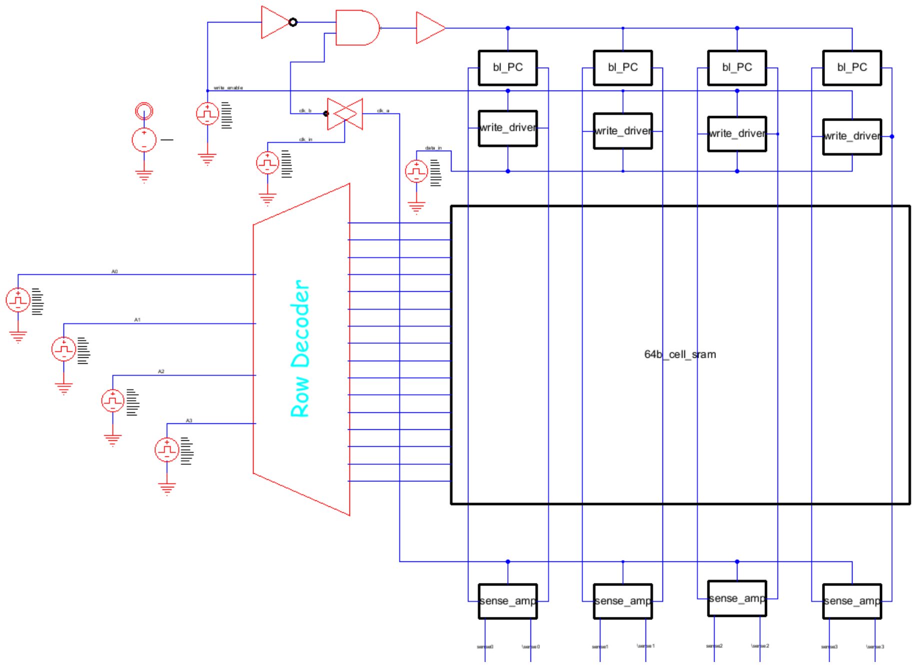


ESE 3700: Project 2

Krishna Karthikeya Chemudupati
Morgan Wang

University of Pennsylvania
Professor Jane Li

May 3, 2025

Contents

1 Introduction

In this project, we design, simulate, and evaluate a custom 16x4 SRAM array using 22nm HP CMOS technology. We also designed peripheral circuits for wordline decoding, bitline precharging, data writing, and differential sense amplification. The objective of the project is to validate both functional correctness and performance characteristics of the SRAM, including timing behavior and power consumption, to ultimately compute a figure of merit that captures its efficiency.

$$FOM = 60 \cdot Bitcell\ Area \cdot Power \cdot Delay^2$$

The first major task involves circuit-level design of all constituent blocks in Electric, followed by SPICE-level netlist generation and behavioral validation via Ngspice simulations. Once basic functionality is verified, the timing characteristics of the design are evaluated by measuring the worst-case read and write access delays across all cells in the array. These measurements inform the minimum safe clock period required for reliable operation. Using this period, the power consumption is measured under repeated full-array write operations, alternating between writing all 0s and all 1s. The average current draw over these cycles, in conjunction with the supply voltage, yields the dynamic power consumption of the array. The final step is to compute a figure of merit defined by the product of the minimum access time and the energy consumed per operation.

Memory Operation Summary

The 16×4 SRAM operates in two primary modes: read and write. Each of the 16 wordlines corresponds to a distinct row in the memory array, and each row contains 4 SRAM cells corresponding to 4 bitlines and their complements. Address inputs $A_3A_2A_1A_0$ select a unique wordline through the row decoder.

Write Operation: During a write, the `WE` signal is high. A pulse is applied to the `clk_b` line, and a NAND2 gate combining `WE` and `clk_b` disables the bitline precharge transistors. The `data_in` signal is driven onto both BL and $\overline{\text{BL}}$ using a pair of tristate buffers and inverters. The wordline corresponding to the decoded address would have been selected, turning on the access transistors of the selected row. The resulting voltage difference between BL and $\overline{\text{BL}}$ flips the internal cross-coupled inverters of the target SRAM cells, storing the new value.

Read Operation: During a read, the `WE` signal is low. On the rising edge of `clk_b`, the bitlines are precharged to V_{DD}. On the next clock phase, a rising edge on `clk_a` enables the sense amplifiers. The selected wordline would be driven high by the row decoder, connecting the internal nodes of the selected SRAM cells to the bitlines. Depending on the stored data, a differential voltage develops between BL and $\overline{\text{BL}}$, which is detected and amplified by the sense amplifier to produce a full-swing logic level at the output.

This two-phase operation ensures correct timing between precharge, access, and sensing, enabling reliable read and write operations while minimizing contention and power consumption.

Timing Summary

To facilitate reliable access and avoid race conditions in read and write paths, the SRAM operates under a 2-phase non-overlapping clocking scheme. This scheme is derived from a single input clock (clk_{in}) and produces two non-overlapping clock phases, clk_a and clk_b. The read and write processes are each broken down into two sub-phases, governed by specific combinations of the write enable signal (`WE`) and the clock phases. We describe these four phases below.

Precharge phase:

$$\texttt{WE} = 0, \quad \texttt{clk\_b} = 1, \quad \texttt{clk\_a} = 0$$

In this configuration, the bitline precharge circuitry is activated, charging both the true and complementary bitlines to V_{DD}. No wordlines are active during this phase. The memory array remains in a neutral state, preparing for either a read or write operation.

Read phase:

$$\texttt{WE} = 0, \quad \texttt{clk\_b} = 0, \quad \texttt{clk\_a} = 1$$

The wordline is enabled while the precharge circuitry is disabled. The selected memory cell pulls one of the bitlines slightly lower than the other based on its stored data. This small differential is then amplified by the sense amplifier, which is also activated during this phase via the `clk_a` signal.

Write phase:

$$\texttt{WE} = 1, \quad \texttt{data in} = 1/0,$$

The write driver is active and drives the data value onto the bitlines. If the row decoder selects a row, then WL is high, allowing the data stored in that cell to be overwritten with the BL or $\overline{BL}$ value.

This carefully sequenced timing ensures that each operation occurs in a well-defined order without signal overlap, guaranteeing correct and robust memory access.

2 Schematics & Design Choices

2.1 Tier 0

Tier 0 consists of fundamental digital logic gates (Inverter, Buffer, NAND2, and NOR2) which serve as the foundation for all higher-level modules in our SRAM design. These gates were implemented using standard CMOS logic with minimum sizing to optimize area and switching speed. Minimum-sized transistors were both functional for this application and contributed minimal capacitive loading and decreased power consumption. These are reused throughout the hierarchy, including in the row decoder, clock generator, and control logic for the write driver and sense amplifier. Ensuring correct and efficient operation at this level was essential for timing closure and modular reliability in the full memory system.

2.1.1 Inverter

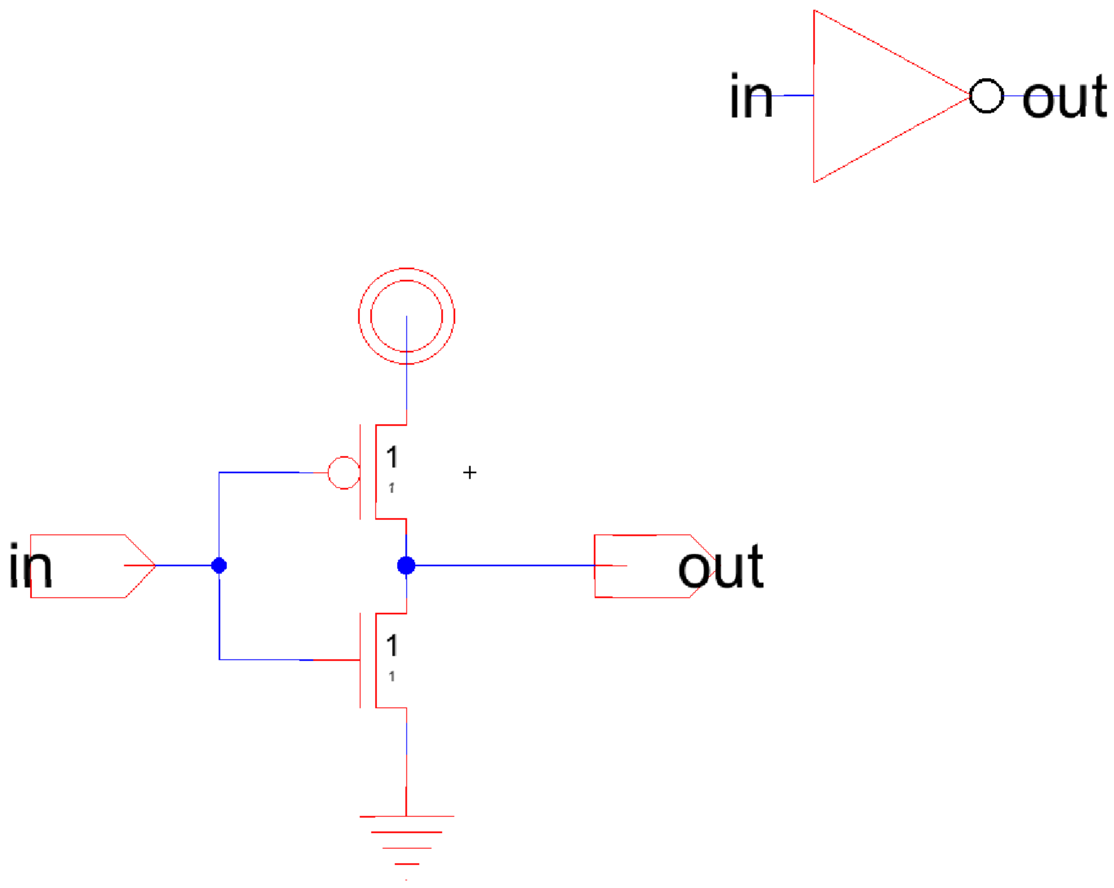
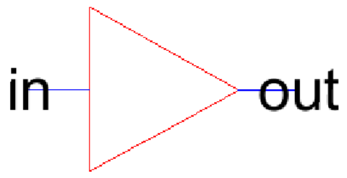


Figure 1: Inverter

2.1.2 Buffer



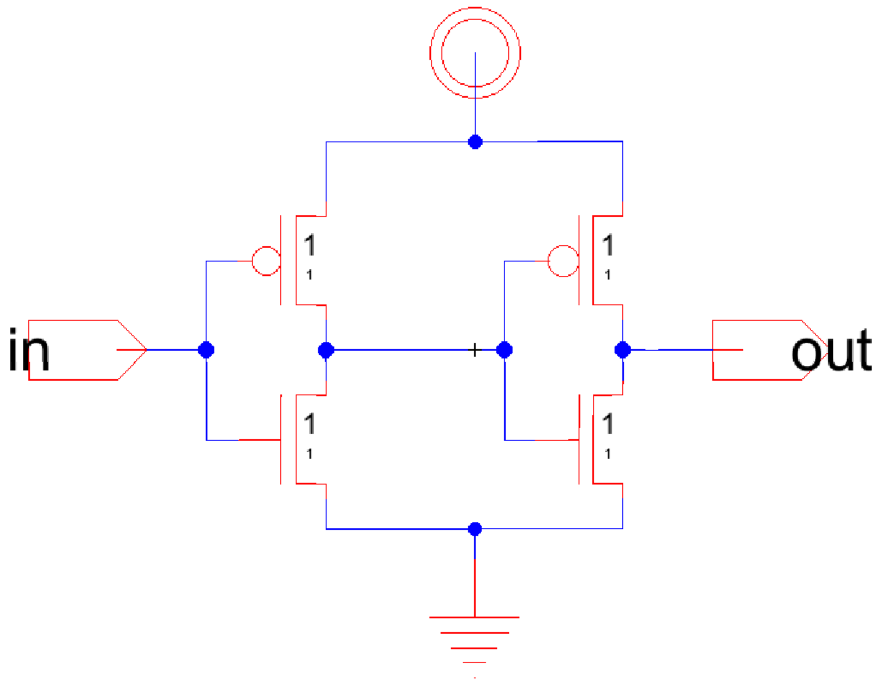


Figure 2: Buffer

2.1.3 NAND2

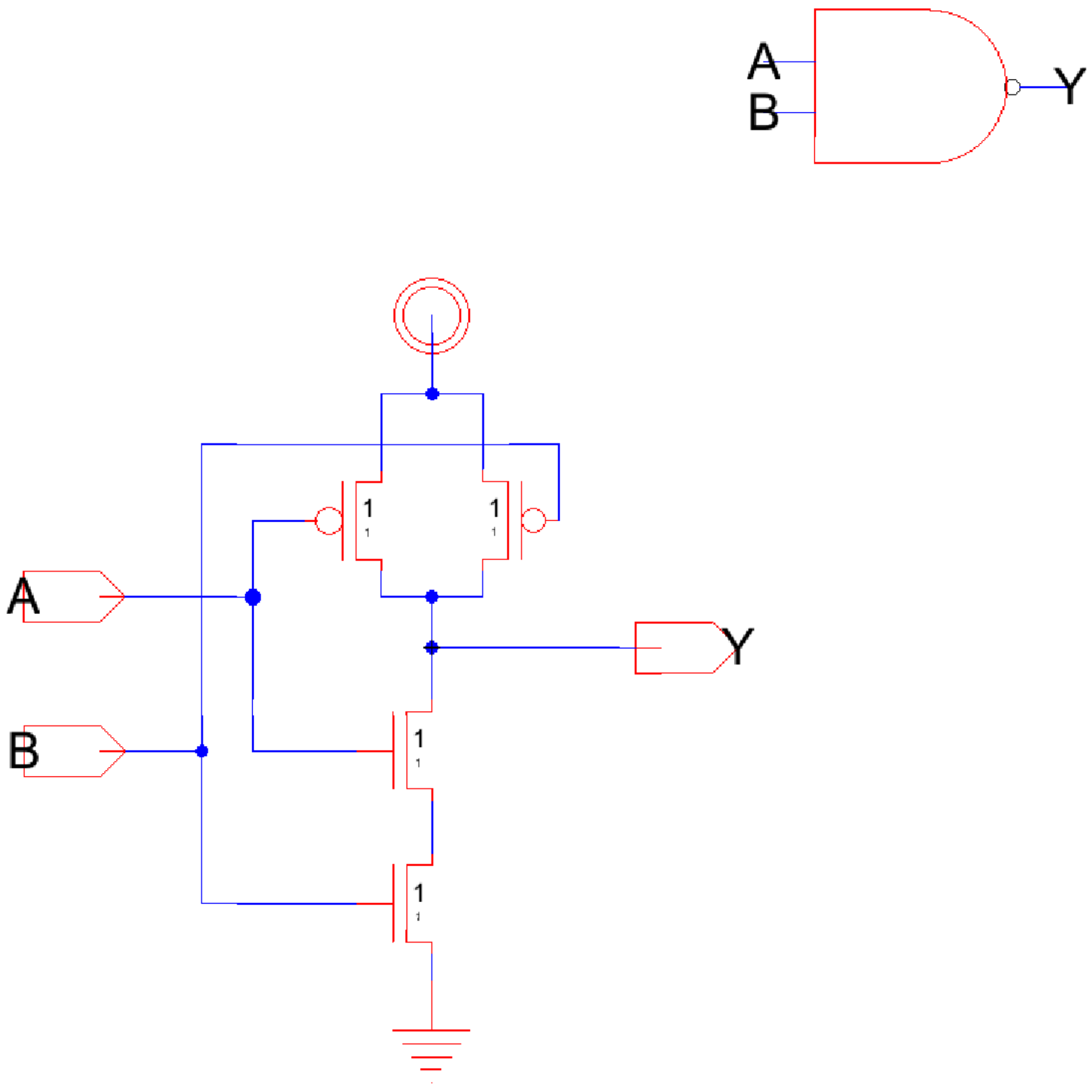
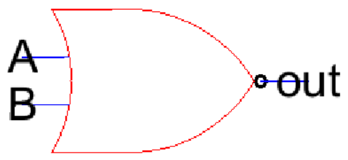


Figure 3: NAND2

2.1.4 NOR2



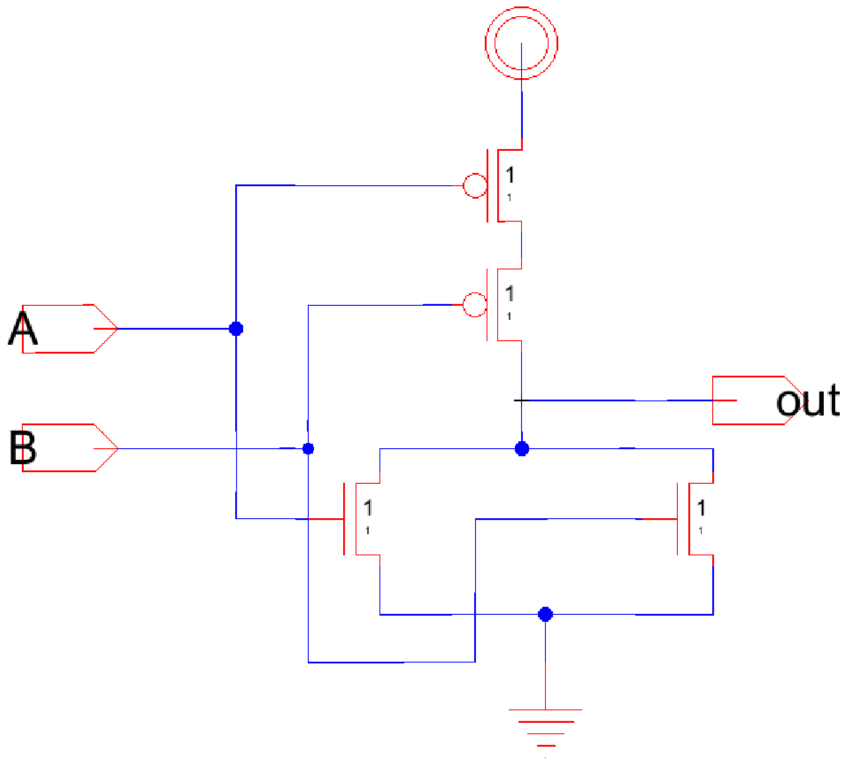


Figure 4: NOR2

2.2 Tier 1

Tier 1 consists of critical functional blocks that support SRAM control and data flow. These modules operate on top of basic logic gates and directly interact with the bitlines, wordlines, and clock domains. Each block was designed to fulfill a specific role in the memory read/write cycle and was simulated in isolation before system-level integration. Together, they form the operational backbone of the SRAM's periphery and internal timing.

2.2.1 Tristate Buffer

This buffer is used in the write driver to drive bitlines only when the write enable signal is active. When disabled, the output goes to high-impedance, preventing contention with other circuitry on the bitlines, such as the precharge circuit or SRAM cell access transistors.

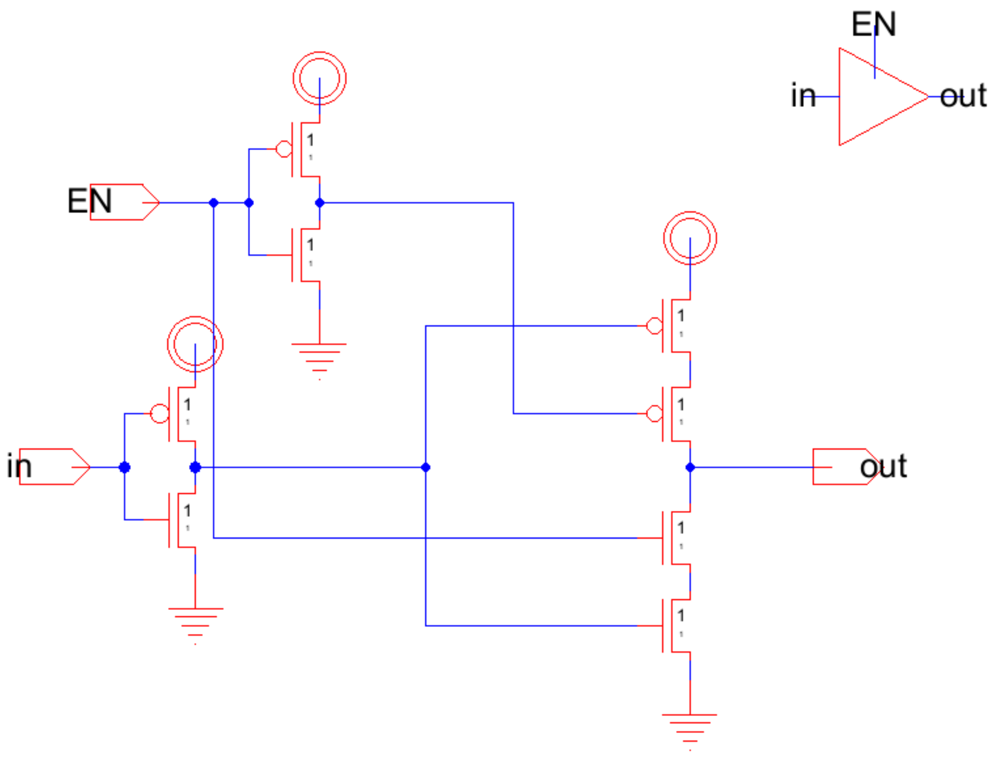


Figure 5: Tristate Buffer

2.2.2 Tristate Inverter

Paired with the tristate buffer, the tristate inverter allows us to simultaneously write both the true and complement of the data to BL and $\overline{BL}$, respectively. It also enters a high-impedance state during non-write phases to avoid interfering with reads and precharges.

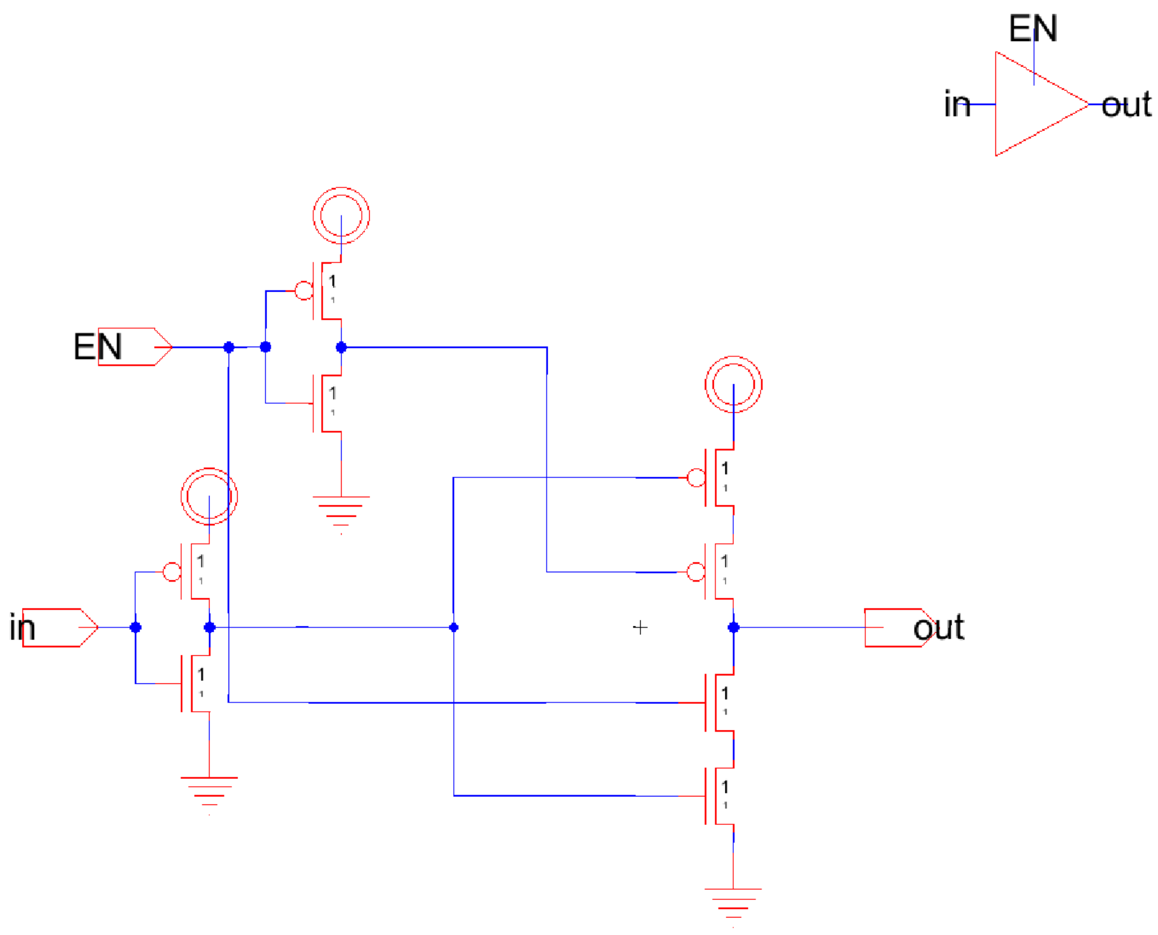


Figure 6: Tristate Inverter

2.2.3 6T SRAM Bitcell

The 6T cell is the core memory storage element, composed of two cross-coupled inverters and two access transistors. Each word stores one bit and interfaces with shared bitlines for read and write access.
Min. sized transistors were used because that maximized FOM – the marginal decrease in delay with larger transistors was outweighed by the increased power consumption and area.

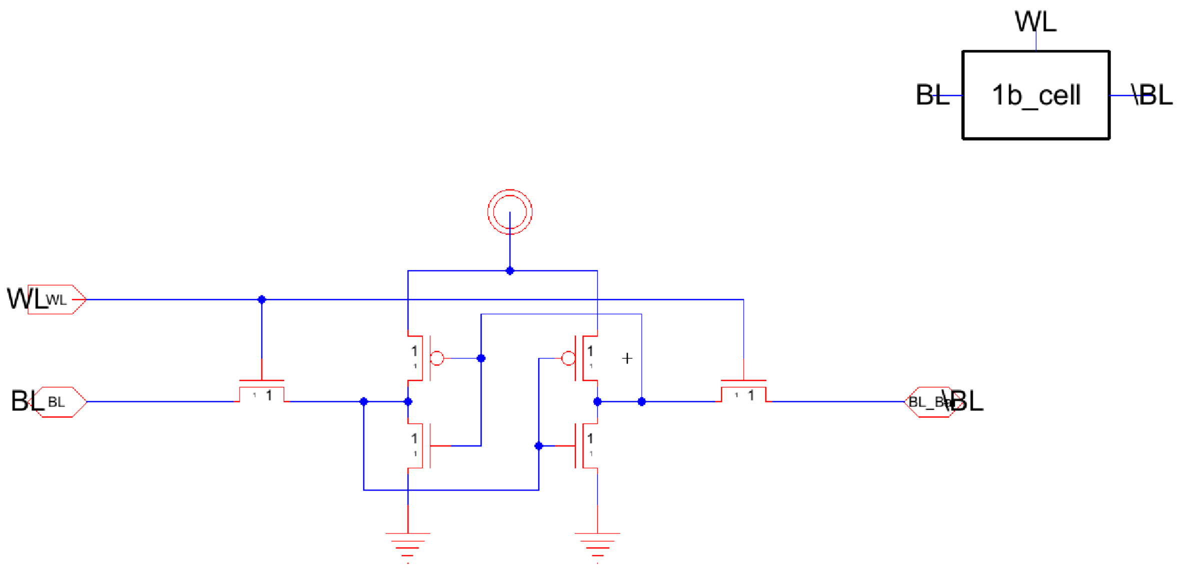


Figure 7: 1-bit 6T SRAM cell

2.2.4 Two-Phase Clock Generator

This module derives two non-overlapping clocks, `clk_a` and `clk_b`, from a single system clock (`clk_in`). The `clk_b` signal, in conjunction with the write enable signal (`WE`), controls the precharge circuitry to ensure the bitlines are properly initialized prior to each access. The `clk_a` signal is connected to the sense amplifiers and acts as the enable signal for differential bitline sensing during read operations. These non-overlapping clocks prevent contention between dynamic stages and guarantee safe and glitch-free bitline transitions.

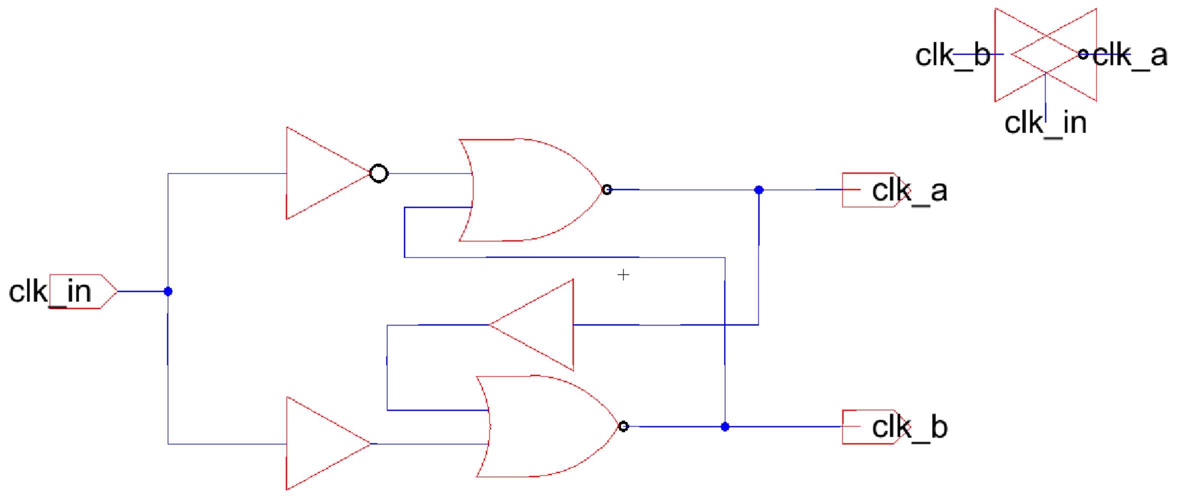


Figure 8: 2 Phase Clock Generator

2.2.5 Bitline Precharge Circuit

This circuit uses PMOS transistors to precharge both BL and $\overline{BL}$ to V_{DD} before a read or write cycle begins. An equalization PMOS between the bitlines ensures they are matched, minimizing differential skew. The circuit is enabled during the precharge phase where the write enable signal and clk_a are low, and clk_b is high.

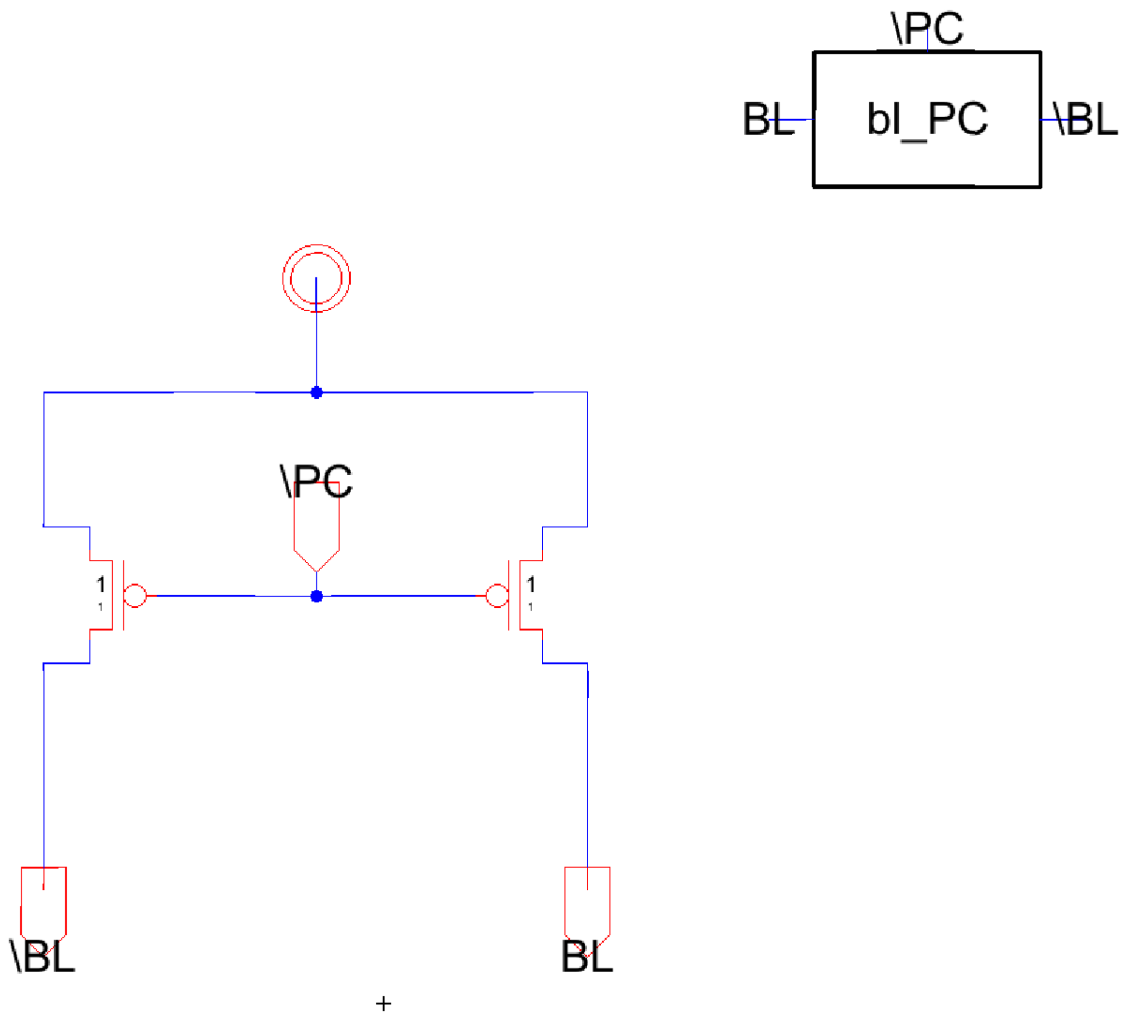


Figure 9: Bitline Precharge

2.2.6 Write Driver

The write driver forces the bitlines to a known logic level based on the input data and write enable signal. It includes tristate logic to avoid shorting or interfering with the precharge or sense amplifier during non-write operations. Data is differentially driven onto BL and $\overline{BL}$.

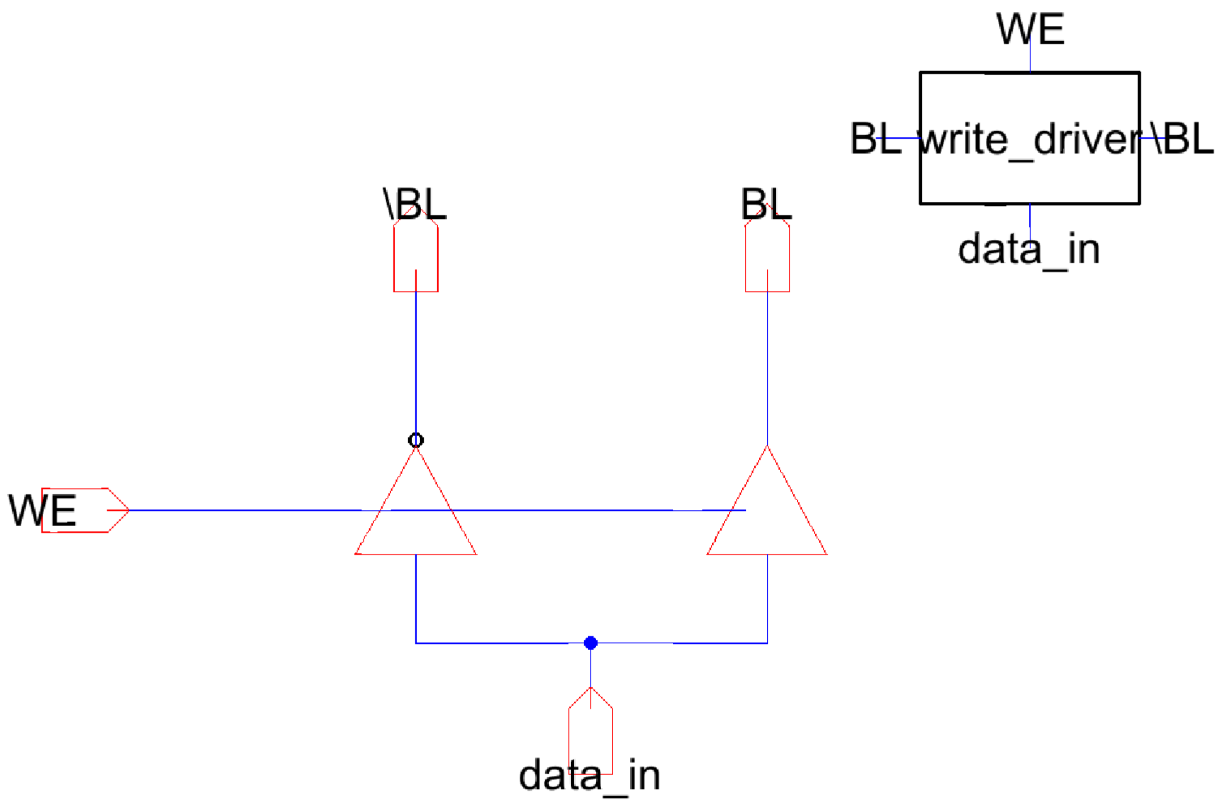


Figure 10: Write Driver

2.2.7 Digital Sense Amplifier

This clocked differential amplifier detects minute voltage differences between BL and $\overline{BL}$ during a read and amplifies them to full-swing digital logic levels. The sense amplifier only activates when $sense_{clk}$ is high, reducing power consumption and avoiding premature evaluation.

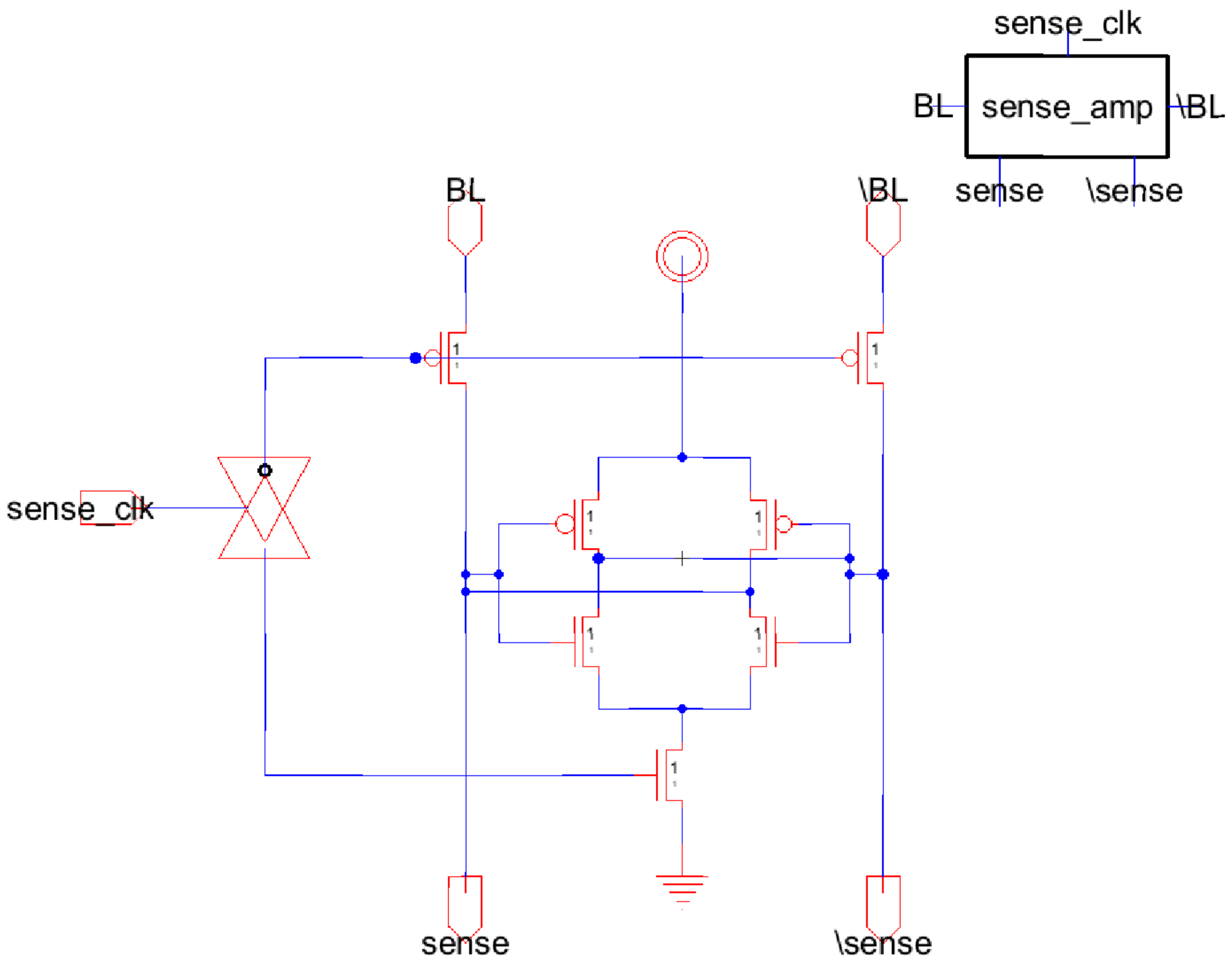


Figure 11: Digital Sense Amplifier

2.3 Tier 2

Tier 2 includes high-level integration blocks that combine the Tier 1 components into complete SRAM subsystems. These schematics bring together multiple bitcells, control elements, and address decoding logic to form the full 64-bit memory structure. This level of abstraction was used to simulate system behavior, evaluate delay across full address ranges, and validate end-to-end read/write correctness. The row decoder activates a single wordline based on a 4-bit input, while the column and full SRAM blocks demonstrate scalable architecture and data flow.

2.3.1 16-Row Column Slice

This schematic instantiates a single SRAM bit column with 16 vertically stacked 1-bit cells, each connected to a distinct wordline and sharing a common bitline pair (BL, $\overline{BL}$). Each cell represents one bit of a different word, enabling isolated examination of vertical access behavior. This configuration was used to simulate timing-critical delays such as worst-case wordline transitions and bitline discharge contention. It is also driven by the output of the row decoder and served as a critical setup for delay measurement during read operations.

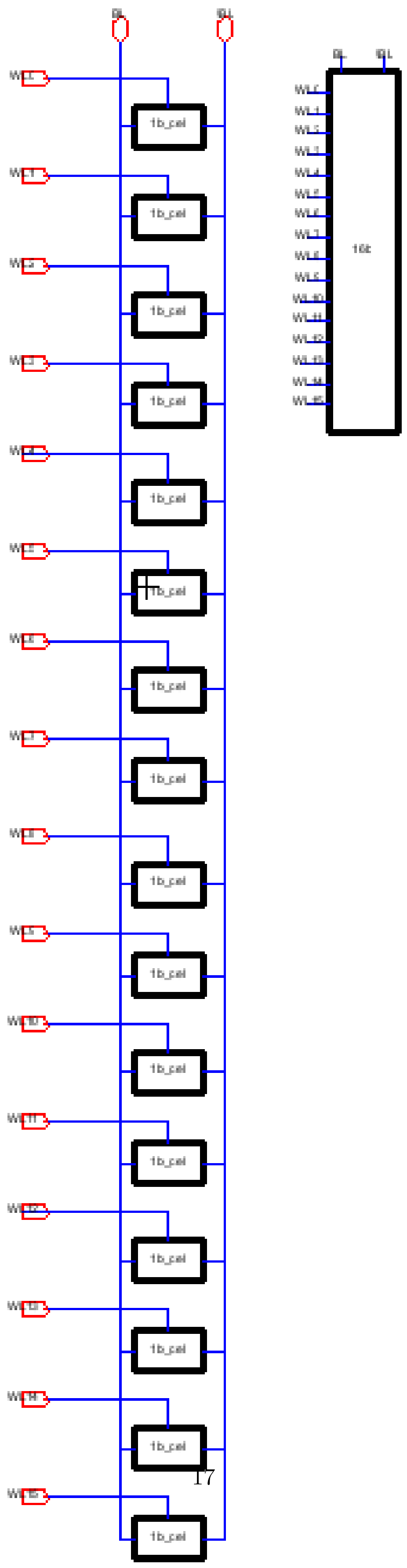


Figure 12: 16 bit SRAM Column Slice

2.3.2 64-bit SRAM Array

This schematic composes the complete 64-bit SRAM array by horizontally replicating four 16-row column slices. Each column stores one bit per word, and together they form a 4-bit word per row, distributed across 16 addressable rows. All columns share the same set of wordlines, which are selected via the row decoder. This block served as the top-level datapath in system-level simulation and was the target for both delay characterization and full read/write verification. The columnar architecture reflects standard SRAM design methodology, where word-level access is achieved through simultaneous activation of multiple independent bitlines.

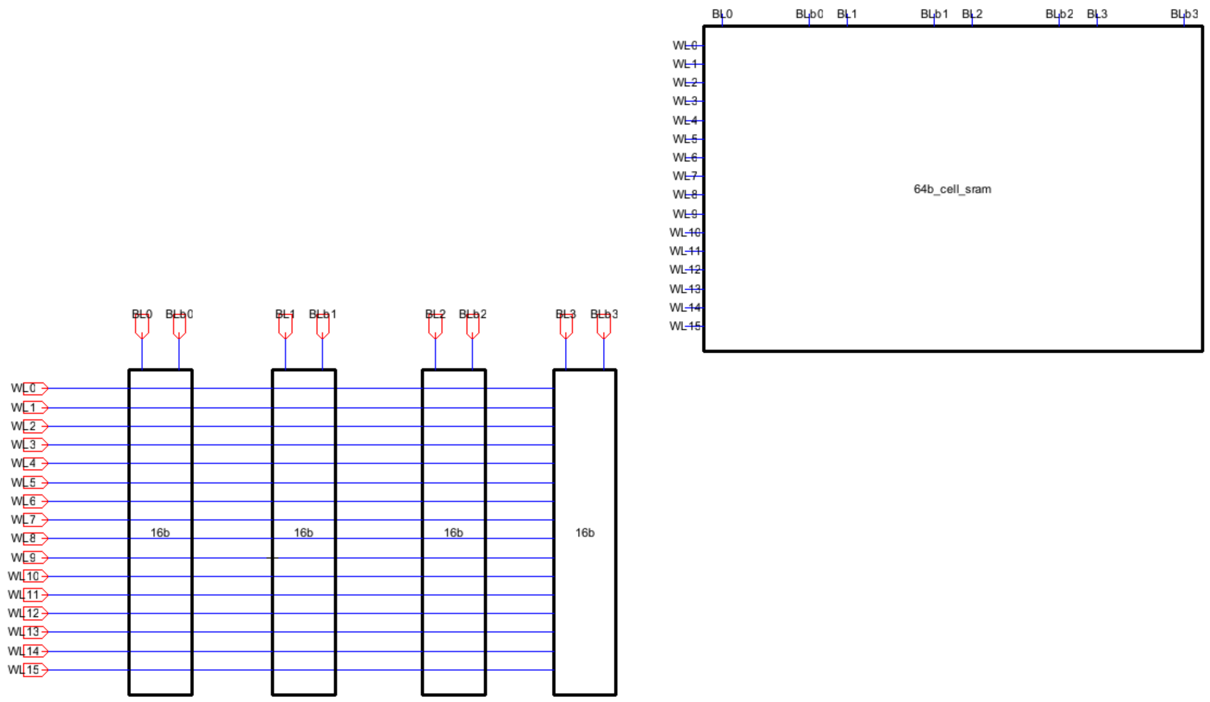


Figure 13: 64-bit SRAM Array

2.3.3 Row Decoder

To address the delay limitations of the simpler NAND4-based decoder analysed in class, we redesigned the row decoder using a two-level logic scheme. In the optimized design, the 4-bit address is first decomposed into 8 intermediate signals using NAND2 gates, each representing a specific pairwise combination of address lines and their complements. These are then passed to 16 NOR2 gates, each configured to detect a unique 4-bit input pattern. This structure eliminates the deep pull-down stack and distributes the logical effort across two shallower stages, significantly improving switching speed. The reduction in fan-in per gate improves gate drive and transition sharpness, leading to a lower critical path delay. This decoder was used in the final implementation for timing-sensitive simulations.

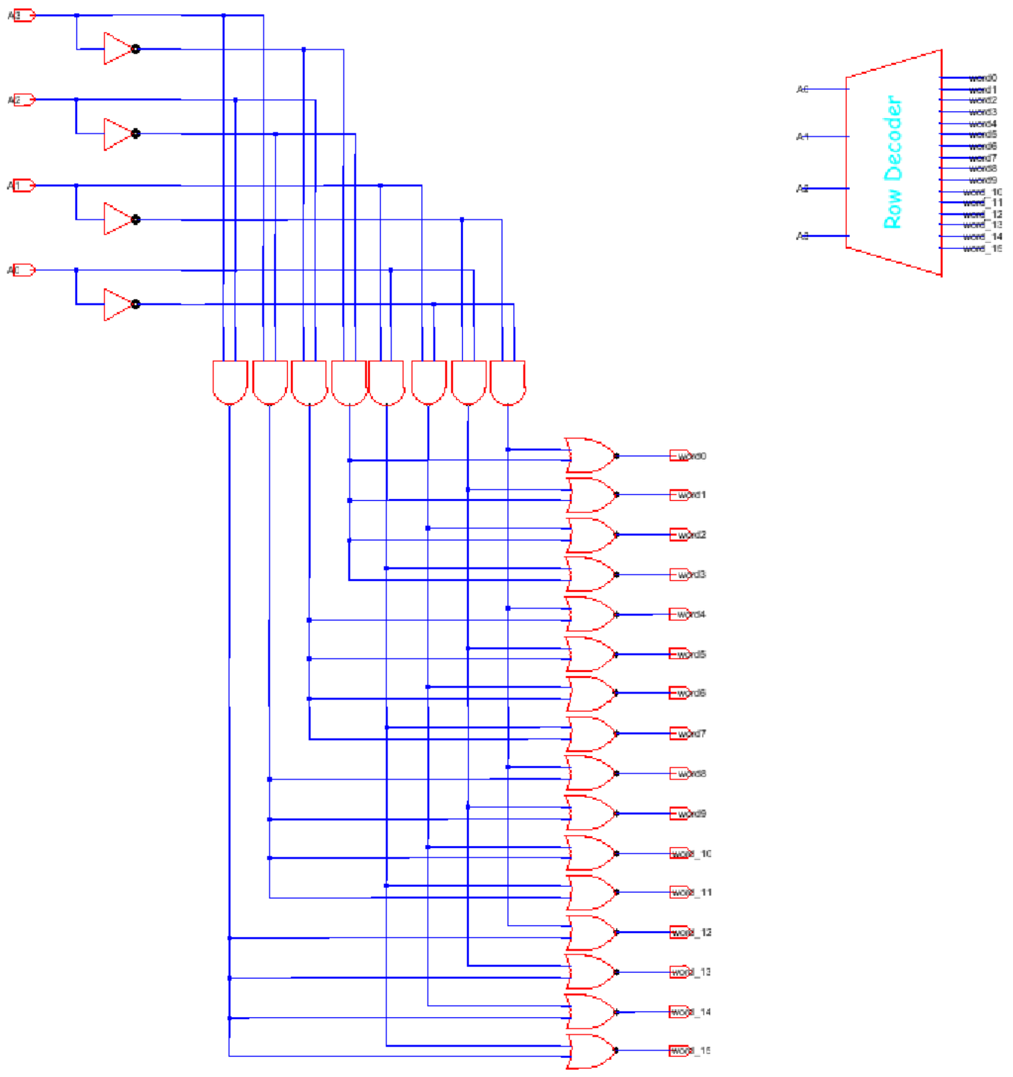


Figure 14: Row Decoder

3 Testing and Results

3.1 Row Decoder

Figure 15 illustrates the transient simulation results for the row decoder module. The inputs to the decoder (`A0` through `A3`) sweep through all 16 possible 4-bit binary combinations in sequence, beginning from `0000` up to `1111`. Each horizontal trace in the waveform corresponds to one of the 16 decoder output lines (`word0` through `word15`), and the simulation verifies that exactly one wordline is asserted (logic high) at any given input combination while all others remain at logic low.

The correctness of the decoder is evidenced by the one-hot nature of the output: for each unique 4-bit input address, a single corresponding wordline becomes active, while the remaining wordlines stay inactive. This strict one-hot behavior is essential for safely addressing a single row in an SRAM array without activating multiple rows simultaneously. Thus, this waveform confirms that the decoder accurately maps each input combination to its intended wordline, and demonstrates proper functionality of the row decoding logic.

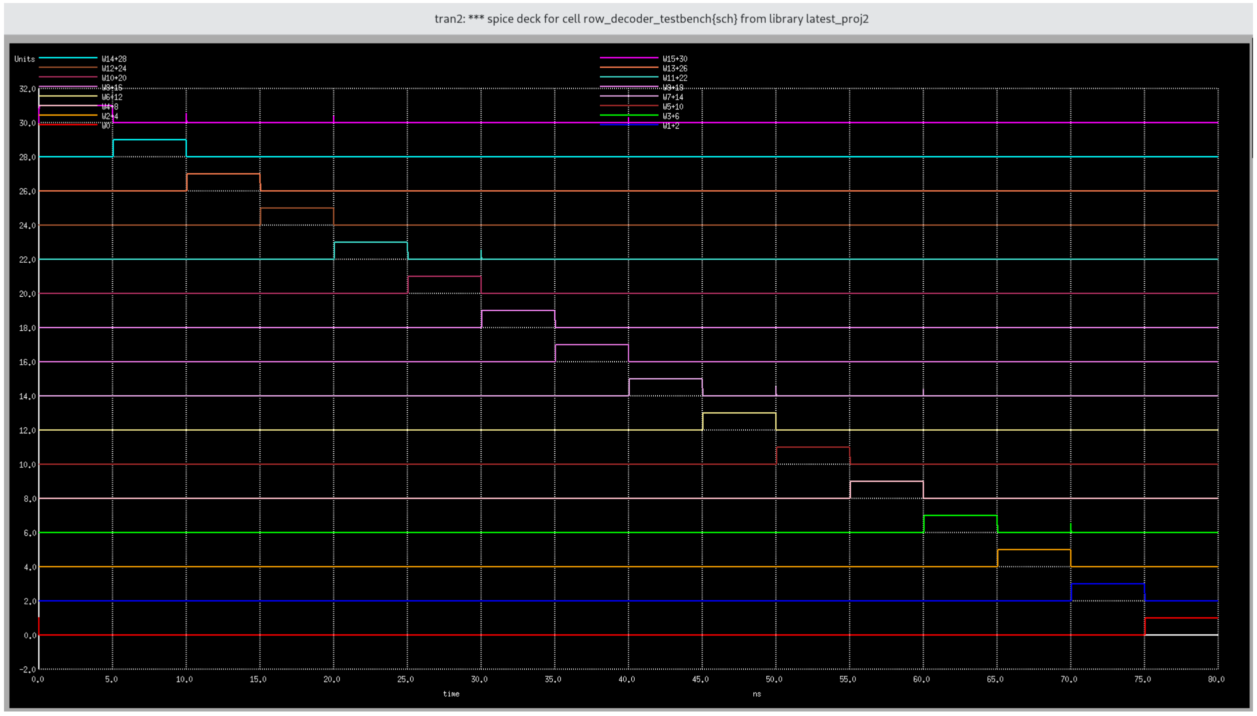
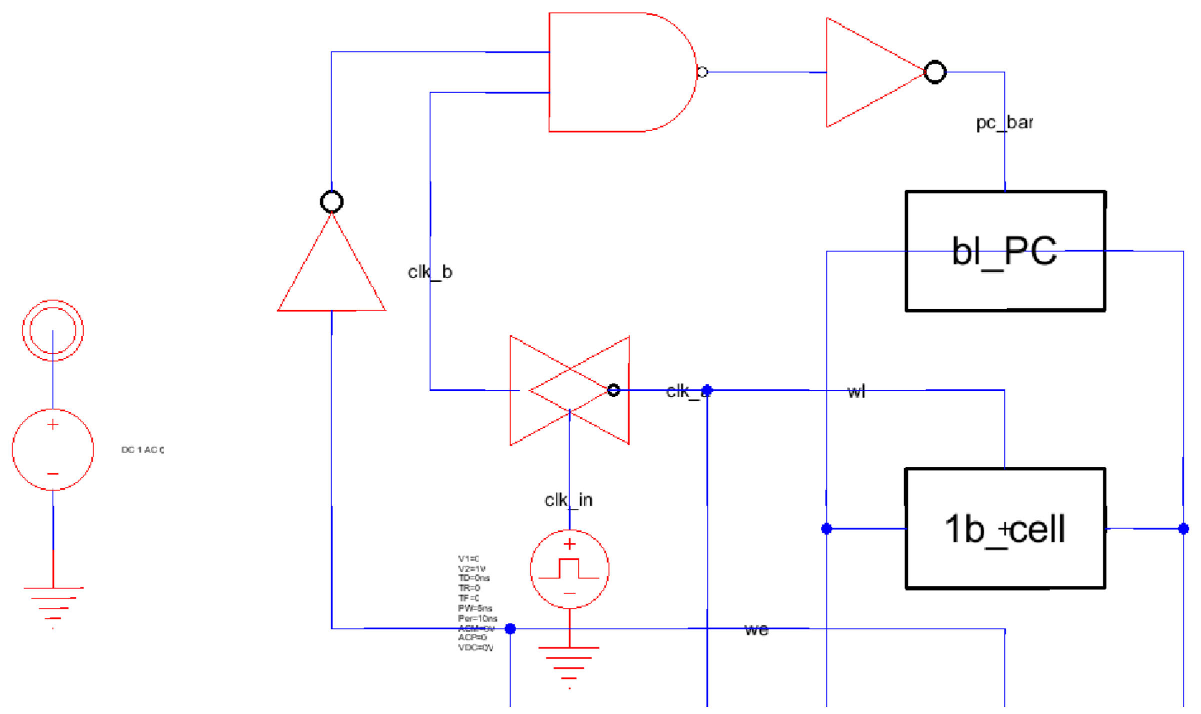
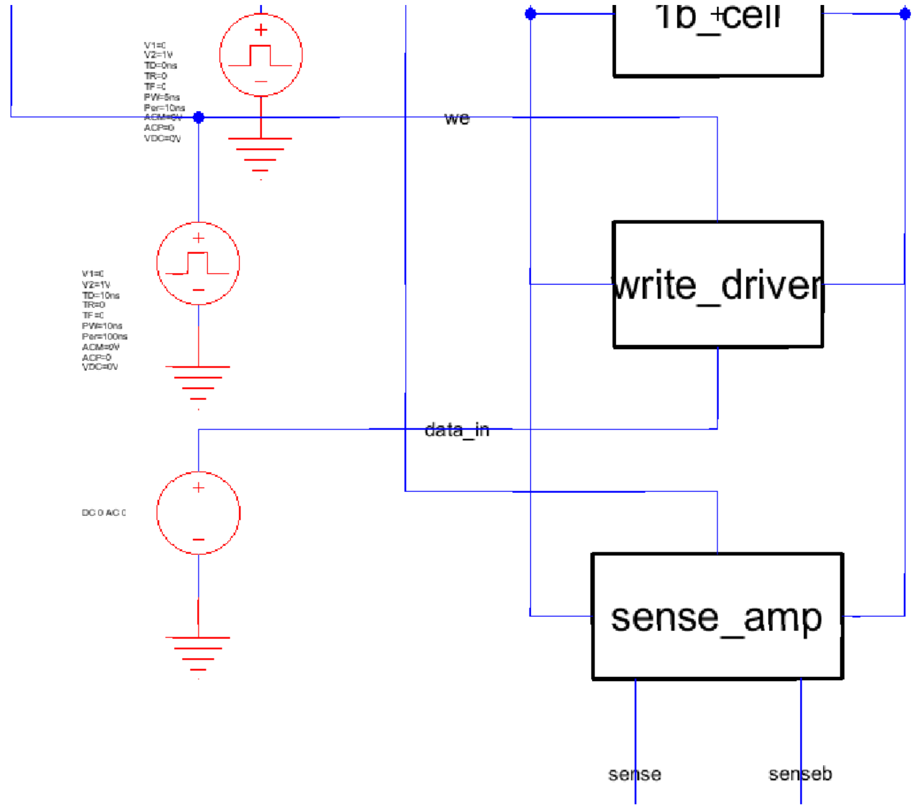


Figure 15: Row Decoder Verification

3.2 1b SRAM Cell



(a) RW Testbench - BL Precharge, 2 Phase clock, 1b SRAM Cell



(b) RW Testbench - Write Driver, Sense Amp, Data Input

Figure 16: RW Testbench

To test the 1b SRAM cell, we used the following test case (tab. 1):

1.	Write 1	Read 1	Read 1
2.	Write 1	Read 1	Read 1
3.	Write 0	Read 0	Read 0
4.	Write 0	Read 0	Read 0
5.	Write 1	Read 1	Read 1

Table 1: 1b SRAM Comprehensive Functionality Test Cases

Passing these cases shows that we can, 1). write data to the SRAM cell, 2). read written data, 3). perform non-destructive reads, 4). overwrite previous data with either the same or different values. Consequentially, this case ensures that memory periphery devices are properly working, although individual device tests will be given below.

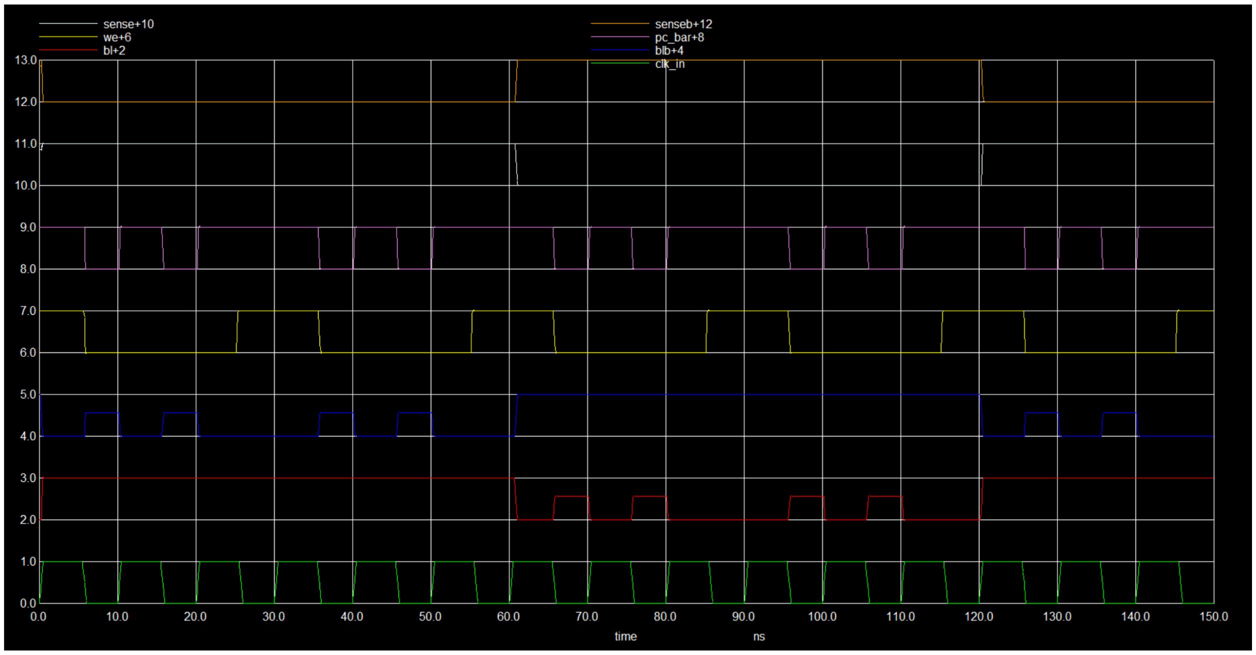


Figure 17: 1b SRAM Verification using tab. 1

Note that in the above plot, the key trace is sense (shown in the light grey colour, 2nd trace from the top). We see that we are able to read strong logic highs and lows in the intended cycle through the same transient.

3.2.1 Two-Phase Clock Generator

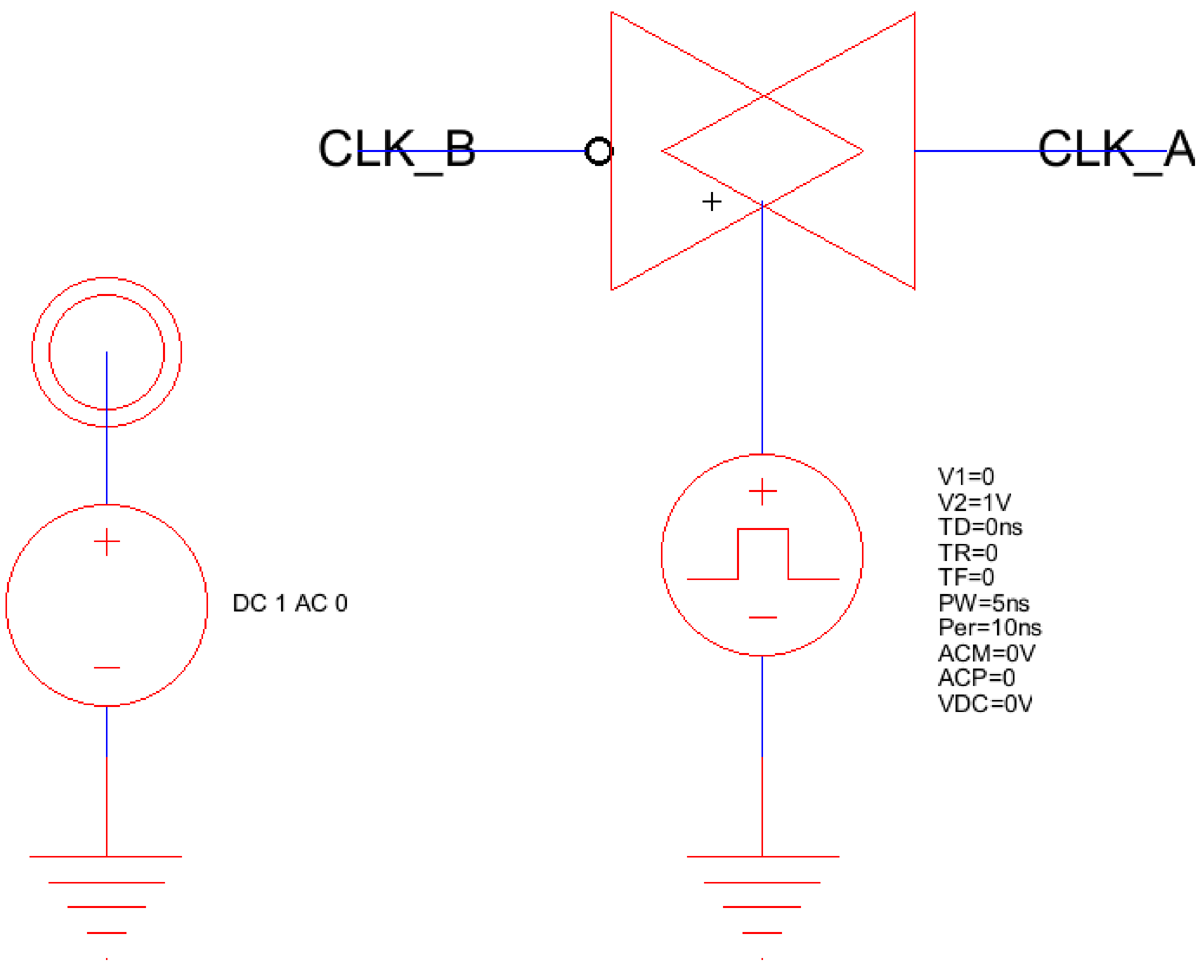


Figure 18: Testbench used to verify functionality of the Two-Phase Clock Generator

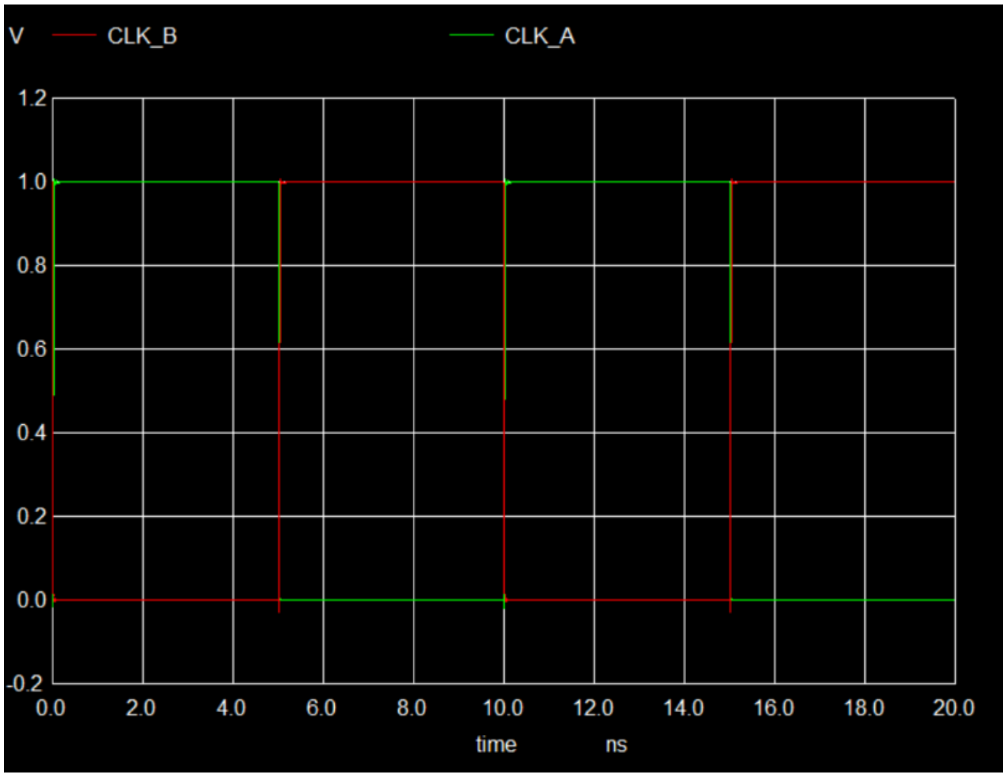


Figure 19: Transient simulation of Two-Phase Clock Generator Testbench

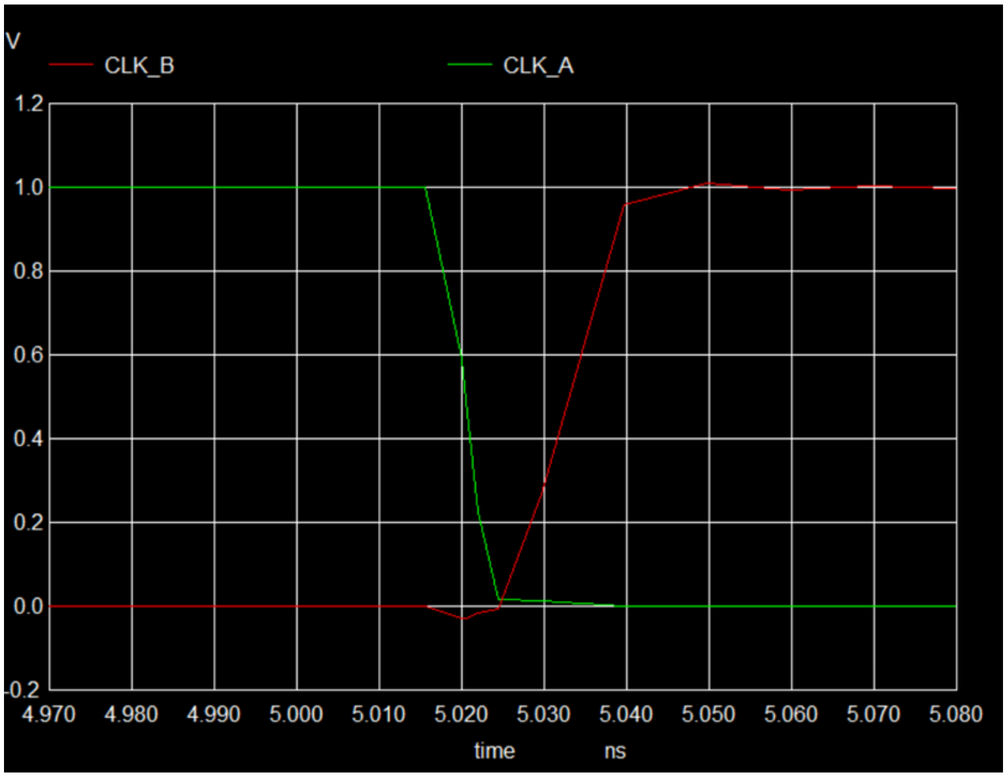


Figure 20: Visible non-overlapping clocks generated by the Two-Phase Clock Generator

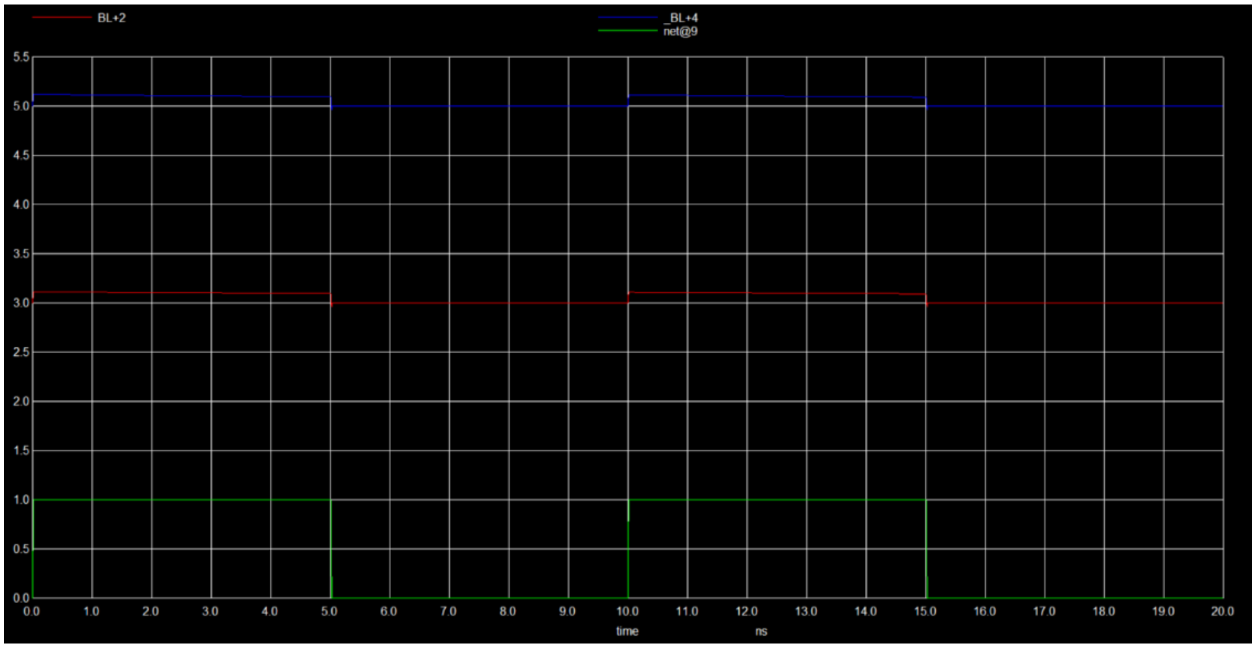


Figure 22: Functional verification of the bitline precharge circuit

3.2.2 Precharge Circuit

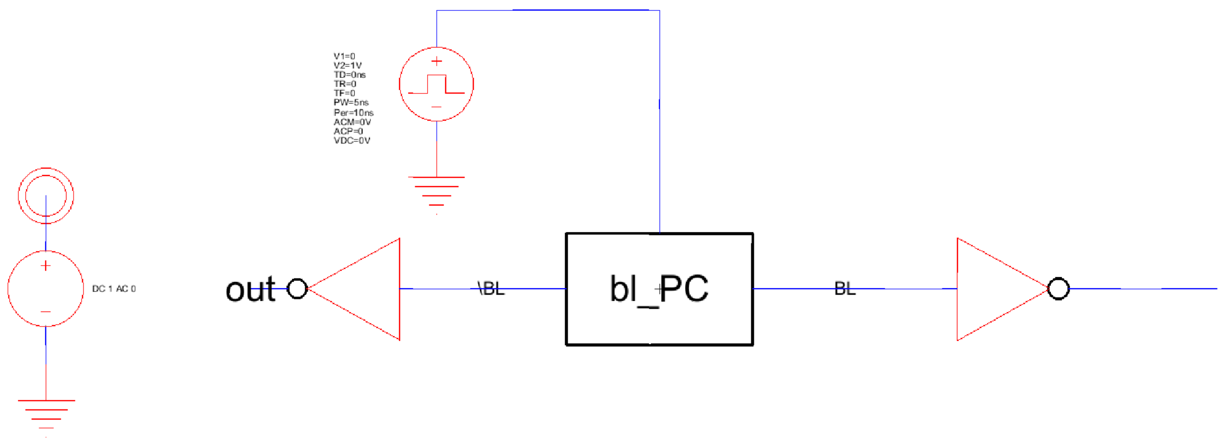


Figure 21: Testbench used to verify functionality of the Bitline Precharge circuit

3.2.3 Sense Amplifier

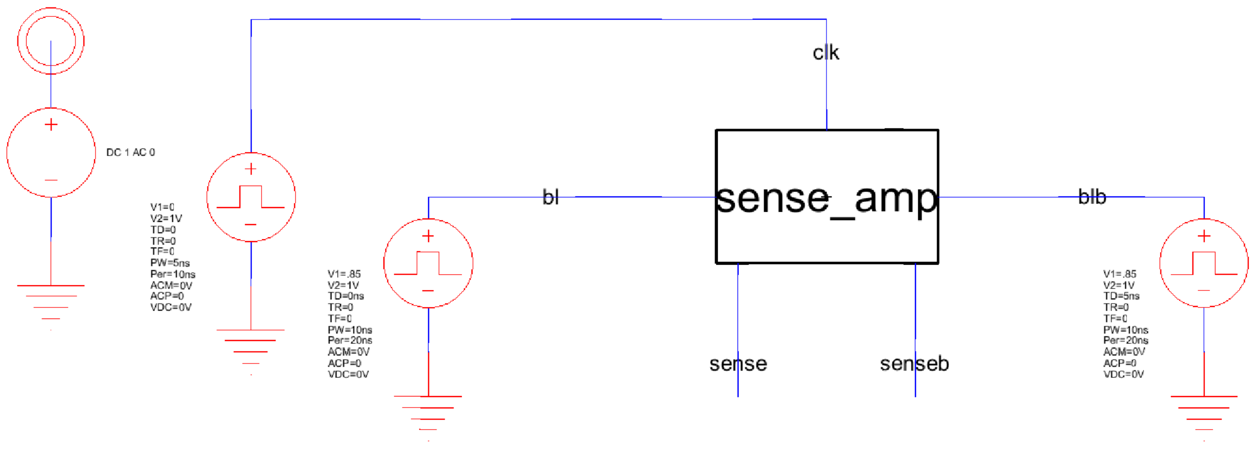


Figure 23: Testbench used to verify functionality of the

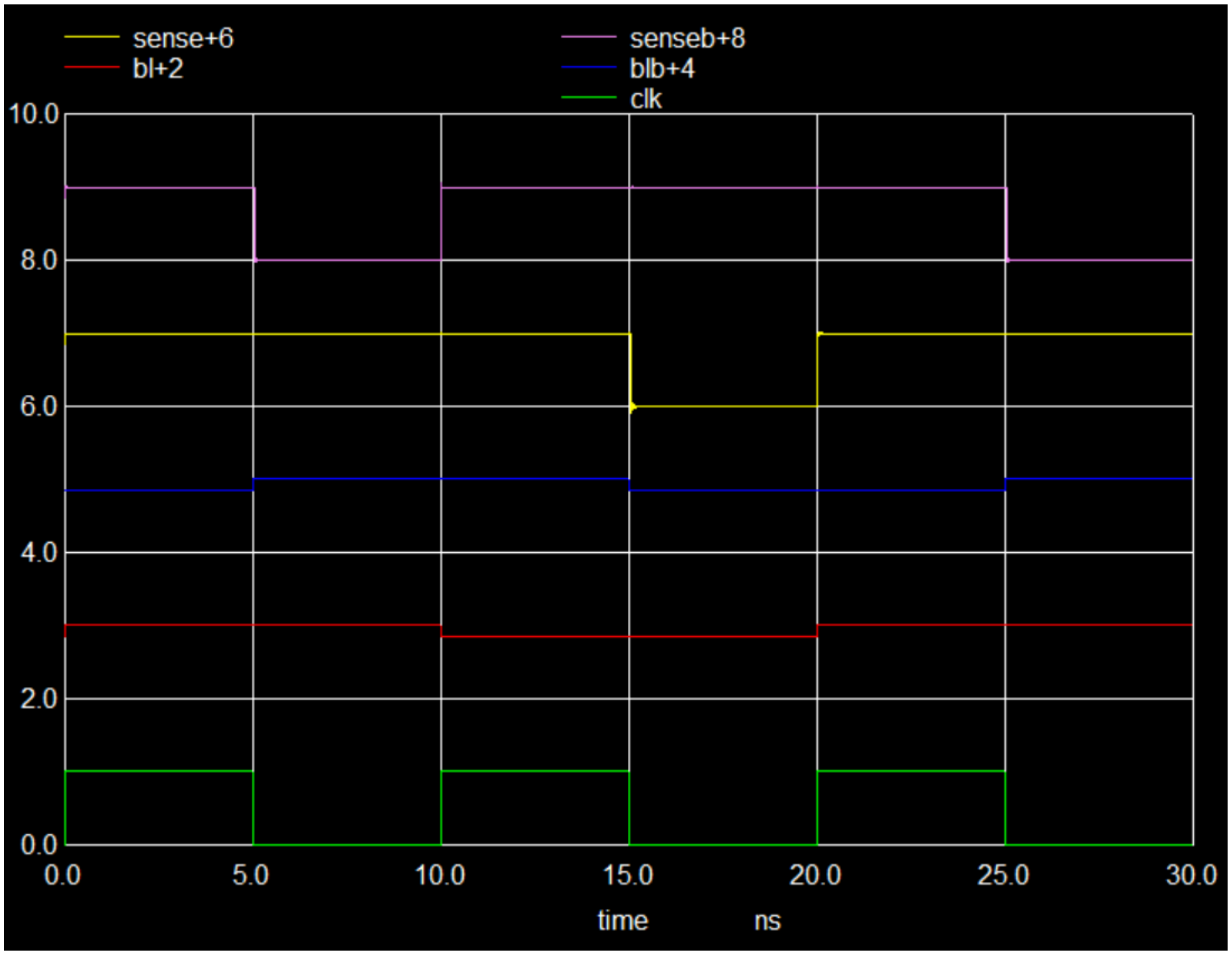


Figure 24: Functional verification of the sense amplifier

3.2.4 Write Driver

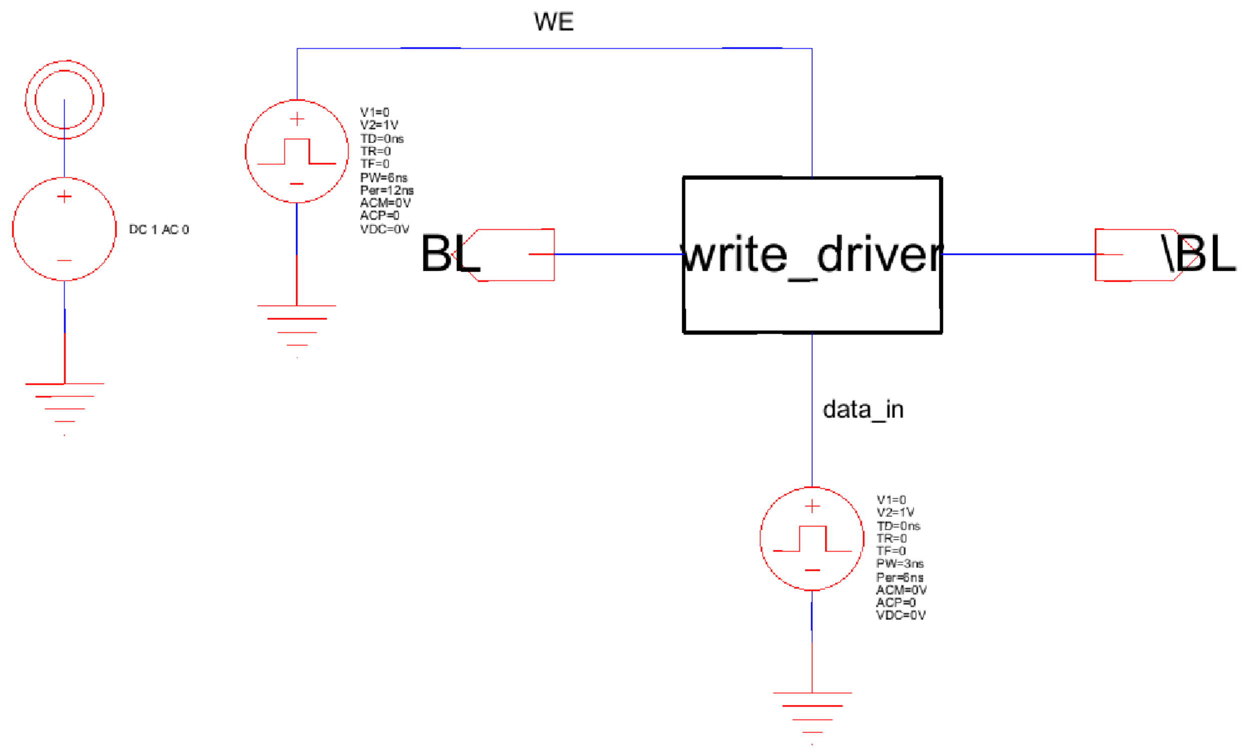


Figure 25: Testbench used to verify functionality of the write driver

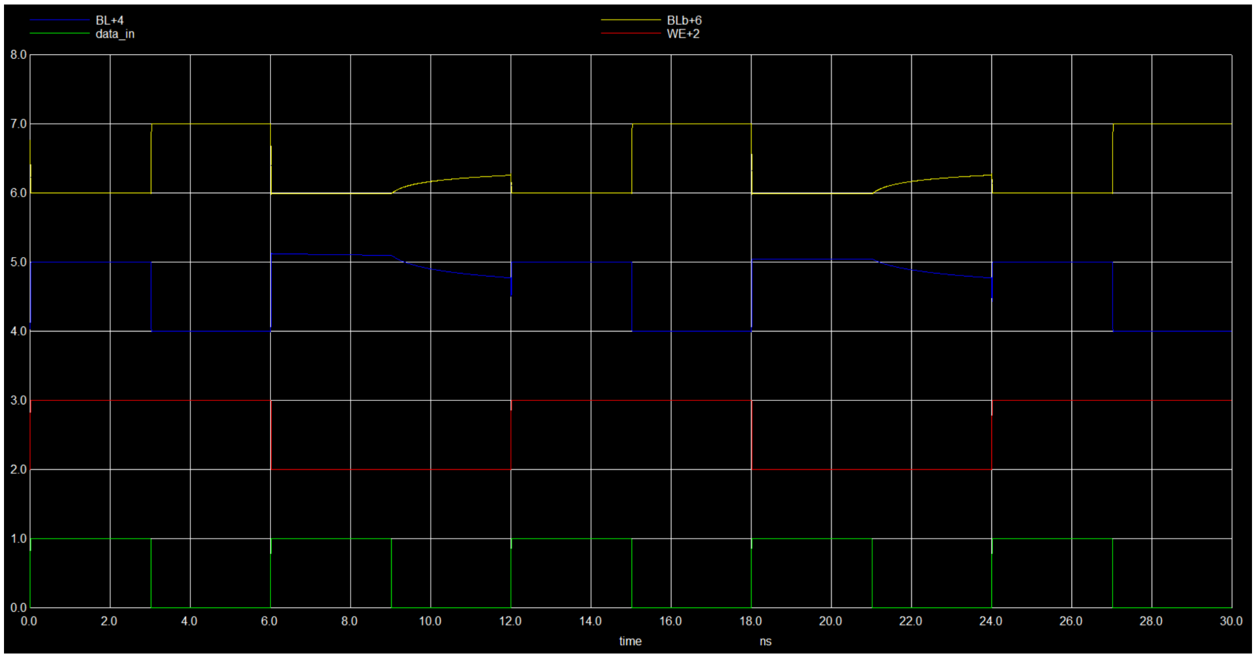


Figure 26: Functional verification of write driver and thus, the tristate buffer/inverters

3.3 16x4 SRAM Array

Figure 27 verifies correct read and write functionality across all 16 addressable rows of the SRAM array. To comprehensively test the row decoder and memory integrity, we applied 4-bit address inputs corresponding to all $2^4 = 16$ combinations by driving the decoder inputs (`A0` through `A3`) with staggered square waves of doubling periods. This ensures that the binary-encoded address increments from 0000 to 1111 over the 65 ns window, cycling through every unique wordline activation. During each address phase, a unique 4-bit data input is written to the targeted row, and subsequently read. The `we` waveform controls the write phase, while the output signals (`sense[3:0]`) confirm that the correct data is retrieved from the corresponding word. As shown, the sensed outputs follow the staggered address inputs exactly, confirming the correctness of the row decoder, data path, and peripheral circuits over all combinations.

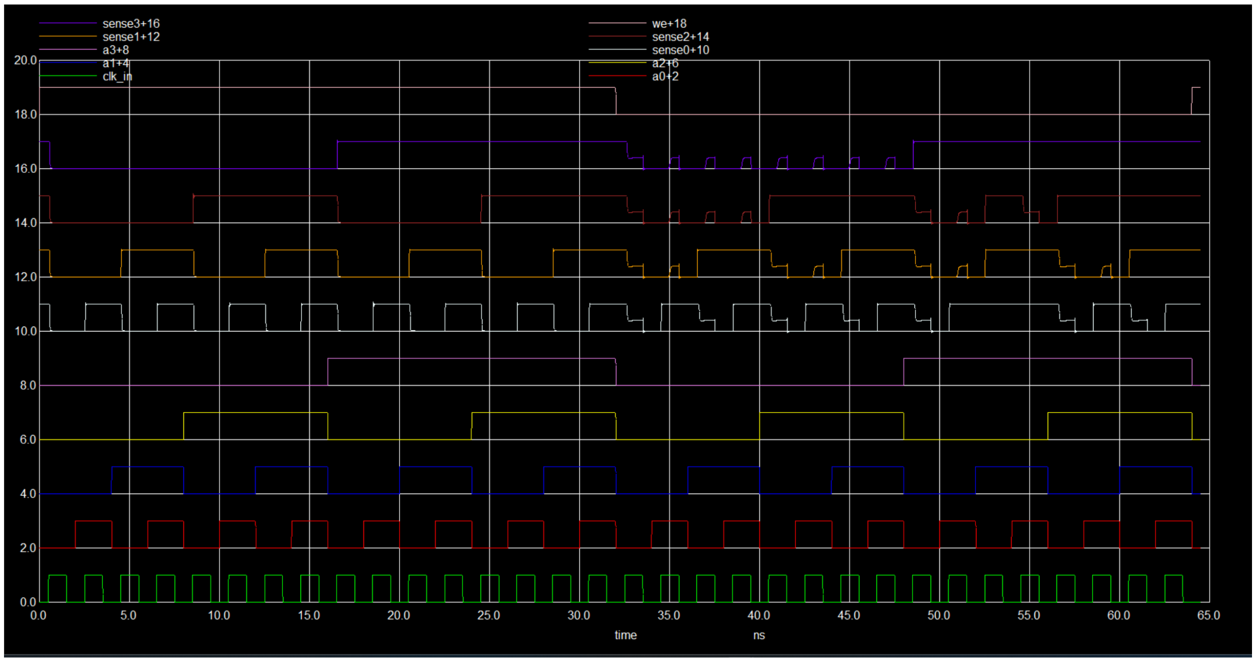


Figure 27: Read and Write Verification for all 16 Addresses

The input waveforms used to the row decoder are shown in Figure 28.

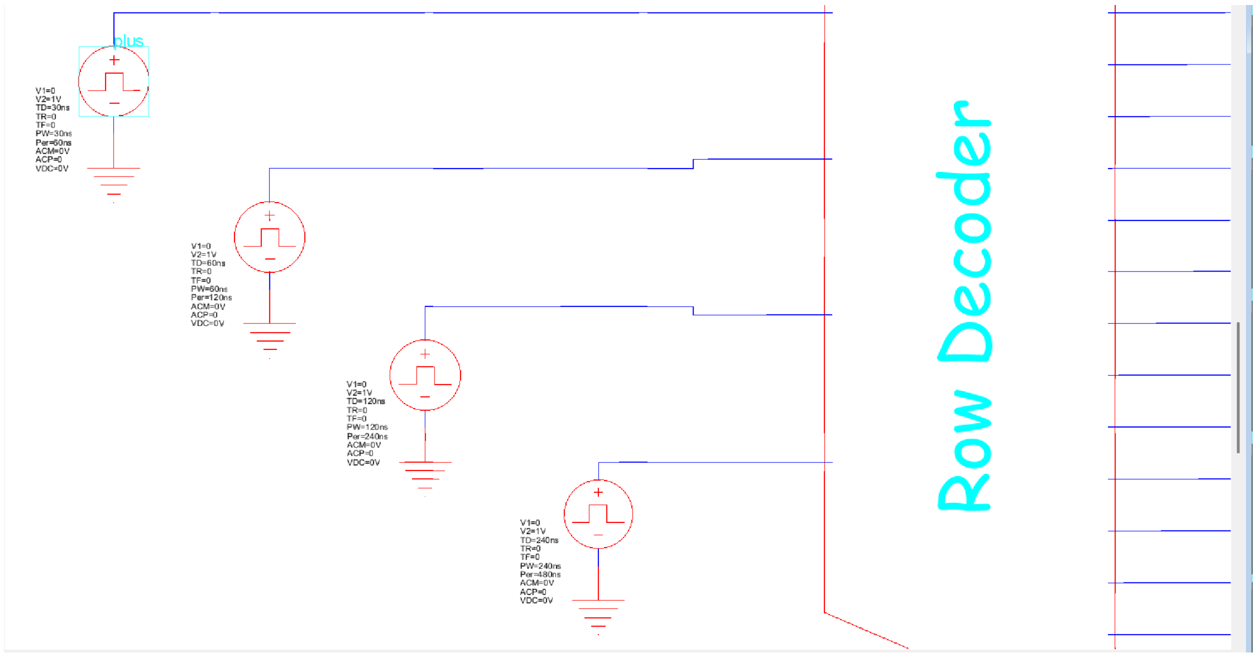


Figure 28: Row Decoder Inputs for Functional Verification of 64 Cell SRAM Array

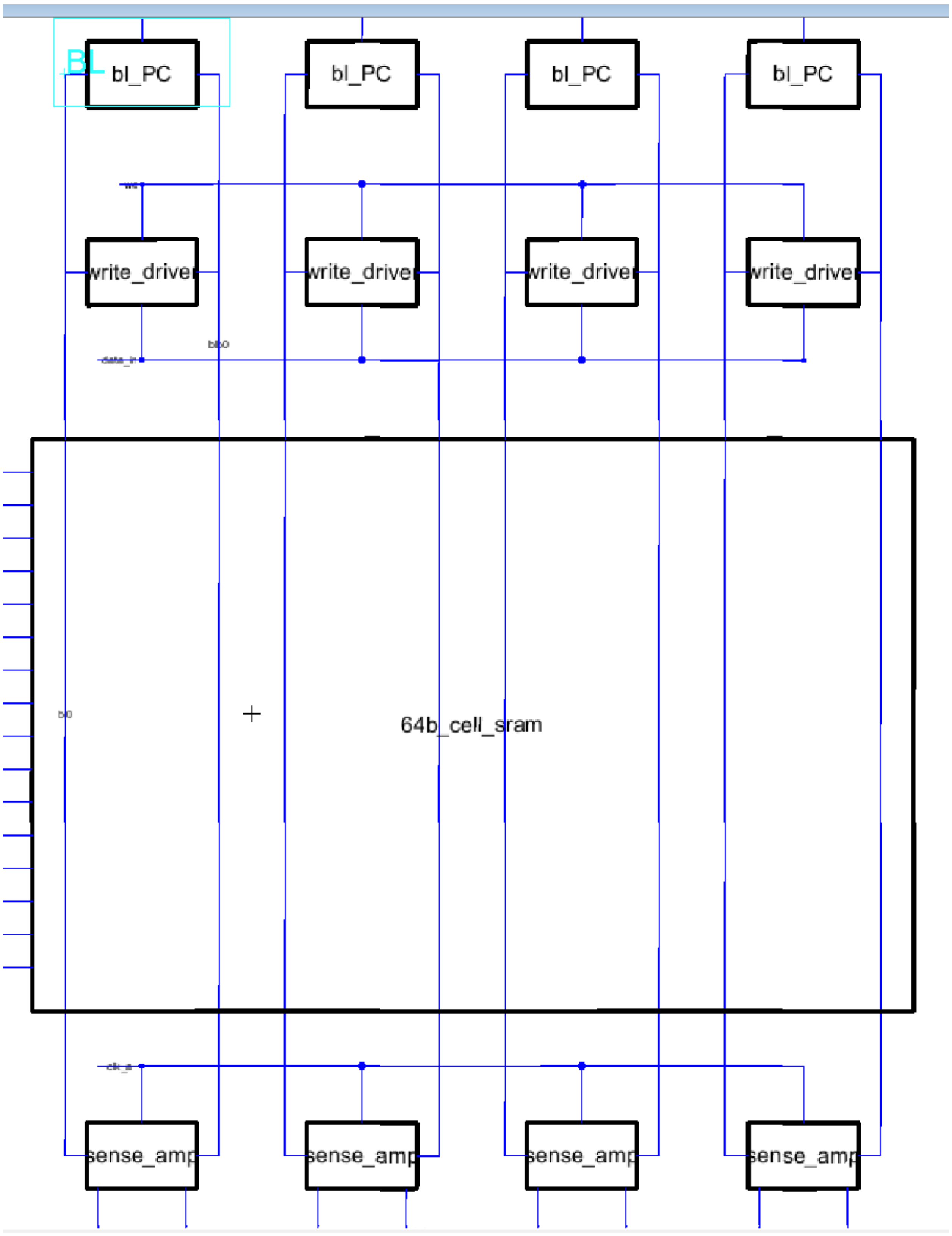


Figure 29: 64-bit SRAM

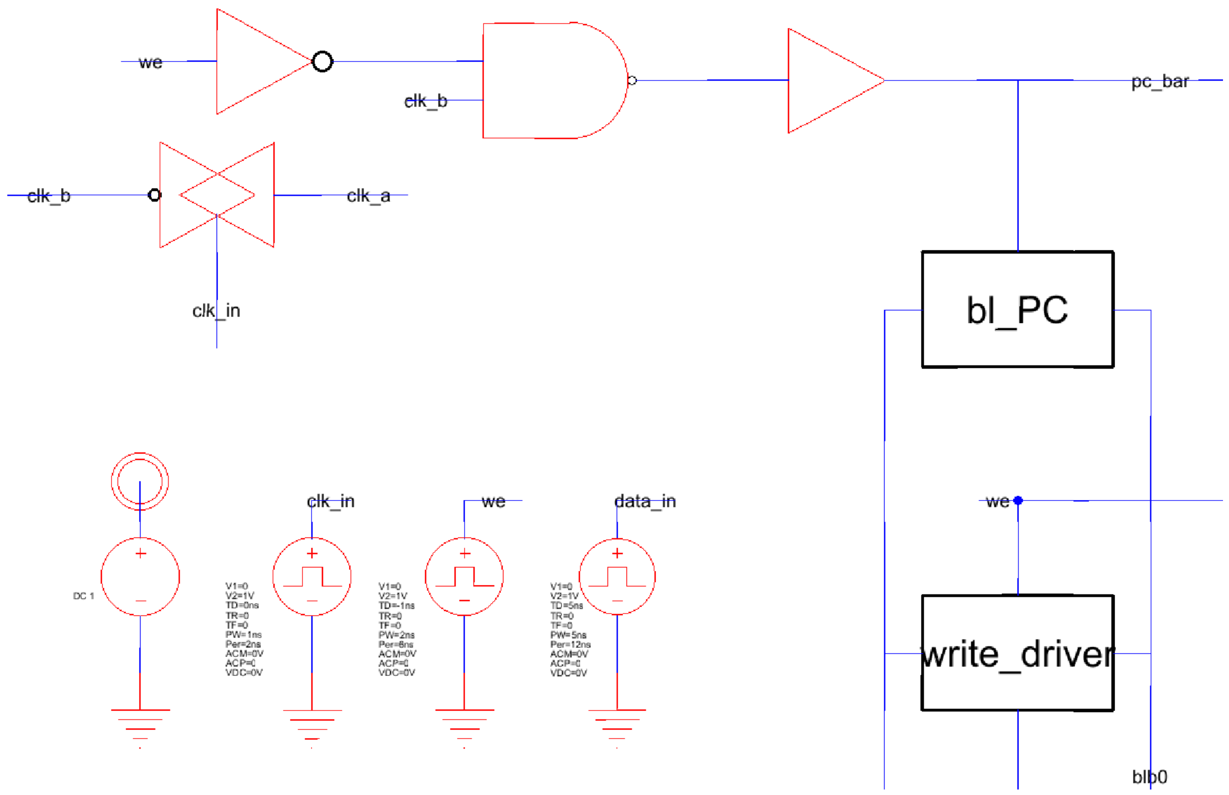


Figure 30: 16x4 SRAM Cell Testbench

4 Design Metrics

4.1 Delay Analysis

The following testbench was used to carry out the delay analysis.

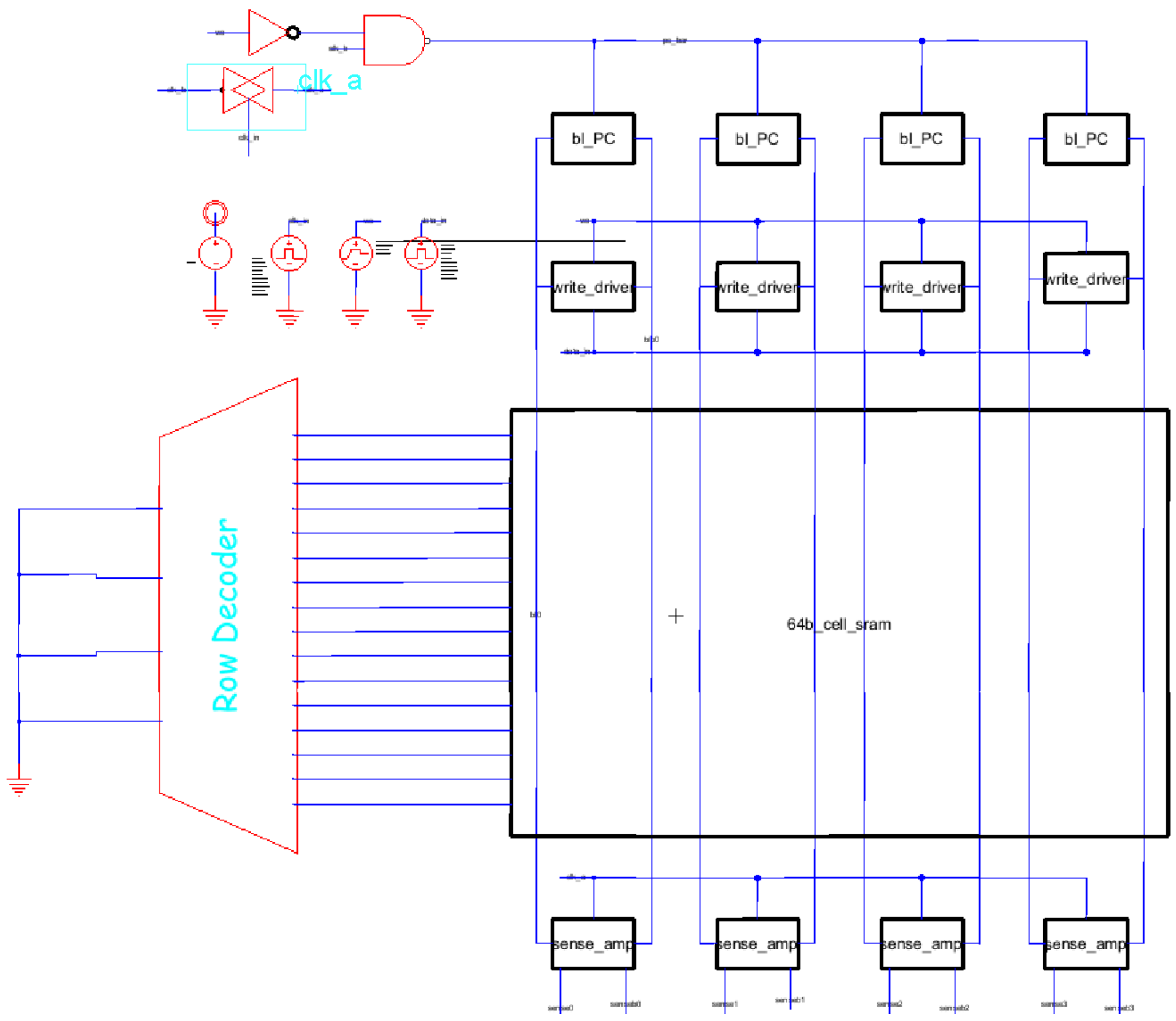


Figure 31: Complete testbench setup for delay analysis

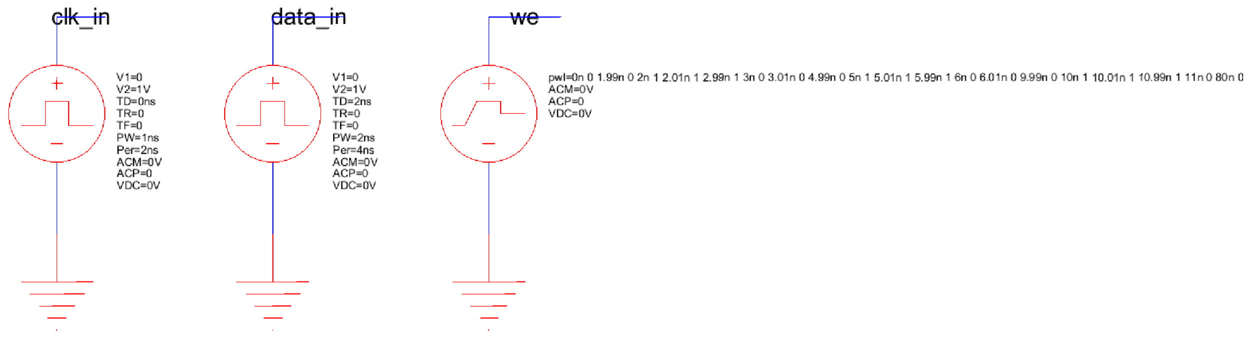


Figure 32: Waveforms used for delay analysis

The write enable waveform was supported by a VPWL as this allowed for selective activation of the write driver. For the read delay analysis, this flexibility allowed for the output at the sense amplifier to be switched when we wanted to measure a fall, and for the out put to be reset to a logic high when we wanted to measure a rise. For the write delay analysis, the tuning of the write enable signal allowed or simulataneous measurement of both rise and fall times at the internal output node of the targeted 1-bit cell.

4.1.1 Read Delay

For reads, the critical path is from the $data_{in}$ to the output of the sense amplifier. This analysis is seperated into fall time and rise time delay for the read operation.

Fall Time Read Delay Our total reading fall time delay, as measured from the rising edge of $clk_a = sense_{clk}$ to the falling edge of the sense amplifier (0000 was written to the first word) was approximately 121.1ps. The following transient simulation was generated by using a VPWL input to the write enable input, maintaining a logic high at write enable such that the bitlines were precharged and the output was written to be a logic low at the sense amplifier output.

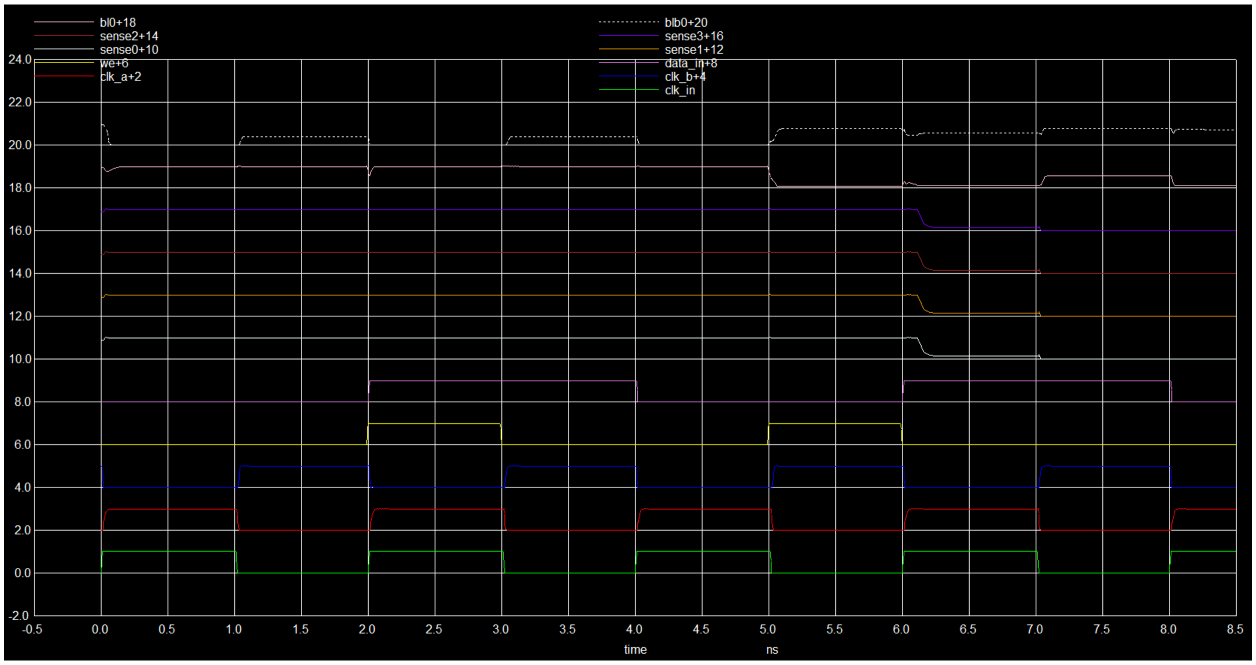


Figure 33: Complete timing diagram for fall time read delay analysis

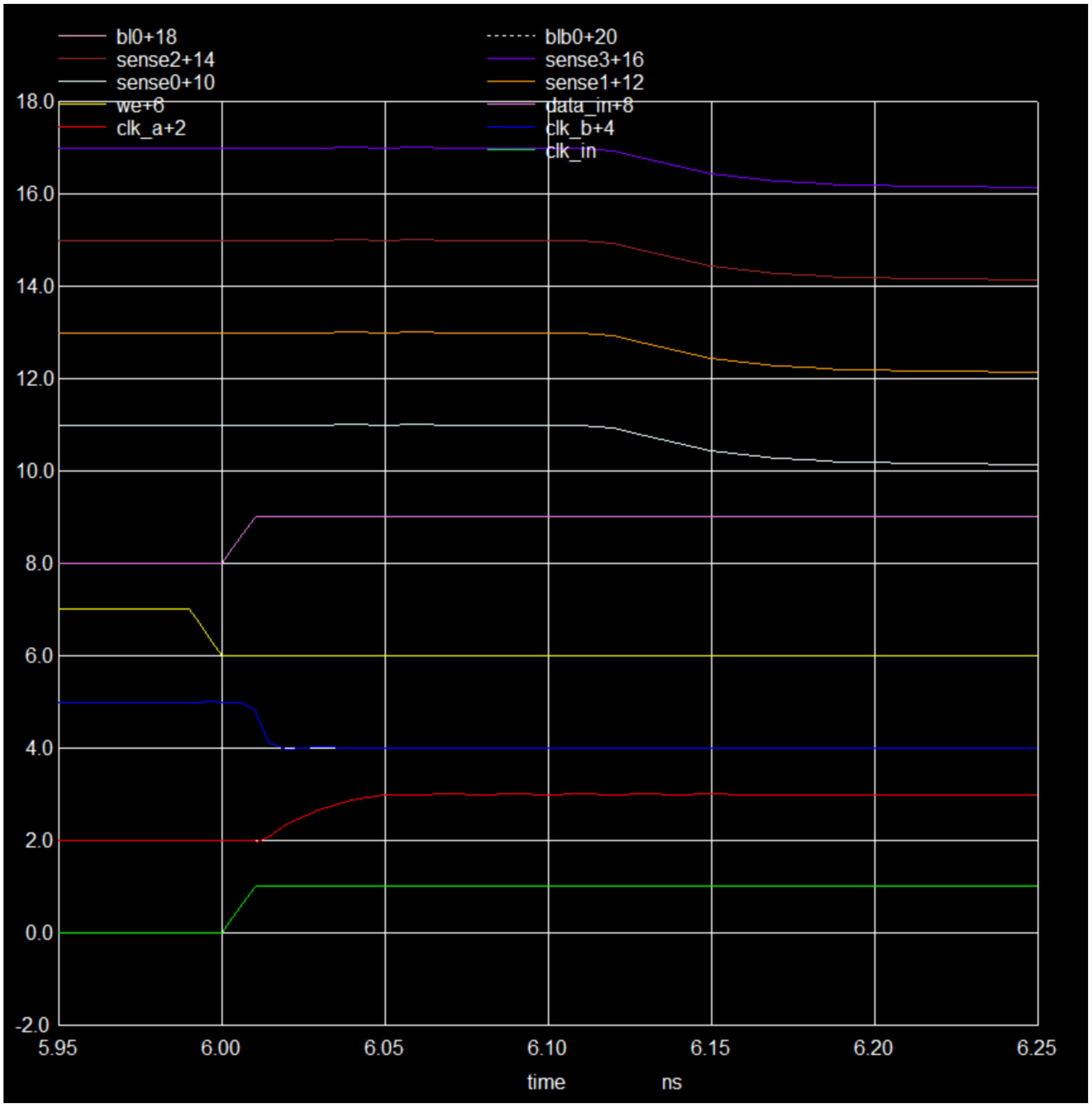


Figure 34: Section of timing diagram used to extract fall time read delay

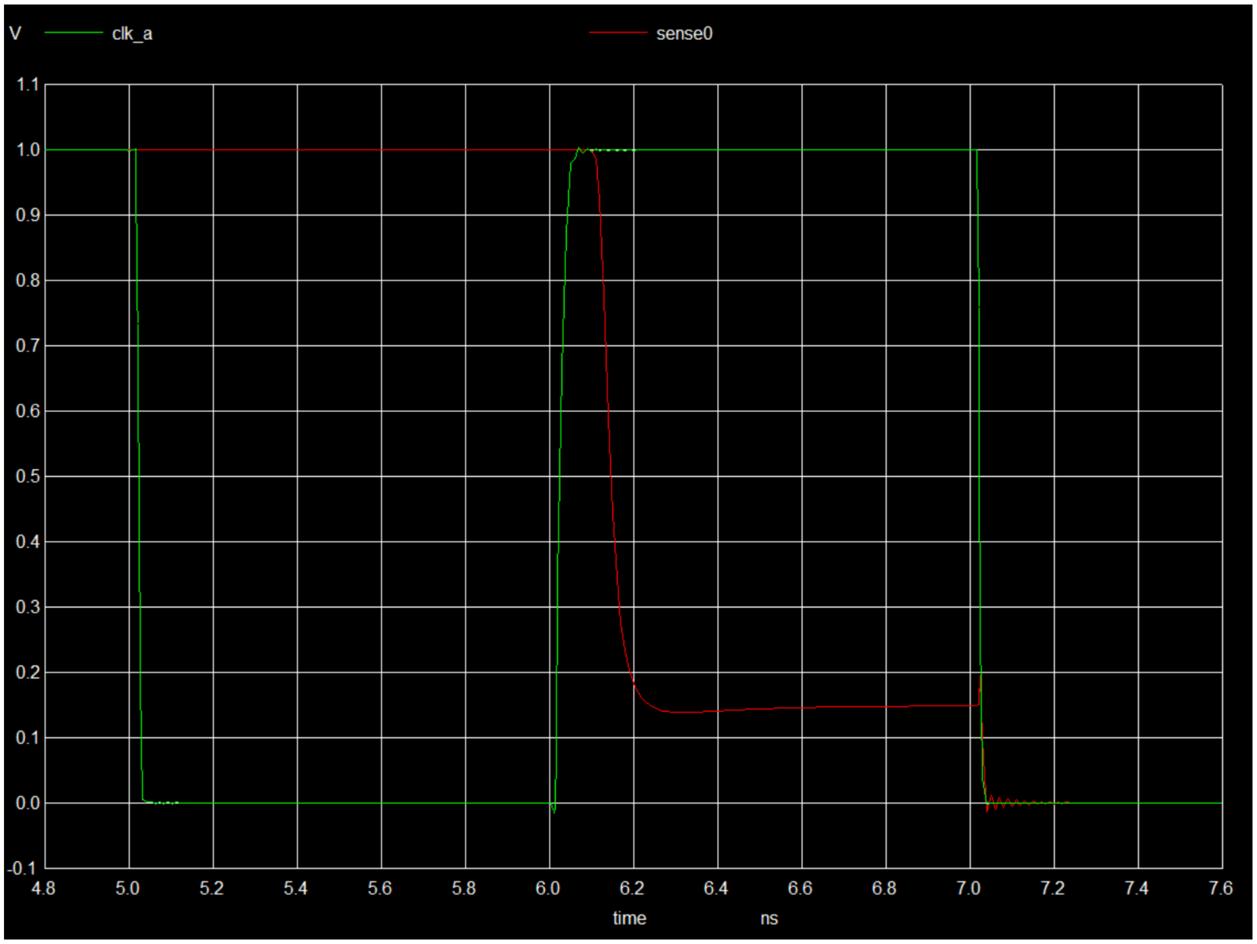
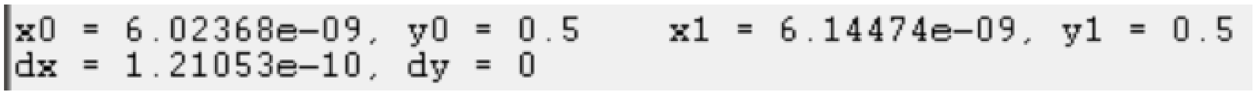


Figure 35: Aligned traces for $sense_{clk}$ (or as shown here, clk_a) and $sense_0$



```
x0 = 6.02368e-09, y0 = 0.5     x1 = 6.14474e-09, y1 = 0.5
dx = 1.21053e-10, dy = 0
```

Figure 36: Ngspice terminal output affirming an approximate fall time read delay of 121.1ps

Rise Time Read Delay Our total reading rise time delay, as measured from the rising edge of $clk_a = sense_{clk}$ to the rising edge of the sense amplifier (1111 was written to after 0000) was approximately 118.9ps.

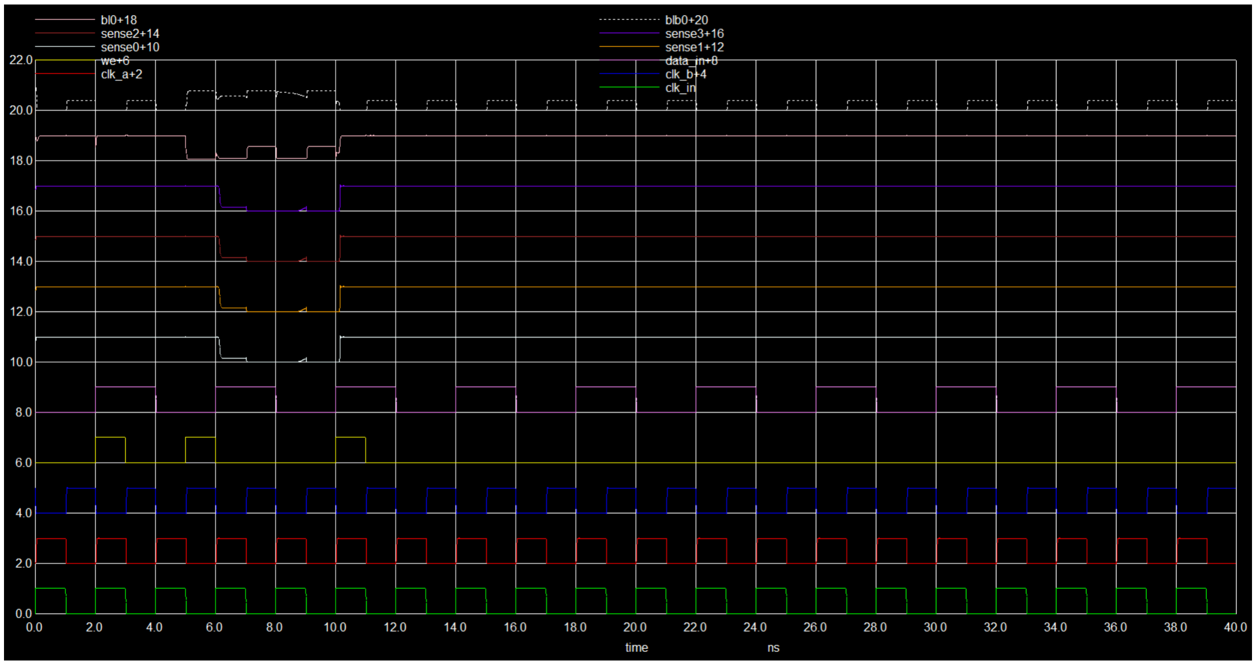


Figure 37: Complete timing diagram for rise time read delay analysis

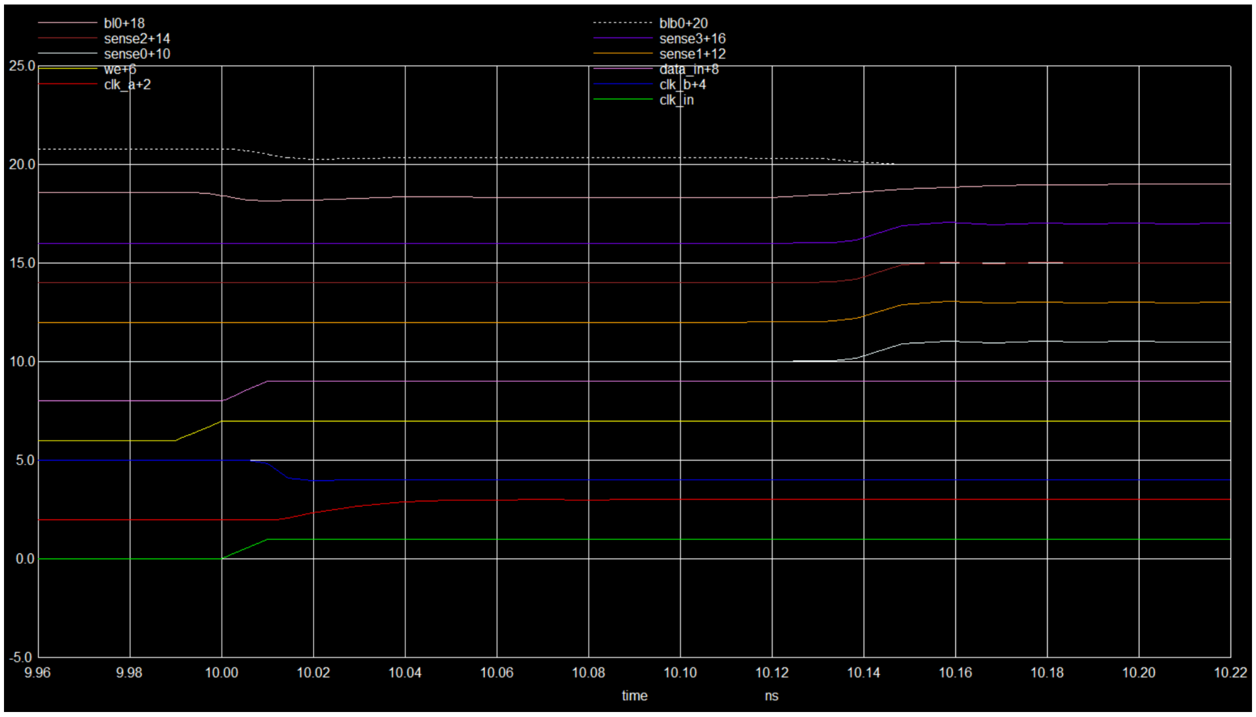


Figure 38: Section of timing diagram used to extract rise time read delay

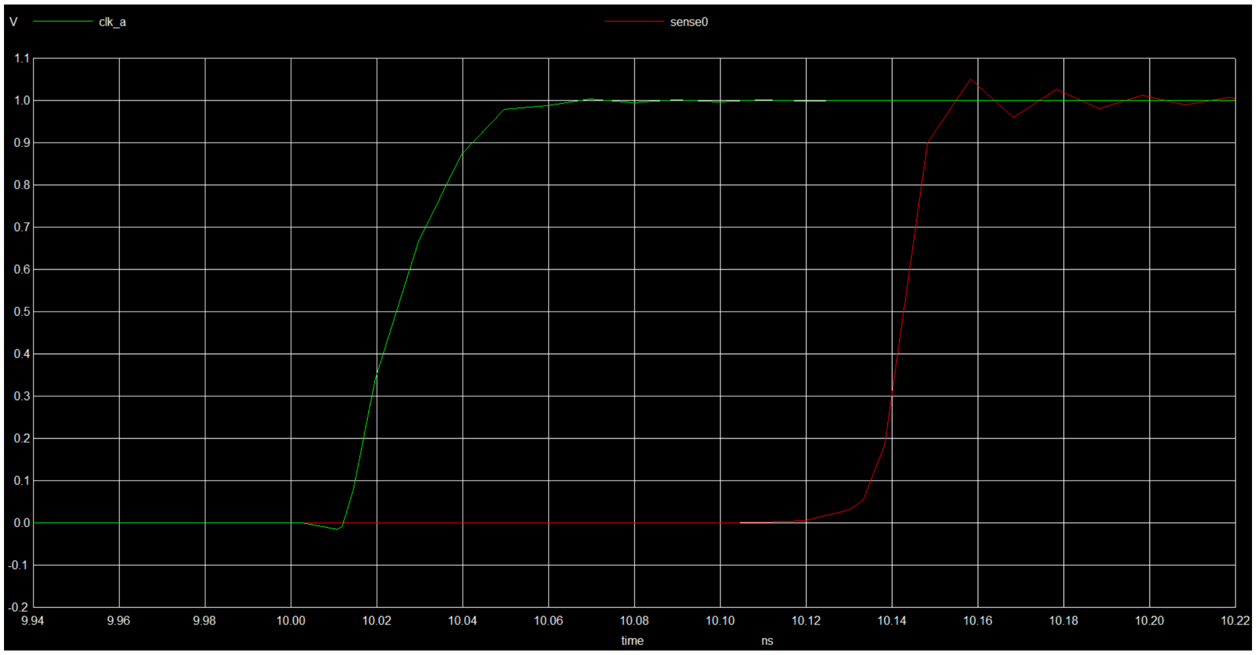
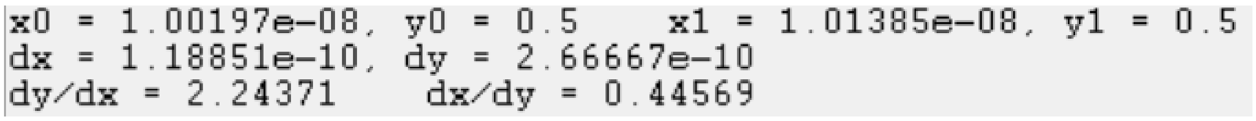


Figure 39: Aligned traces for *senseclk* (or as shown here, clk_a) and $sense_0$



```
x0 = 1.00197e-08, y0 = 0.5    x1 = 1.01385e-08, y1 = 0.5
dx = 1.18851e-10, dy = 2.66667e-10
dy/dx = 2.24371    dx/dy = 0.44569
```

Figure 40: Ngspice terminal output affirming an approximate rise time read delay of 118.9ps

4.1.2 Write Delay

For writes, the critical path is from the write enable input to the internal "Q" output node within a 1-bit cell. The same setup was used as from the read delay analysis, with the key difference being the traces being analysed for delay. The output node of the cell at row 0, column 0 was identified as `x-64b-cell@0.x-16b-colu@0.x-1b-cell-@0.` by inspecting the SPICE deck of the testbench as well as the Ngspice terminal output of the transient simulation. Any future references to an output node of "output@00" should be interpreted as the aforementioned node.

The following transient simulation provides a simple plot through which both the write fall time delay and write rise time delay can be analysed.

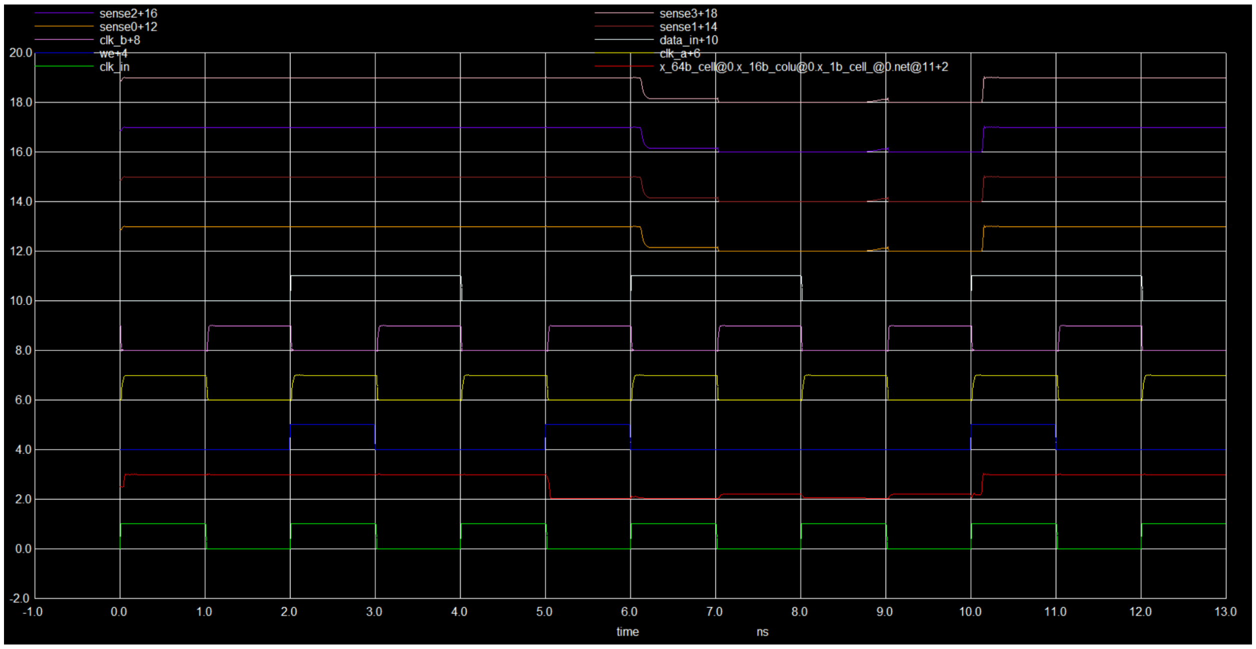


Figure 41: Complete transinet simulation including voltage at output@00 shown in bright red, as well as write enable shown in blue

Both fall and rise time write delay will be further analysed using this simulation below.

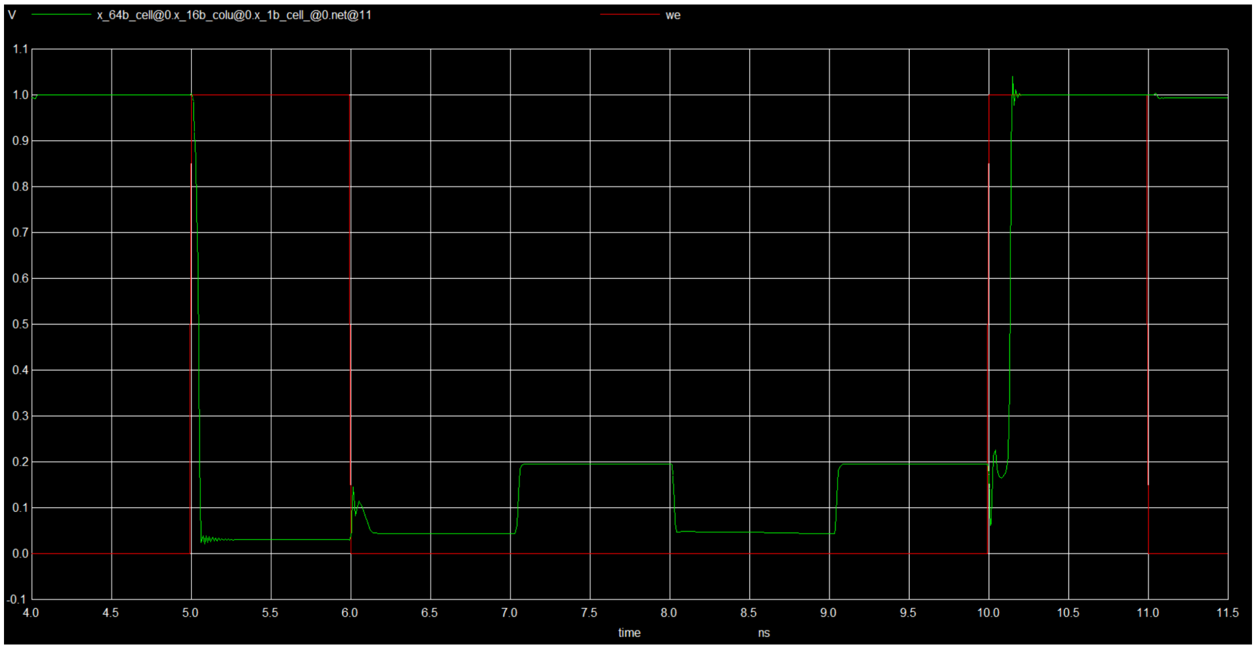


Figure 42: Relevant traces for write delay analysis

Fall Time Write Delay Our total delay, measured from the rising edge of the clock to the falling edge of the output@00 was approximately 45.2ps.

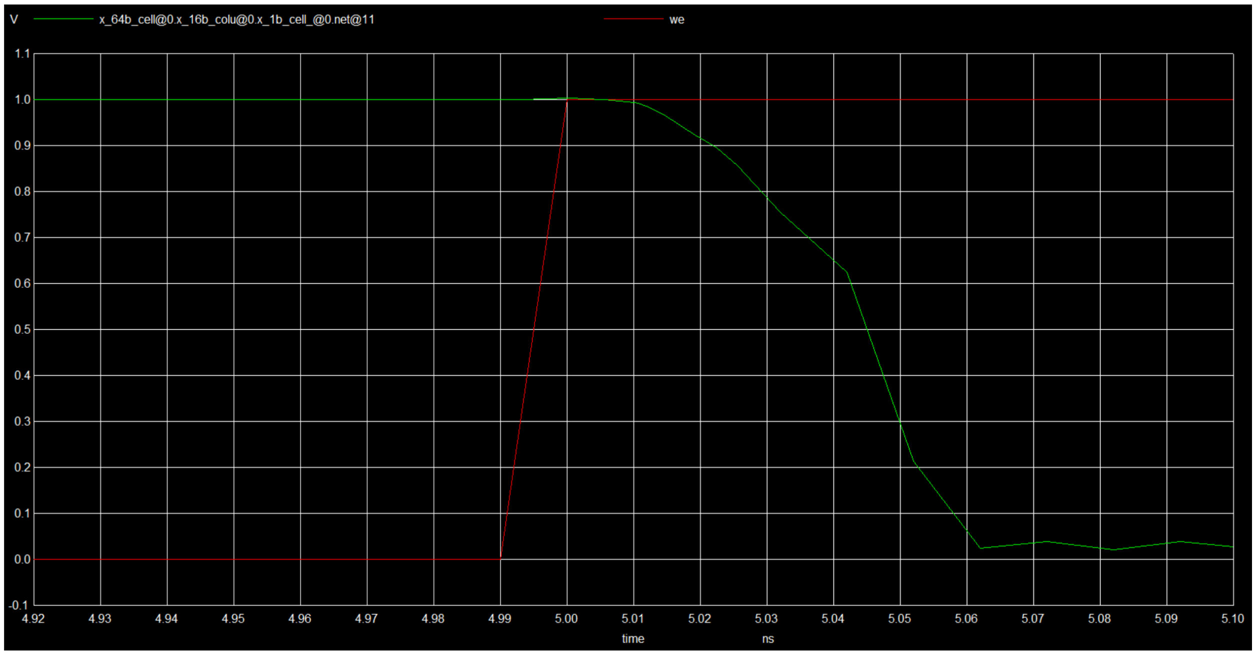
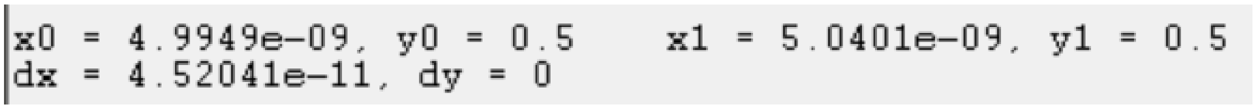


Figure 43: Side by side analysis of write enable and output@00 traces



```
x0 = 4.9949e-09, y0 = 0.5     x1 = 5.0401e-09, y1 = 0.5
dx = 4.52041e-11, dy = 0
```

Figure 44: Ngspice terminal output affirming an approximate fall time write delay of 45.2ps

Rise Time Write Delay Our total delay, measured from the rising edge of the clock to the rising edge of the output@00 was approximately 137.5ps.

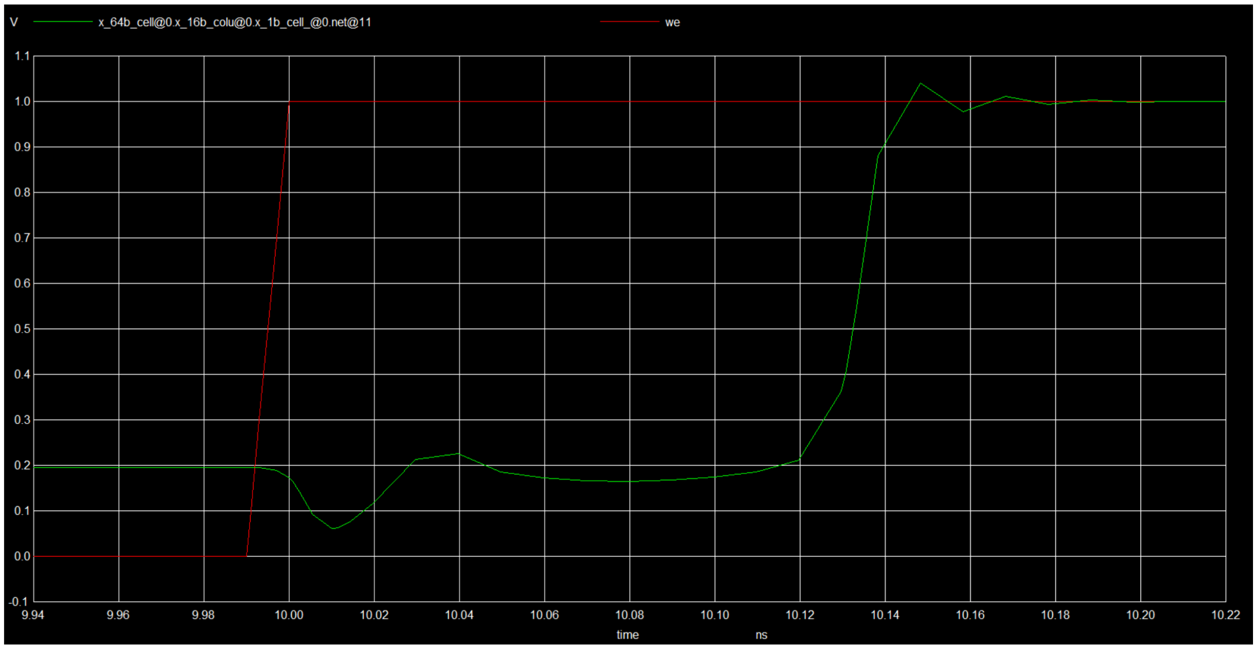
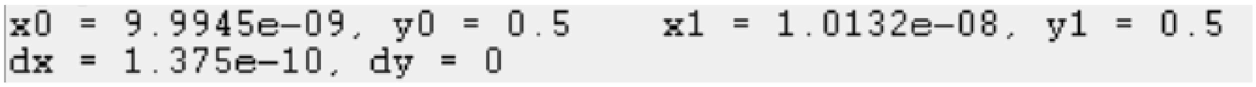


Figure 45: Side by side analysis of write enable and output@00 traces



```
x0 = 9.9945e-09, y0 = 0.5     x1 = 1.0132e-08, y1 = 0.5
dx = 1.375e-10, dy = 0
```

Figure 46: Ngspice terminal output affirming an approximate rise time write delay of 137.5ps

4.1.3 Clocking Restrictions

After thorough delay analysis, we find that the rise time write delay of the SRAM Memory Array is the largest delay, at 137.5ps. To find the minimum clock frequency of our design, we can assume a 50% duty cycle and account for marginal headroom over our maximum measured delay of 137.5ps and establish a pulse width of 140ps, and thus a clock period of 280ps. This corresponds to a clock frequency of approximately 3.57 GHz.

$$\texttt{CLK frequency} = \frac{1}{280\text{ps}} = 3.571428570 \cdot 10^{9}\ \text{Hz} \approx 3.57\ \text{GHz}$$

4.2 Power

Using the established maximum clock frequency from the delay analysis of roughly 3.57 GHz, the following testbench was setup in order to analyse power thorugh the 64 bit Memory Array.

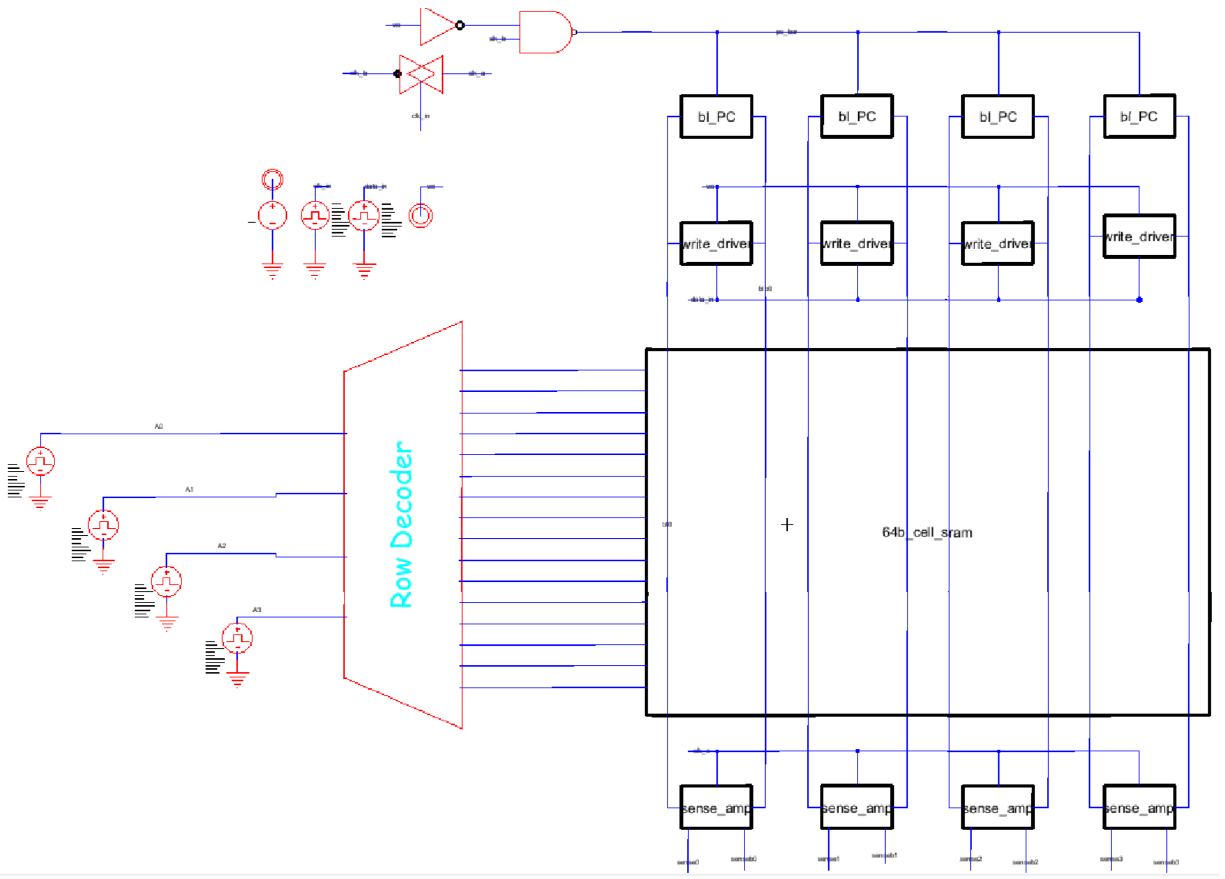


Figure 47: Complete testbench for power consumption

Aligning with the clock period and frequency previously discussed, we used the following pulse as our clock input.

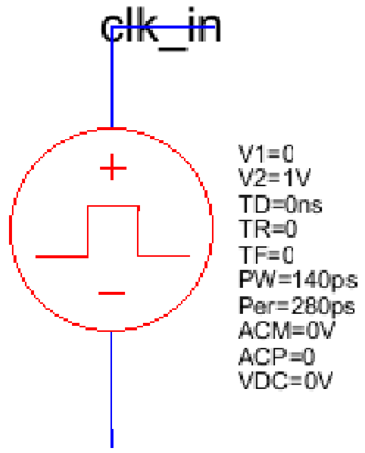


Figure 48: Clock used for power consumption analysis

The address bits shown below were used, in compliance with the clock used. The pulses driving the four address bits were staggered in multiples of one another to ensure all 16 word lines are activated for each of the full array write cycles.

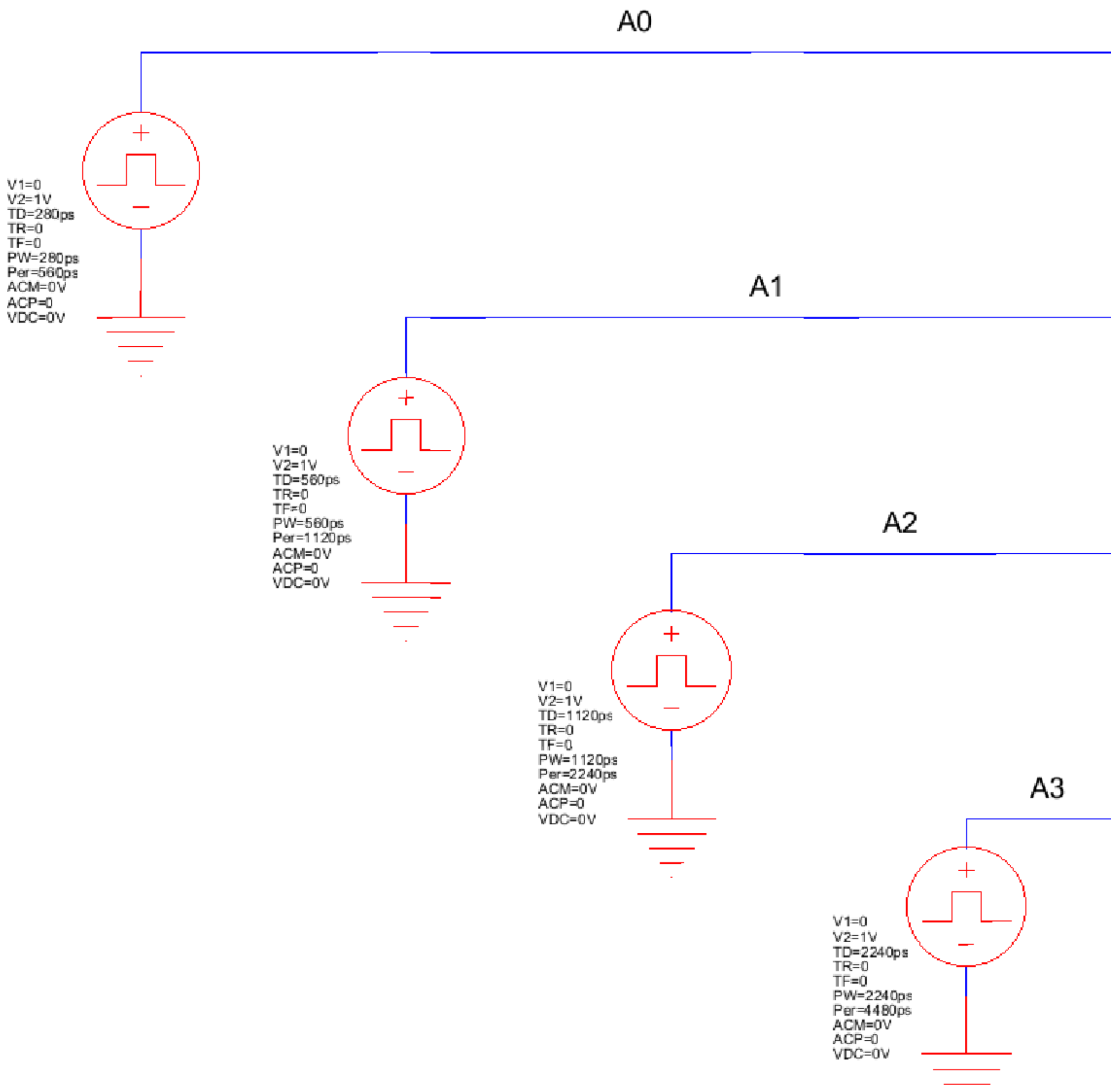


Figure 49: Address bits driven by VPulses to ensure complete activation of the memory array

Furthermore, the data driving the write driver is driven by a pulse such that the full memory array is written with 0s before being written by 1s, and then repeating the data input sequence. Moreover, since we are interested in constantly writing the full array, write enable is held at a constant logic high.

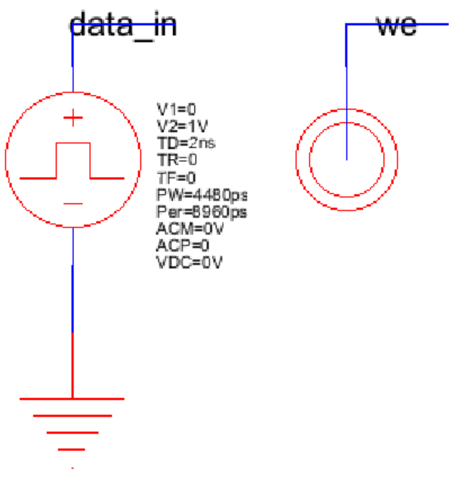


Figure 50: `data-in` and `we`

A transient simulation was carried out such that the entire array was written with 0s, then 1s. This two full array write operatioins were repeated 4 times. The power consumption during this operation is calculated by measuring the average current through the global VDD source. The simulated signals and measurements are shown in the figures below.

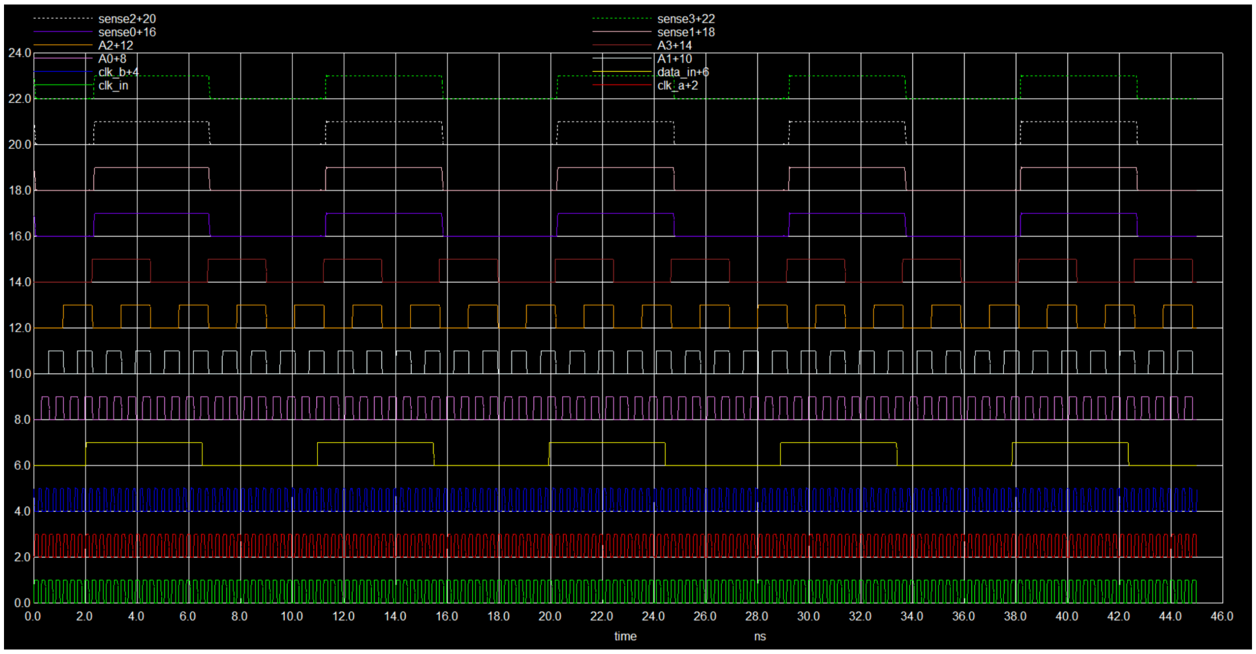
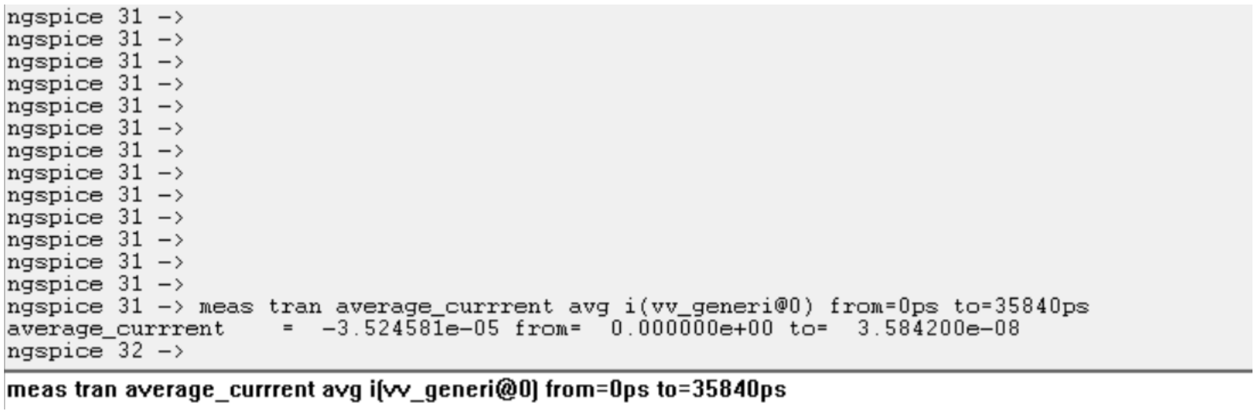


Figure 51: Complete transient response of the power consumption analysis



```
ngspice 31 ->
ngspice 31 ->
ngspice 31 ->
ngspice 31 ->
ngspice 31 ->
ngspice 31 ->
ngspice 31 ->
ngspice 31 ->
ngspice 31 ->
ngspice 31 ->
ngspice 31 ->
ngspice 31 ->
ngspice 31 ->
ngspice 31 -> meas tran average_currrent avg i(vv_generi@0) from=0ps to=35840ps
average_currrent    =  -3.524581e-05 from=  0.000000e+00 to=  3.584200e-08
ngspice 32 ->
meas tran average_currrent avg i(vv_generi@0) from=0ps to=35840ps
```

Figure 52: Average current through global VDD source for 4 cycles of two full array write operations measured as $3.524581 \cdot 10^{-5}$ A, or approximately 35.2 μA

Given this average current, we can multiply by our 1V rail to find power consumption for this write operation to be approximately 35.2μW.

4.3 Optimizations

Throughout the design of our 64-bit SRAM, we incorporated several key optimizations across the circuit hierarchy to improve area efficiency, delay performance, and energy consumption. These choices were informed by careful architectural planning, logical minimization, and sensitivity to clocking behavior across different operating modes of the SRAM.

A major optimization was in the design of the 4-to-16 row decoder. While a naive implementation would use sixteen NAND4 gates (each comprising 8 transistors) followed by sixteen inverters (2 transistors each), totaling 160 transistors, we adopted a modular strategy using shared NAND2 and NOR2 gates. Specifically, our design instantiates 8 NAND2 gates and 16 NOR2 gates, each of which uses 4 transistors. This results in a total of $8 \times 4 + 16 \times 4 = 96$ transistors—an over 40% reduction in transistor count compared to the NAND4 approach. Beyond area savings, this modular structure allowed us to reuse intermediate signals across multiple outputs, which simplified routing and balanced delays across the decoder output network. It also preserved regularity in layout and ensured consistent logical effort across paths.

In the bitline precharge design, we used a single shared active-low enable signal generated by a NAND2 gate fed by $\overline{\text{WE}}$ and clk_b. This ensured that precharge only occurred when write operations were disabled, and during the appropriate clock phase, reducing unnecessary switching and thereby minimizing dynamic power dissipation.

We also opted to implement our sense amplifier with strong positive feedback and cross-coupled inverters, enabling fast differential sensing of small voltage swings on the bitlines. Each sense amplifier is clocked using clk_a, ensuring that sensing occurs only after bitline differential development is complete. Buffers at the output of the sense amp ensured signal integrity without significantly impacting the delay.

On the write path, we employed tristate buffers and tristate inverters controlled by the WE signal. This allowed precise control over when data is actively driven onto the bitlines and when the write drivers remain in high-impedance mode. This approach prevents contention between precharge and write circuitry and eliminates short-circuit current paths, thereby improving energy efficiency.

Clocking was a central part of our optimization strategy. A two-phase non-overlapping clocking scheme using clk_a and clk_b ensured that precharge, read, and write phases occurred sequentially without temporal overlap. This avoided contention on bitlines, enabled correct sense amplifier triggering, and minimized spurious transitions that would increase switching power.

Finally, careful sizing of transistors was employed throughout the design to ensure sufficient noise margins and drive strength where needed (e.g., for access transistors and pull-up/down networks in the SRAM cell), without over-provisioning in paths where switching activity or capacitive load was minimal. This transistor sizing balance ensured both area and energy optimization without compromising reliability or performance.

Together, these optimizations yielded a highly compact, low-power SRAM array with consistent and predictable behavior across all operating modes. The design choices reflect a balance between theoretical rigor and practical considerations for physical implementation.

4.4 Design Metrics Table

Metric	**Description**	**Value**
Delay	worst case read/write delay	137.5ps
Power	avg. energy consumed by 4 cycles of writing 1s and 0s	35.2uW
Bitcell Area	area of 1 SRAM cell	$6 \cdot WL$
FOM	figure of merit	$1.16 \cdot 10^{-37}$

Table 2: FOM Summary Table

$$FOM = 60 \cdot BitcellArea \cdot Power \cdot Delay^2 \tag{1}$$

$$FOM = 60 \cdot 6(22 \cdot 10^{-9})^2 \cdot 35.2 \cdot 10^{-6} \cdot (137.5 \cdot 10^{-12})^2 = 1.159 \cdot 10^{-37} \tag{2}$$

5 Statement of Academic Integrity

We, Krishna Karthikeya Chemudupati and Morgan Wang, certify that we have complied with the University of Pennsylvania's Code of Academic Integrity in completing this project.