SRAM 16x8 Synchronous Memory

Nitzan Chen
Tel Aviv University
September 2025

1. Introduction & System Overview

1.1 Background

Static Random Access Memory (SRAM) is one of the most critical due to its high speed, low latency, and reliability. SRAM is widely used in CPU caches, register files, embedded buffers, and high-performance digital logic blocks. Unlike Dynamic RAM (DRAM), which requires periodic refresh operations, SRAM stores information in bistable latches and retains data if power is applied.

This project focuses on the design, implementation, and verification of a synchronous 16×8 SRAM block, modeled in Verilog RTL (Register Transfer Level). Although small in scale (128 bits total capacity), this SRAM module demonstrates the essential principles of memory design, including memory cell design, decoding logic, peripheral circuits, synchronous timing, and verification methodology.

1.2 Why 16x8 SRAM?

The 16x8 configuration provides 128 bits of storage. It is intentionally chosen to be small enough for complete design, verification, and documentation within a student project, yet large enough to illustrate all the key architectural blocks: 6T cells, row and column decoders, write drivers, and sense amplifiers.

1.3 Objectives

The main objectives of this project are:

- Implement an RTL level synchronous SRAM.
- Model realistic support logic (decoders, drivers, sense amps).
- Verify correctness using a self-checking testbench.
- Document design methodology and results in a professional format.

1.4 Tool and Workflow

Tools used include Icarus Verilog for simulation, GTKWave for waveform analysis, and Microsoft Word for documentation. The workflow consisted of:
1. Writing RTL code for SRAM and peripheral blocks.
2. Developing a testbench for functional verification.
3. Running simulations and analyzing results.
4. Preparing industrial-style documentation with diagrams.

2. SRAM Implementation Overview

A 16x8 synchronous SRAM is built from the following components, each playing a specific role:

2.1 SRAM Cell (6T)

The fundamental storage element. Each cell stores a single bit using six transistors (two cross-coupled inverters for data retention, plus two access transistors controlled by the WL). Modes include:

- Hold: WL=0, latch isolated.
- Read: WL=1, BL/BL_NOT precharged, cell discharges one side.
- Write: WL=1, write driver forces BL/ BL_NOT.

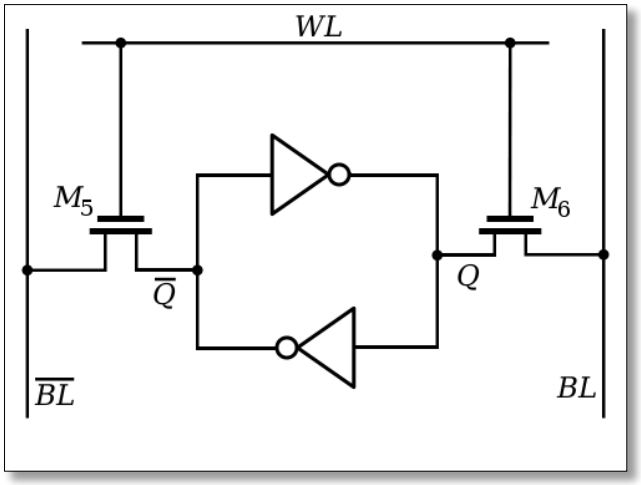

Figure 1: 6T SRAM Cell

2.2 Row Decoder (4-to-16)

Takes a 4-bit row address and activates exactly one of 16 WL. This ensures that only one row of memory cells is selected at a time.

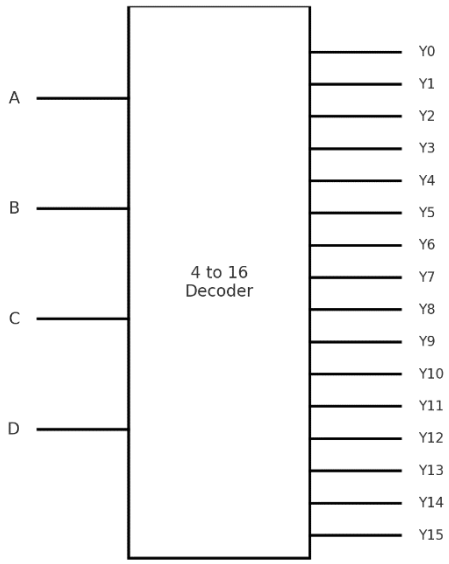

Figure 2: 4-to-16 Row Decoder

A	B	C	D	Output
0	0	0	0	Y0
0	0	0	1	Y1
0	0	1	0	Y2
0	0	1	1	Y3
0	1	0	0	Y4
0	1	0	1	Y5
0	1	1	0	Y6
0	1	1	1	Y7
1	0	0	0	Y8
1	0	0	1	Y9
1	0	1	0	Y10
1	0	1	1	Y11
1	1	0	0	Y12
1	1	0	1	Y13
1	1	1	0	Y14
1	1	1	1	Y15

Figure 3: 4-to-16 Row Decoder TT

2.3 Column Decoder (3-to-8)

Takes a 3-bit column address and selects one of 8 columns (BL pairs). Together with the row decoder, it isolates a single memory cell.

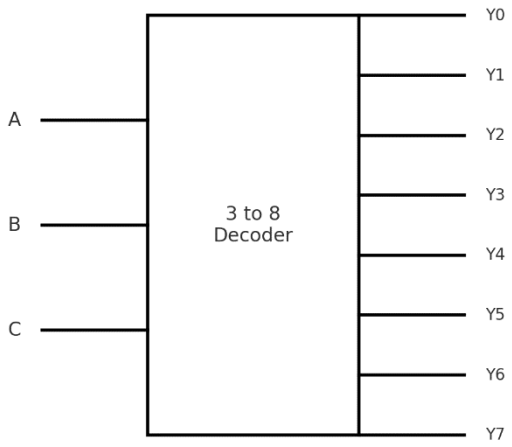

Figure 4: 3-to-8 Column Decoder

A	B	C	Output
0	0	0	Y0
0	0	1	Y1
0	1	0	Y2
0	1	1	Y3
1	0	0	Y4
1	0	1	Y5
1	1	0	Y6
1	1	1	Y7

Figure 5: 3-to-8 Column Decoder TT

2.4 Write Driver

The write driver forces the data value onto the BL during a write. It must be stronger than the latch so that it can flip the stored bit reliably. If din=1→ BL=Vdd and BL_NOT=0. if din=0 → BL=0 and BL_NOT=Vdd.
It is enabled only when we_n=0.

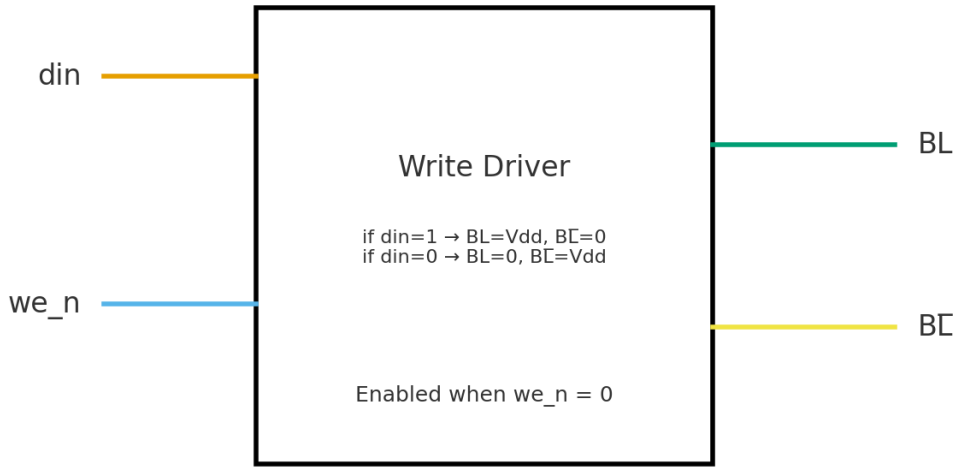

Figure 6: Write Driver

2.5 Sense Amplifier

During a read, the sense amplifier detects the small voltage difference between BL and BL_NOT and amplifies it to a full digital value. In silicon, latch-based sense amps are common. In RTL, this is abstracted into a synchronous read process.

2.6 Full SRAM Array (16x8)

Combines the row decoder, column decoder, write drivers, and sense amplifiers around the 16x8 grid of SRAM cells. This integration allows deterministic, synchronous read/write operations.

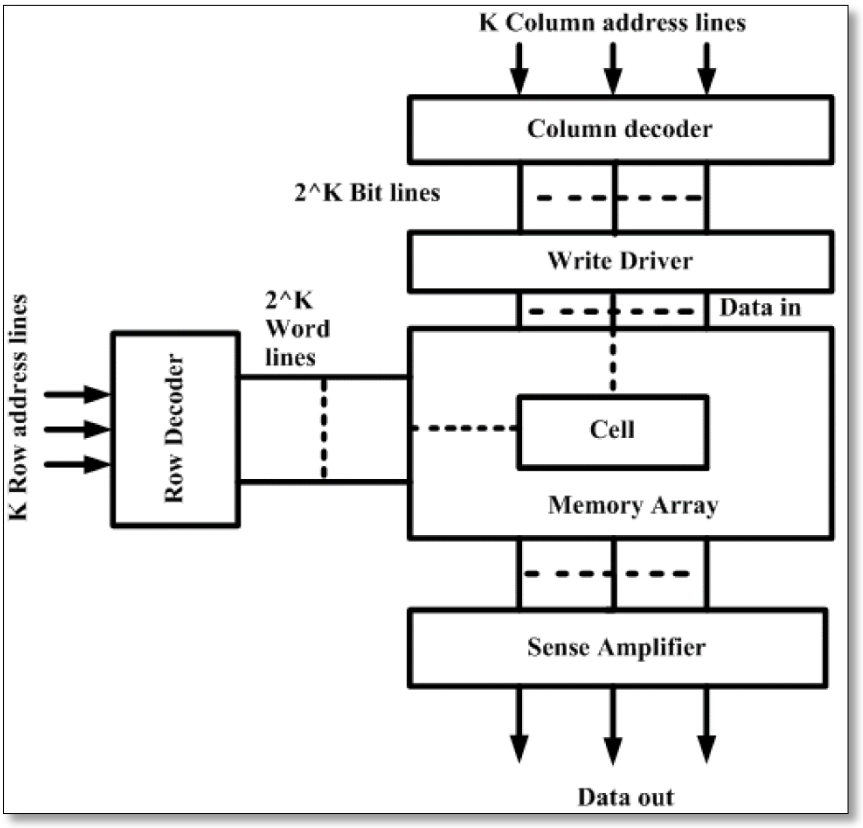

Figure 8: Full 16x8 SRAM Architecture

3. Verification Environment

3.1 Goals & Strategy

The verification goal is to prove that the RTL behaves like a synchronous, single-port SRAM:

- Functional correctness (every addressable bit stores/reads the right value)
- Overwrite integrity (the last write to a location is the value that persists).
- Timing consistency (1-cycle read latency; dout updates only on a rising clock edge; no mid-cycle glitches).

We use a self-checking testbench with deterministic tasks:

- do_reset – issues a clean, synchronous reset.
- write_bit(row,col,val) - performs a single-cycle write.
- read_bit(row,col,exp) - performs a synchronous read and checks dout against exp, calling $fatal on mismatch.
- A helper make_addr(row,col) packs {row[3:0], col[2:0]} into the 7-bit address.

3.2 Testbench Architecture

- **Clock**: 10 ns period, posedge is the only sampling/commit point.
- **Reset**: active-low rst_n asserted for two cycles; memory must come up all-zeros.
- **Write**:
    1. Drive addr and din on the negedge.
    2. Assert we_n=0 for exactly one rising edge (the write occurs at that posedge).
- **Read**:
    1. Drive addr on a negedge, keep we_n=1.
    2. On the next posedge, dout is compared to the expected value (#1 small settle before the check).

3.3 Phases

Phase 1 - Reset sweep

- Iterate all 16x8 locations and read 0.

Phase 2 - Checkerboard write

- Write (row ^ col) & 1 to every location.

Phase 3 - Read-back of checkerboard

- Read every location and check expected bit.

Phase 4 - Overwrite (last-write-wins)

- Repeatedly write to the **same** location: 0 → read 0 → 1 → read 1 → 0 → read 0.

Phase 5 - Timing consistency (1-cycle latency, no mid-cycle glitch)

- Choose two addresses **A** and **B** (with known expected values from the checkerboard).
- Sequence:
    1. Drive addr=A on a negedge, we_n=1.
    2. On the next posedge, dout must equal expA - this is the 1-cycle read latency.
    3. Mid-cycle, change addr to B (while clock is low/high but *before* the next posedge).
    4. Critically: dout must stay at expA until the next posedge.
    5. On the next posedge, dout updates to expB.

3.4 Timing Consistency

The design only assigns dout inside the @(posedge clk) process (read path), so:

- Before the rising edge, dout holds its previous value.
- At the rising edge, dout samples the selected memory bit.
- Therefore, changing addr mid-cycle cannot affect dout until the next posedge.

3.5 Overwrite Example

In a single-port synchronous memory, a write at posedge commits the new value to the addressed cell. Writing the same address on a later cycle must replace the old value.

The "Overwrite Example" figure shows a single address being:

- Written with 0 → immediately read back 0.
- Later written with 1 → read back 1.
- Later written with 0 again → read back 0.

3. Results & Conclusions

Simulation was run with Icarus Verilog and GTKWave. Console output shows successful initialization and checkerboard pattern verification.
Waveforms confirm that dout remains stable during a cycle and only updates on the following posedge

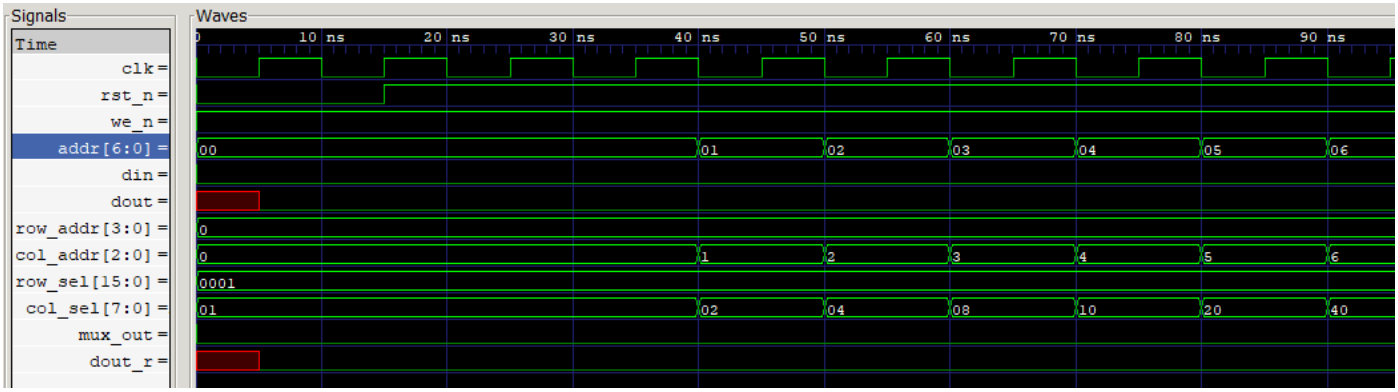

Figure 9: initialization following posedge

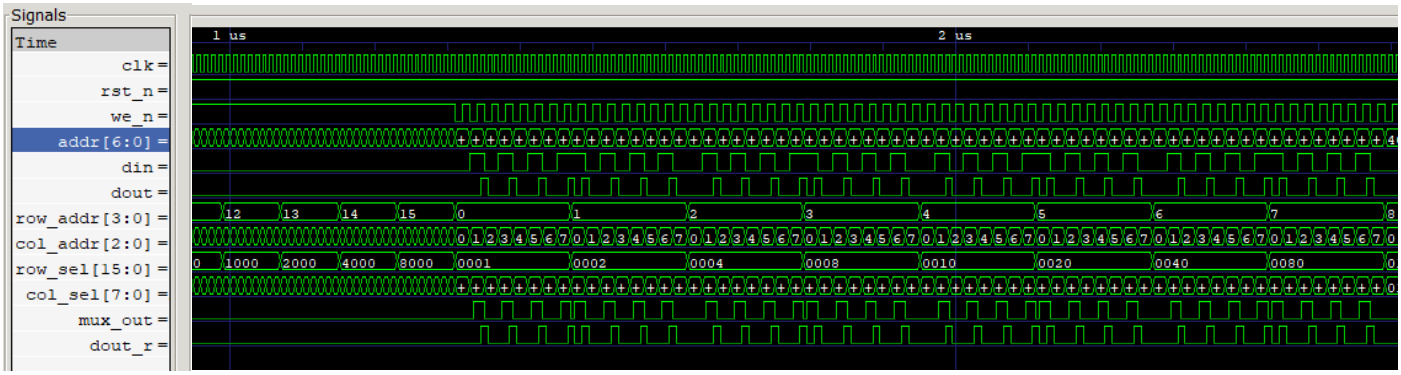

Figure 10: read & write operations following posedge

This project implements and verifies a 16x8 synchronous SRAM in Verilog RTL, including row/column decoders, write driver, and sense amplifier. A self-checking testbench validates reset, overwrite behavior, and timing consistency, demonstrating correct synchronous memory operation with one-cycle latency.

4. Appendices

Simulation commands:

iverilog -g2012 -o sram16x8_tb.vvp sram16x8.v row_decoder.v col_decoder.v mux8to1.v sense_amp.v write_driver.v sram16x8_tb.v

vvp sram16x8_tb.vvp

gtkwave sram16x8.vcd