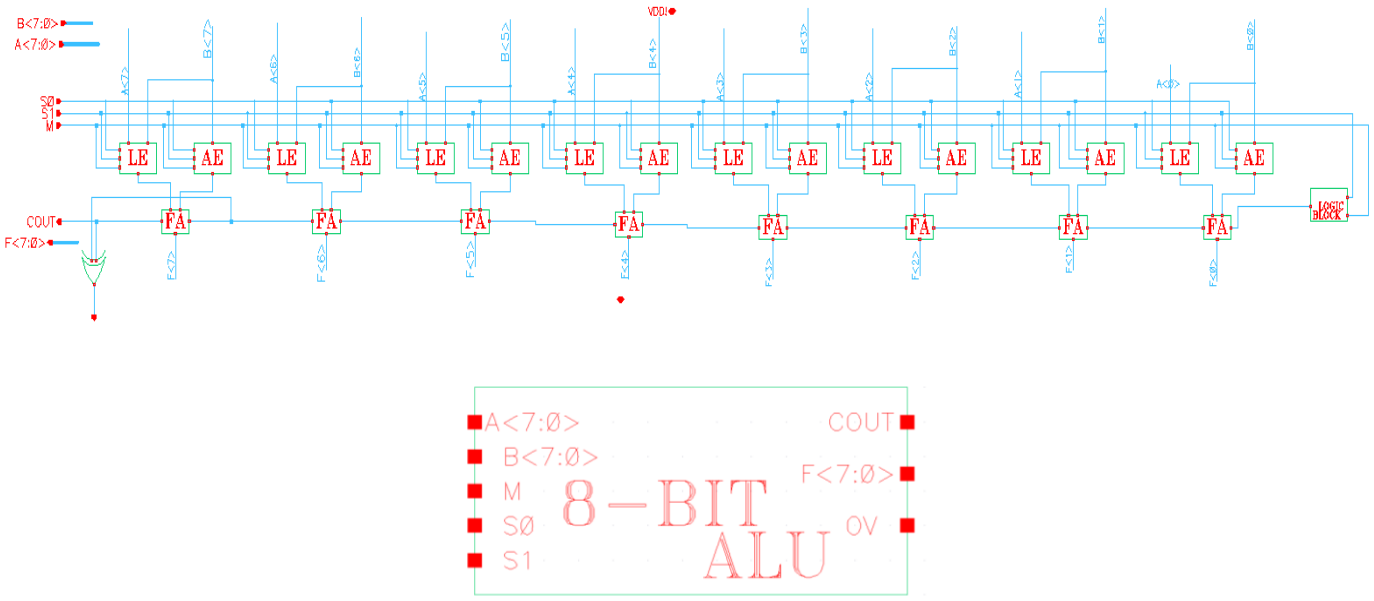


Figure 17: schematic ,symbol of 8-bit ALU block without registers

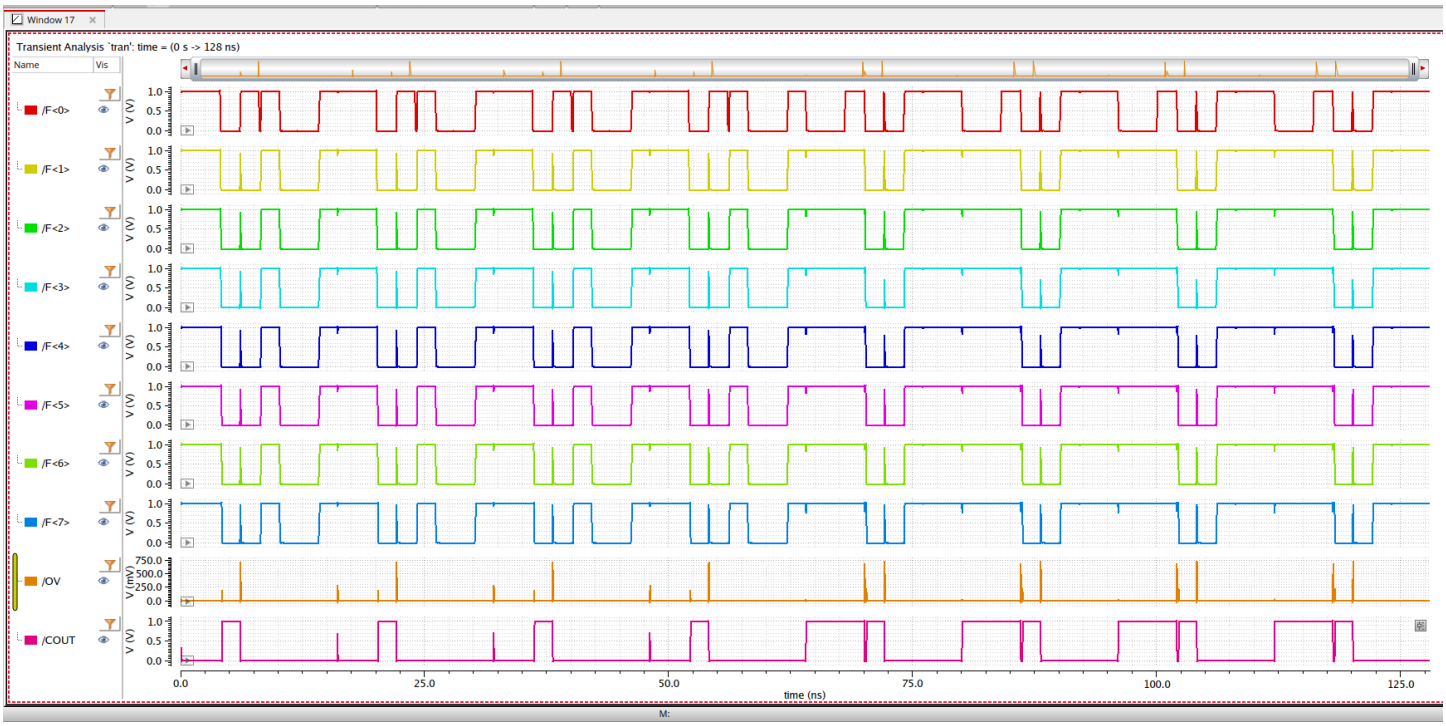
The 8-bit ALU is implemented as a bit-slice architecture, where eight identical 1-bit ALU slices are connected in series to form the full-width datapath. Each slice contains its own AE block, LE block, and full adder, allowing it to independently perform arithmetic and logical operations based on the shared control signals M and S1S0. The A and B operand buses are distributed horizontally to every slice, while the ripple-carry chain propagates from the LSB full adder (bit 0) to the MSB full adder (bit 7), enabling multi-bit addition, subtraction, increment, and decrement operations.

Because both arithmetic and logic paths exist inside each slice, the ALU can compute all eight required functions in parallel, and the correct result is selected internally. The outputs from all eight slices are then combined into the final 8-bit result F[7:0], while the COUT from the MSB slice represents the final carry-out of the entire unit. This modular, repeated structure simplifies design, ensures consistency between bits, and enables straightforward scaling from 1-bit to 8-bit width. The waveform demonstrates that the 8-bit ALU responds correctly to all eight control combinations of S1S0 and the mode bit M. As the control inputs change over time, the output bus F[7:0] updates according to the expected arithmetic or logic operation. The input buses A[7:0] and B[7:0] are varied through a sequence of patterns, and the resulting outputs match the functional behavior of the AE, LE, and full-adder chain.COUT transitions are visible when arithmetic operations overflow beyond bit-7, while OV toggles appropriately for signed overflow events. Minor output glitches appear when internal signals propagate through the ripple-carry chain, but each output bit always settles to the mathematically correct final value after the brief transient. Overall, the waveform verifies that the 8-bit ALU implements all required arithmetic and logical operations correctly.

Table 4: Functionality Verification Table

M	S1	S0	Function	A (Input)	B (Input)	Expected F[7:0]	C0	OV
1	0	0	Decrement	00000000	00000000	11111111	0	0
1	0	1	Add	00000000	11111111	11111111	0	0
1	1	0	Subtract	00000000	00000000	00000000	1	0
1	1	1	Increment	00000000	11111111	00000001	0	0
0	0	0	Complement	00000000	00000000	11111111	0	0
0	0	1	AND	00000000	11111111	00000000	0	0
0	1	0	Identity	00000000	00000000	00000000	0	0
0	1	1	OR	00000000	11111111	11111111	0	0

GLITCH OUTPUT:



The 8-bit ALU waveform shows clear glitching across multiple output bits because the design is fully combinational and implemented using a ripple-carry structure. When the inputs or control signals change, the AE, LE, and full-adder stages update at different times due to unequal propagation delays. As the carry ripples from the least significant bit toward the most significant bit, intermediate nodes temporarily assume incorrect values, producing short spikes and irregular transitions on F0–F7. These momentary toggles are expected in an unregistered ALU and do not indicate malfunction—once all signals finish propagating through the eight logic stages, the outputs consistently settle to the correct final values.

Figure 18-19: waveforms of 8-bit ALU block without registers

PDA CALCULATION:

Table 5-7: Area calculation Table

Block	Instances	Internal Gates	Count per Block	Total Count
AEBLOCK	8	AND3	2	16
		OR2	1	8
		INV	3	24
LEBLOCK	8	AND2	8	64
		AND3	1	8
		INV	5	40
		OR2	2	16
		OR3	1	8
Full Adder (CGFA1)	8	XOR2	2	16
		NAND2	3	24
Output XOR (final select)	1	XOR2	1	1
C0BLOCK	1	AND2	1	1

Gate Type	Total Count
INV	24 + 40 = **64**
AND2	64 + 1 = **65**
AND3	16 + 8 = **24**
OR2	8 + 16 = **24**
OR3	8
XOR2	16 + 1 = **17**
NAND2	24
Total Gates in 8-bit ALU	**226 gates**

Gate Type	Widths (PMOS/NMOS)	Length (µm)	# of Tran-sistors	Area per Gate (µm²)	Total Area for All Gates (µm²)
INV	PMOS = 0.10 µm NMOS = 0.10 µm	0.05	2	**0.01**	**0.64** *(64 gates × 0.010)*
AND2	PMOS = 0.10 µm NMOS = 0.10 µm	0.05	6	**0.03**	**1.95** *(65 gates × 0.030)*
OR2	PMOS = 0.10 µm NMOS = 0.10 µm	0.05	6	**0.03**	**0.72** *(24 gates × 0.030)*
NAND2	PMOS = 0.10 µm NMOS = 0.10 µm	0.05	4	**0.02**	**0.48** *(24 gates × 0.020)*
XOR2	PMOS = 0.10 µm NMOS = 0.10 µm	0.05	12	**0.06**	**1.02** *(17 gates × 0.060)*
AND3	PMOS: 0.19, 0.19, 0.19, 0.175 µm NMOS: 0.21, 0.21, 0.21, 0.10 µm	0.05	8	**0.07375**	**1.77** *(24 gates × 0.07375)*
OR3	PMOS: 1.37, 1.37, 1.37 µm NMOS: 0.09, 0.09, 0.09, 0.09 µm	0.05	7	**0.2235**	**1.788** *(8 gates × 0.2235)*

$A_{total} = 0.64 + 9.95 + 0.72 + 0.48 + 1.02 + 1.77 + 1.788$

$A_{total} = 8.388\ \mu m^2$

$Power = 57.73\mu W = 57.73 \times 10^{-6}\ W.$

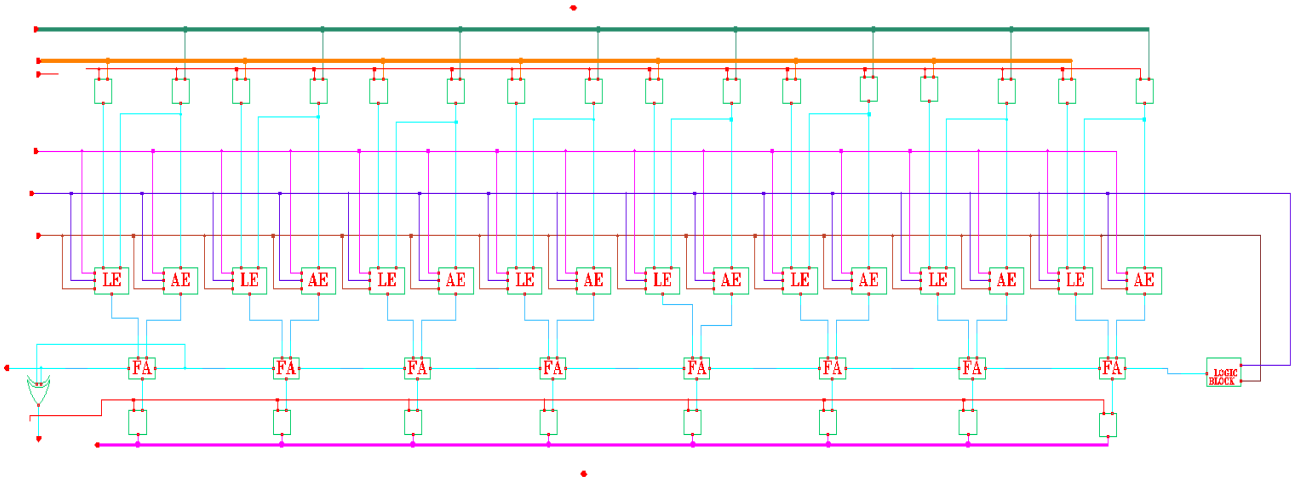
$Delay = \frac{229.494 + 98.196}{2} = 163.843 \times 10^{-12}\ s$

$Area = 8.388 \times 10^{-6}\ m^2$

PDA → Power * delay * Area

$= 7.94 \times 10^{-14}\ (W \cdot s \cdot \mu m^2)$

8-BIT ALU WITH REGISTERS



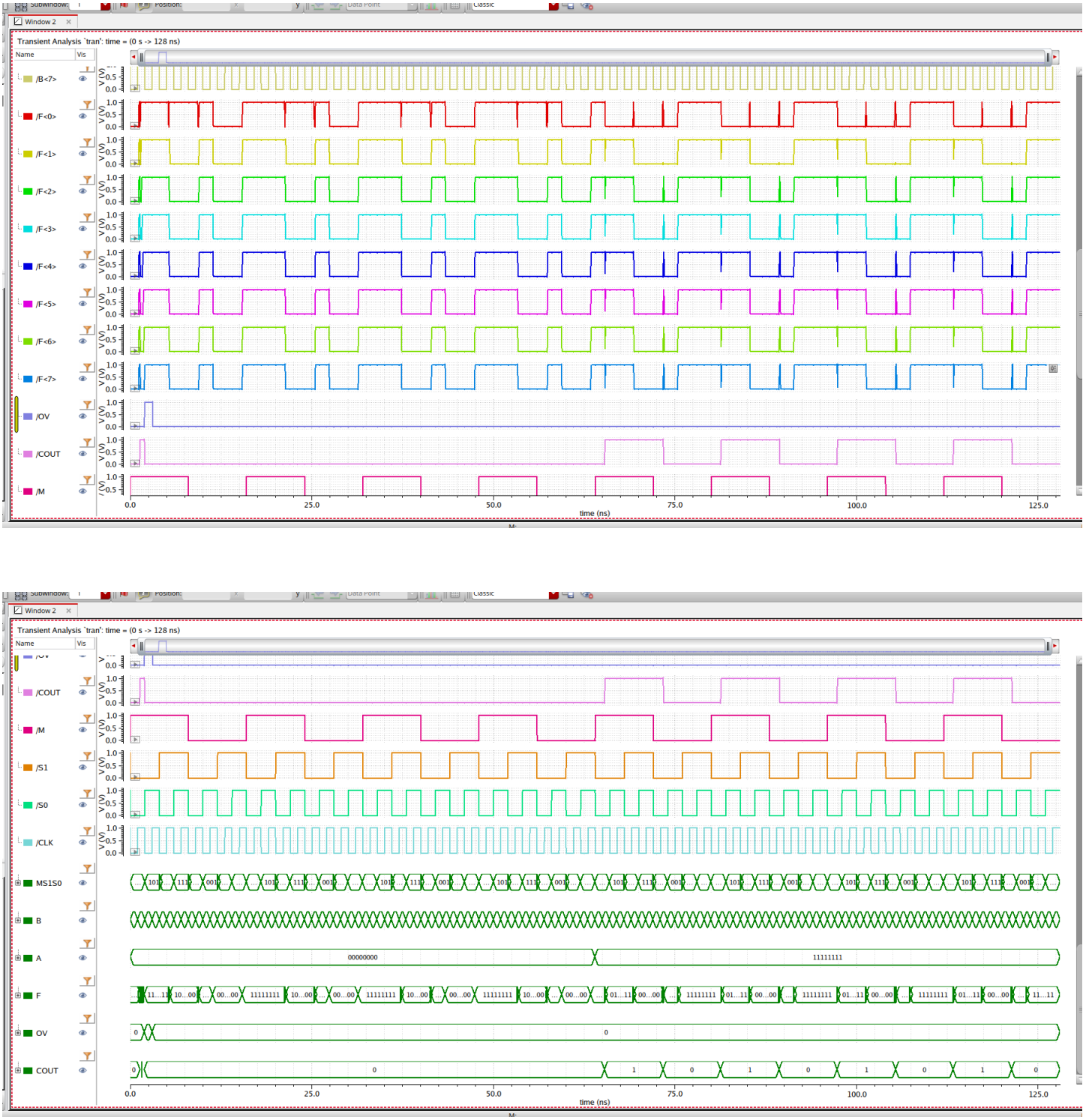


Figure 20-22: schematic, waveforms of 8-bit ALU block with registers

The 8-bit ALU with registers is built by cascading eight copies of the 1-bit ALU slice, each consisting of the Logic Element (LE), Arithmetic Element (AE), a full adder stage, and local carry logic. In this top-level architecture, every input control signal (M, S1, S0) and data input (A<7:0>, B<7:0>) is first passed through a bank of 1-bit D-flip-flops, ensuring that all operands and control lines are fully synchronized to the global clock before entering the ALU core. This addition of registers removes asynchronous hazards and ensures stable inputs at each clock cycle.

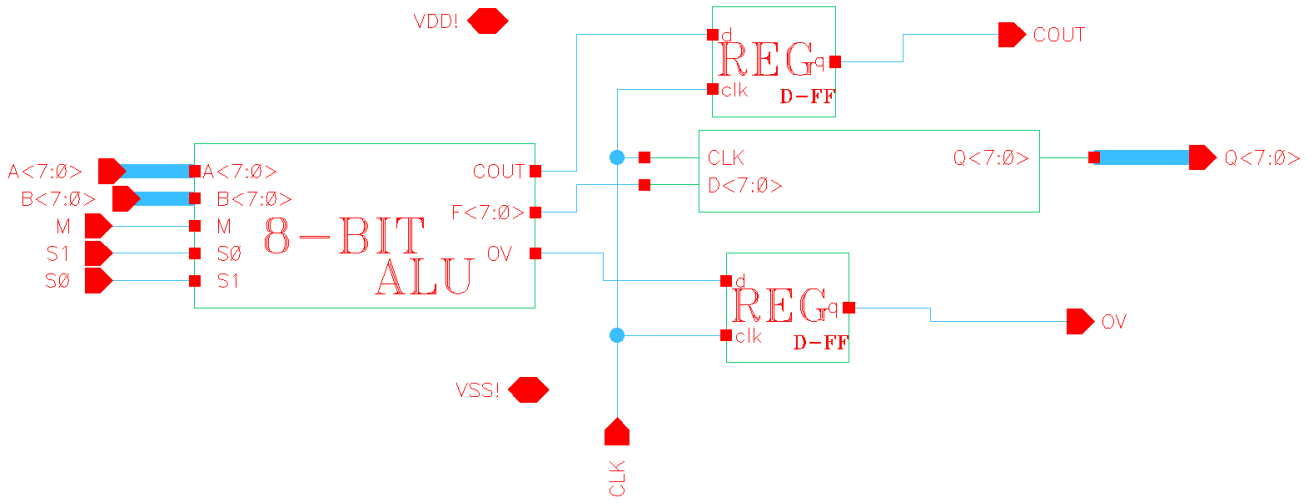
Functionally, the registered 8-bit ALU performs the exact same operations as the verified 1-bit ALU slice—Add, Subtract, Increment, Decrement, AND, OR, NOT, and Identity—but now extended to operate on full 8-bit words. Because the internal signals no longer change instantaneously when the inputs toggle (they wait for the next clock edge), the waveform exhibits significantly fewer glitches compared to the purely combinational ALU. All intermediate transitions inside AE/LE logic are effectively filtered out by the input registers.

The output waveform confirms correct operation:

- A<7:0> and B<7:0> change continuously, but the ALU output F<7:0> updates only on clock edges, demonstrating synchronous operation.
- The control word MS1S0 cycles through all eight functions, and F<7:0> reflects the correct 8-bit result for each mode.
- Carry-out and overflow signals also show clean transitions, updating only when arithmetic operations require them.
- There is no bit-to-bit misalignment; all eight output bits switch together, showing that the registered ALU eliminates ripple-delay skew typical in a large combinational structure.

Overall, the 8-bit registered ALU provides stable, clocked computation suitable for sequential digital systems. By incorporating D-flip-flop registers at both inputs and outputs, the ALU becomes more robust, avoids glitches, and aligns perfectly with CPU-style pipelined or multicycle designs. This makes the registered ALU a reliable building block for datapaths, processors, counters, or any system requiring synchronized multi-bit arithmetic and logic operations.

8-BIT ALU WITH REGISTERS AT OUTPUT



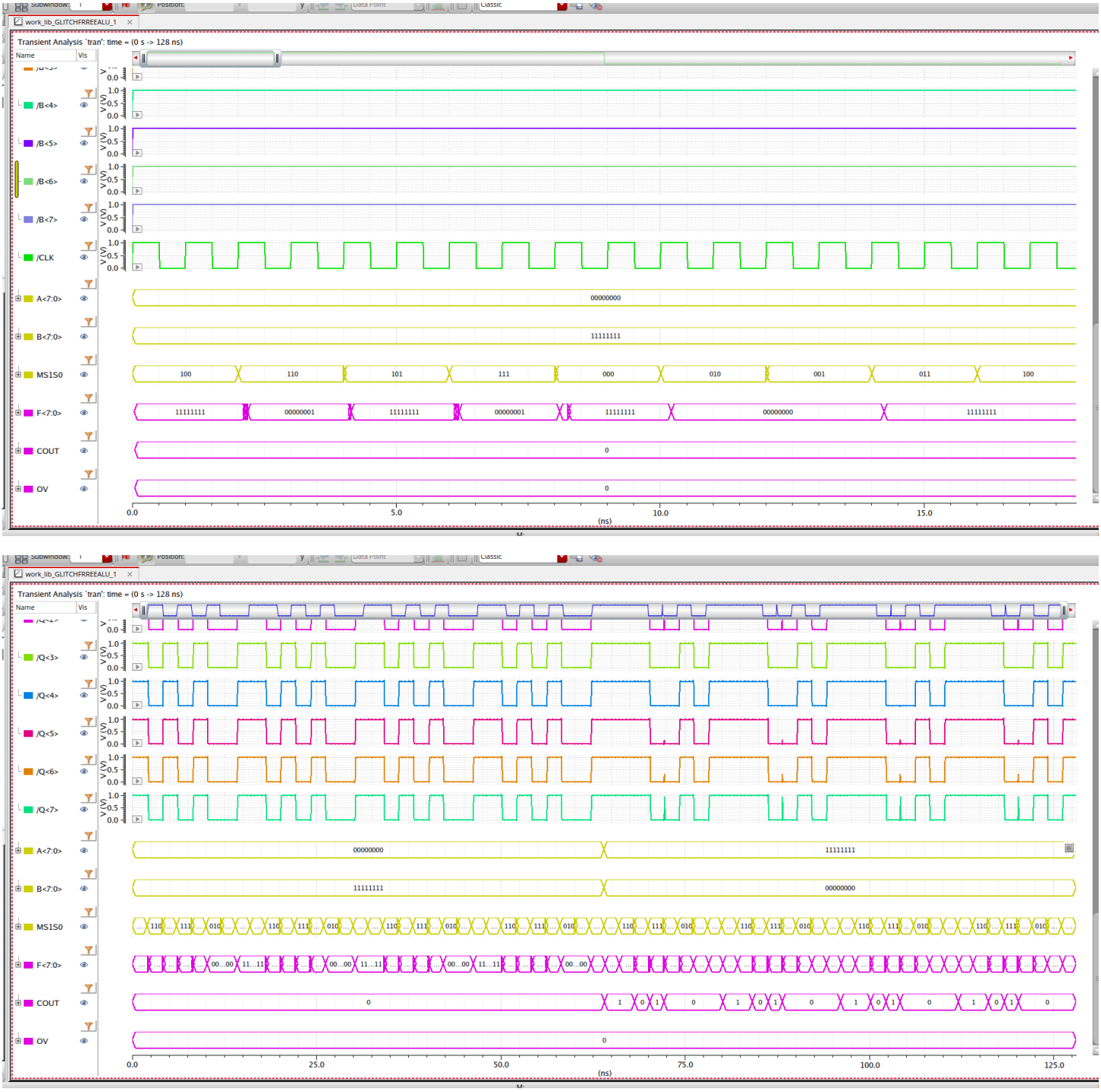


Figure 23-25: schematic, waveforms of 8-bit ALU block with registers at output

The 8-bit ALU with output registers operates exactly like the original combinational 8-bit ALU, performing the same arithmetic and logic functions, but with an important improvement: the results are passed through synchronous D-flip-flop registers before appearing at the final outputs. The internal ALU still generates temporary switching activity due to ripple-carry and logic-level propagation, but these transitions do not directly appear at the final outputs. Instead, the registers capture the result only on the rising edge of the clock, which means the external signals Q<7:0>, COUT, and OV change only once per clock cycle. From the waveform, it is clear that although the raw ALU outputs toggle internally, the registered outputs show significantly fewer glitches compared to the pure combinational 8-bit ALU, because the flip-flops hold a stable value between clock edges.

8-BIT ALU WITH REGISTERS AT INPUTS AND OUTPUTS

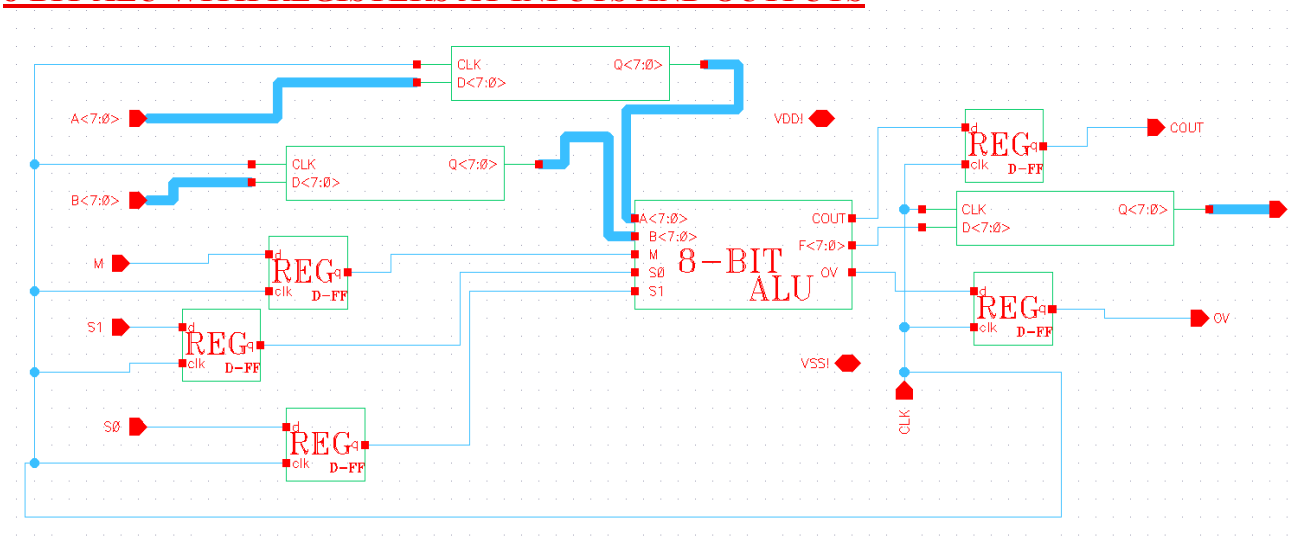
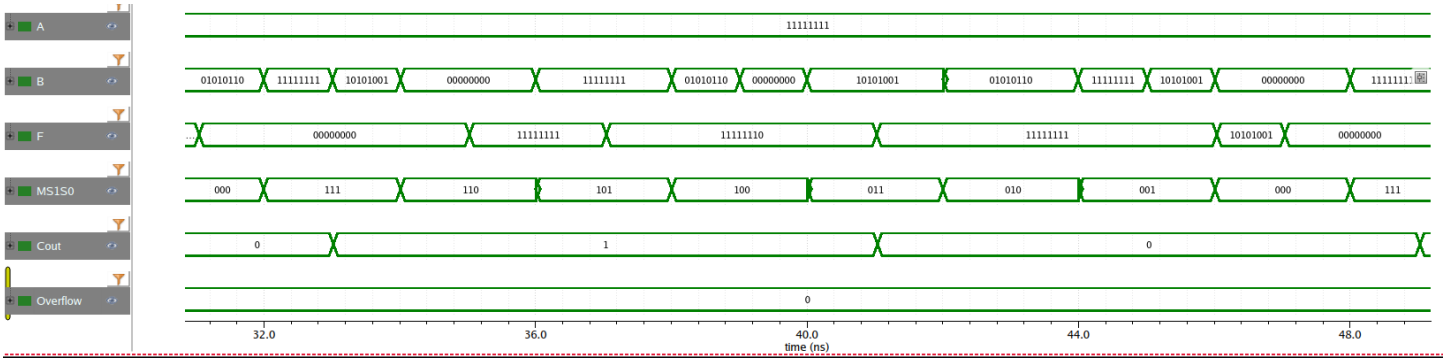
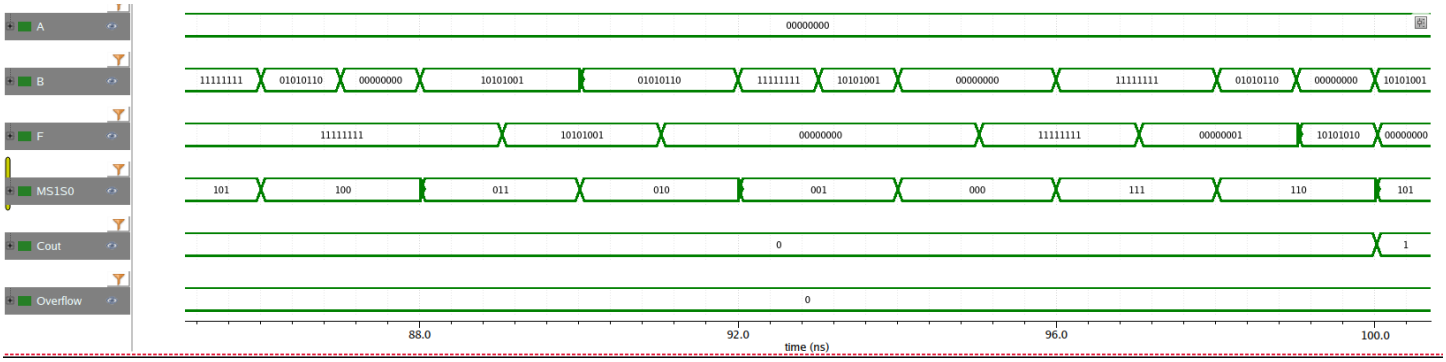


Figure 26: schematic of 8-bit ALU block with registers at input, output

The 8-bit ALU with registers at both its inputs and outputs demonstrates significantly improved temporal stability compared to the purely combinational ALU. By placing D-flip-flop registers on A<7:0>, B<7:0>, M, S1, and S0 before the ALU, and additional registers on F<7:0>, COUT, and OV at the output, all signals entering and leaving the ALU become fully synchronized with the clock. Because the ALU receives stable, edge-aligned inputs and its results are captured only on the clock edge, internal switching settles before the next stage is updated. As a result, the waveform shows much no glitches compared to the unregistered 8-bit ALU, while still performing all eight operations exactly as defined. The registered design ensures that even when inputs toggle rapidly or change simultaneously, the output updates cleanly on the next rising clock edge, confirming correct functionality across all M, S1, S0 combinations. This architecture provides predictable timing, improved signal integrity, and makes the ALU suitable for use in larger synchronous systems.

FUNCTIONALITY VERIFICATION (Different testcases):





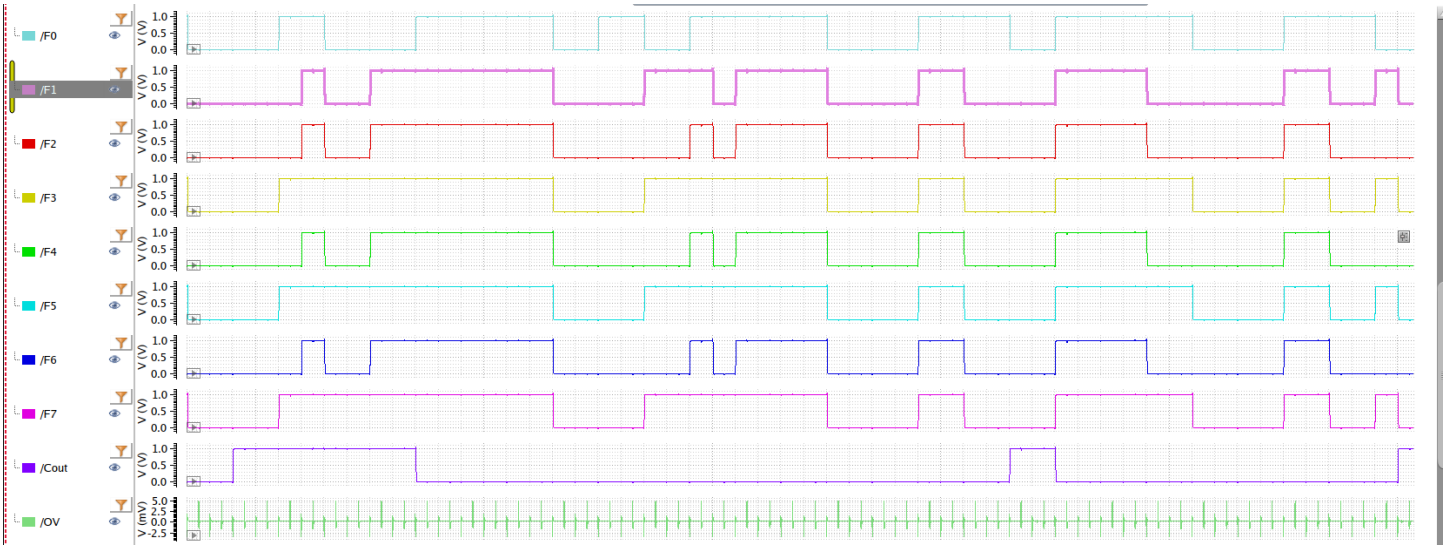
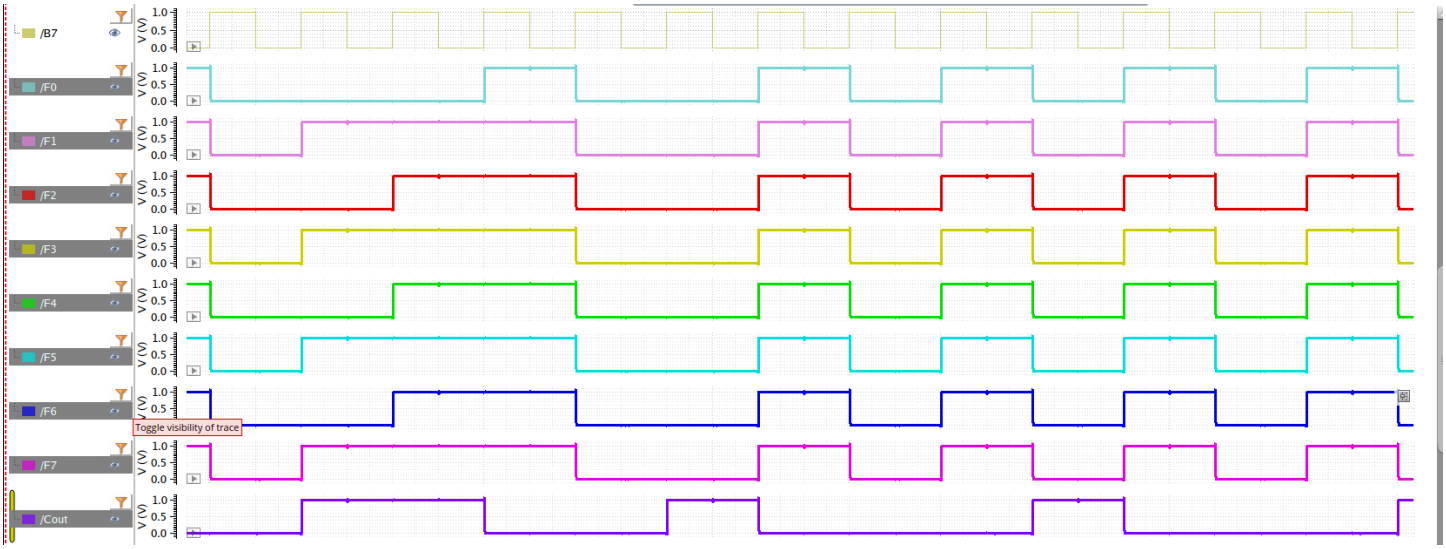
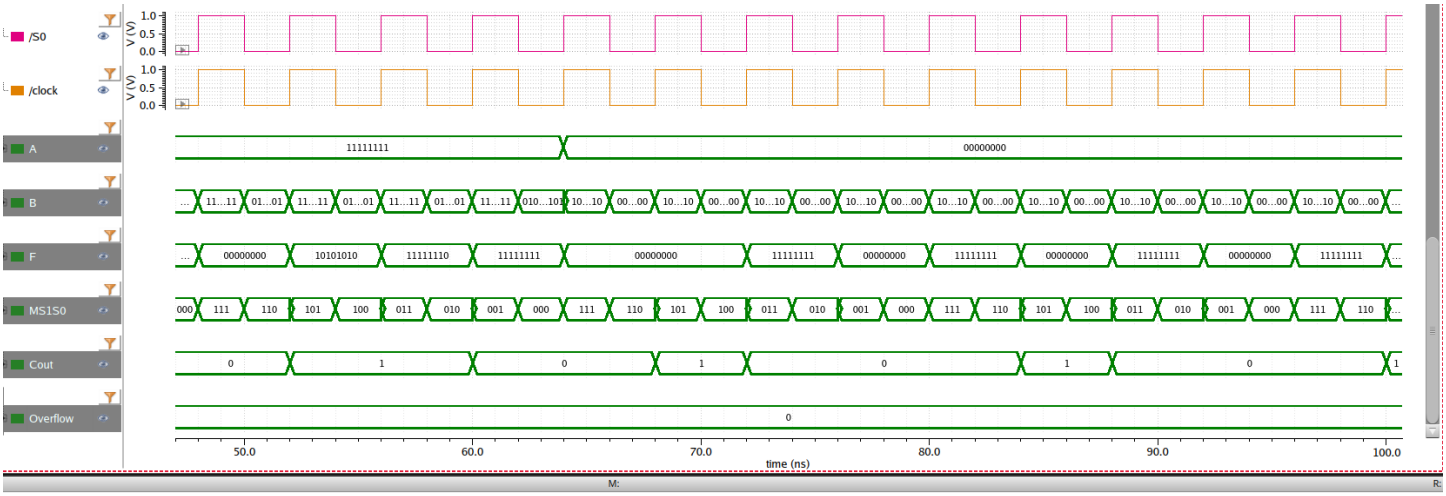


Figure 27: Functionality waveforms of 8-bit ALU block with registers at input, output (case-1)

FUNCTIONALITY VERIFICATION:





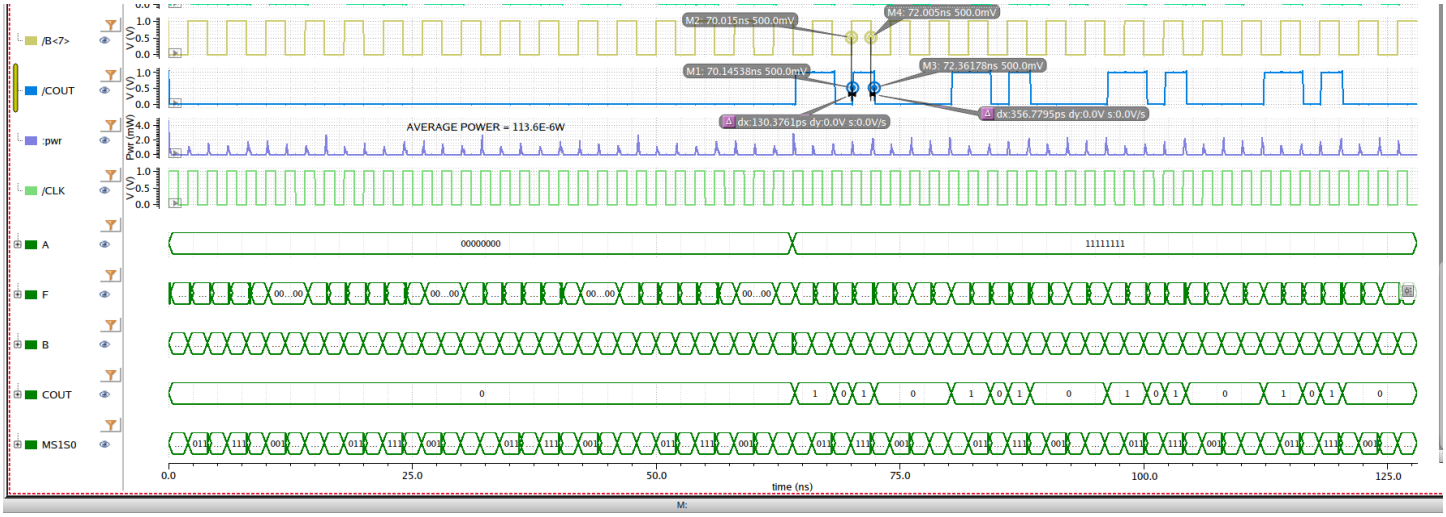


Figure 28: Functionality waveforms of 8-bit ALU block with registers at input, output (case-2)

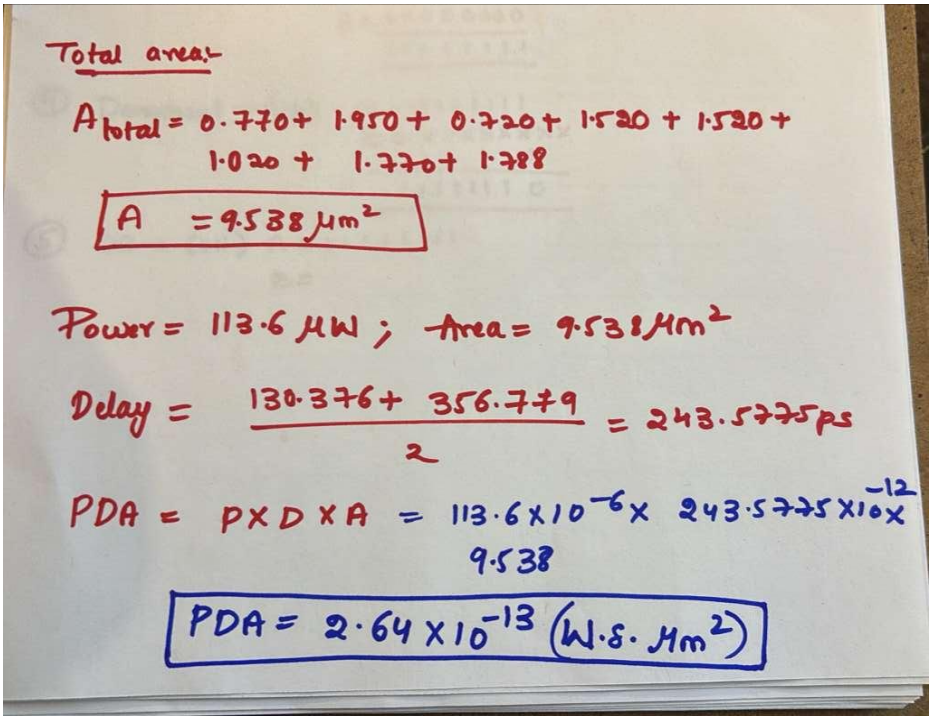
PDA CALCULATION:

Table 8-10: Area calculation Table

Block	Instances	Internal Gates	Count per Block	Total Count
AEBLOCK	8	AND3	2	16
		OR2	1	8
		INV	3	24
LEBLOCK	8	AND2	8	64
		AND3	1	8
		INV	5	40
		OR2	2	16
		OR3	1	8
Full Adder (CGFA1)	8	XOR2	2	16
		NAND2	3	24
Final XOR Selector	1	XOR2	1	1
C0BLOCK	1	AND2	1	1
Register Block (DFF)	13	NAND2	4	52
		INV	1	13

Gate Type	Total Count (Break-down)
INV	24 + 40 + 13 = **77**
AND2	64 + 1 = **65**
AND3	16 + 8 = **24**
OR2	8 + 16 = **24**
OR3	**8**
XOR2	16 + 1 = **17**
NAND2	24 + 52 = **76**

Gate Type	Widths (PMOS / NMOS)	Length (µm)	# of Transistors	Area per Gate (µm²)	Total Area (µm²)
INV	PMOS = 0.10, NMOS = 0.10	0.05	2	0.01	77 × 0.010 = **0.770**
AND2	3 PMOS = 0.10 3 NMOS = 0.10	0.05	6	0.03	65 × 0.030 = **1.950**
OR2	3 PMOS = 0.10 3 NMOS = 0.10	0.05	6	0.03	24 × 0.030 = **0.720**
NAND2	2 PMOS = 0.10 2 give like this for 8 bit alu with registers \NMOS = 0.10	0.05	4	0.02	76 × 0.020 = **1.520**
XOR2	6 PMOS = 0.10 6 NMOS = 0.10	0.05	12	0.06	17 × 0.060 = **1.020**
AND3	PMOS: 0.19, 0.19, 0.19, 0.175 NMOS: 0.21, 0.21, 0.21, 0.10	0.05	8	0.07375	24 × 0.07375 = **1.770**
OR3	PMOS: 1.37, 1.37, 1.37 NMOS: 0.09, 0.09, 0.09, 0.09	0.05	7	0.2235	8 × 0.2235 = **1.788**



Activity Factor = 2, highest clock frequency = 1 GHz

3. RESULT

The ALU simulations verified correct logical and arithmetic functionality for all MS1S0 combinations, and both the 1-bit and 8-bit architectures produced accurate outputs. The unregistered 8-bit ALU showed expected ripple-carry glitches, but all bits settled to the correct values. Once registers were added at the inputs and outputs, the ALU became fully synchronous and hazard-free, with clean transitions occurring only on clock edges. Functionally, both versions were correct, but the registered ALU provided significantly better timing stability and predictable behavior.

The PDA analysis highlighted the design tradeoff between the two architectures. The unregistered ALU contained 226 total gates, giving it the smaller area and lower total transistor count. However, because it relied entirely on ripple-carry logic, it exhibited higher worst-case delay, which increased the overall PDA value. In contrast, the registered ALU added 13 DFF blocks, increasing area (additional INV and NAND2 gates) and slightly increasing power due to clocked elements, but it achieved lower effective delay because the synchronous registers allowed timing to be measured per clock cycle rather than through a long combinational path. As a result, the registered ALU delivered better cycle-to-cycle performance despite higher area and power overhead, while the unregistered ALU remained more compact but slower in worst-case timing.

Numerically, the unregistered ALU produced a PDA of

$$7.94 \times 10^{-14}\ (W \cdot s \cdot \mu m^2),$$

while the registered ALU resulted in a higher PDA of

$$2.64 \times 10^{-13}\ (W \cdot s \cdot \mu m^2)$$

due to its increased power and area overhead.

4. CONCLUSION

The 8-bit ALU was successfully designed, implemented, and verified at the transistor level, demonstrating correct functionality for all arithmetic and logic operations across both the unregistered and registered architectures. The unregistered ALU achieved a lower area and power footprint but exhibited longer worst-case delay, resulting in a smaller PDA value of 7.94×10^{-14}. The registered ALU, while larger and more power-intensive, eliminated glitches and provided clean synchronous operation, though its PDA increased to 2.64×10^{-13}. These results confirmed the expected tradeoff between performance stability and hardware cost. Overall, the project provided valuable experience in hierarchical design, synchronous timing behavior, and power–delay–area optimization in transistor-level VLSI systems.