VeriThoughts Model Evaluation Report: 4x4 SRAM Design

Design Prompt

I need to create a simple 4x4 SRAM in Verilog. Please generate the code for the following three modules.

Module A: `row_decoder_2to4`

- **Ports:**
  - `input wire [1:0] address`
  - `input wire enable`
  - `output wire [3:0] word_lines`
- **Functionality:** This module takes a 2-bit address and an enable signal. When enabled, it should decode the address and assert one of its four output word lines.

Module B: `sram_4x4`

- **Ports:**
  - `input wire [3:0] word_lines`
  - `input wire [3:0] bit_lines`
  - `input wire write_enable`
  - `output wire [3:0] data_out`
- **Functionality:** This is the memory core. It has a 4x4 register array to store 16 bits.
  - **Reading:** It should output the 4-bit data from the row selected by its `word_lines` input.
  - **Writing:** When its `write_enable` input is active, it should take 4 bits of data from its `bit_lines` input and store them in the row selected by the `word_lines`.
  - **Initialization:** Please pre-load the memory with the values `4'h1`, `4'h2`, `4'h4`, and `4'h8` in rows 0 through 3, respectively.

Module C: `sram_system`

- **Ports:**
  - `input wire [1:0] row_address`
  - `input wire enable`
  - `input wire write_enable`
  - `input wire [3:0] data_in`
  - `output wire [3:0] data_out`
  - `output wire [3:0] word_lines_debug`
- **Functionality:** This top-level module should connect the other two. It will instantiate the decoder and the memory core and wire them together correctly. It should also provide a debug output to see the state of the internal word lines.

Generated Code (Version A)

```
//=====================================================
// Top Module: sram_system
//=====================================================
module sram_system (
    input wire [1:0] row_address,
    input wire enable,
    input wire write_enable,
    input wire [3:0] data_in,
    output wire [3:0] data_out,
    output wire [3:0] word_lines_debug
);

    wire [3:0] word_lines;
    wire [3:0] bit_lines;
    wire [3:0] read_mux_out;
    wire [3:0] write_mux_out;

    // Instantiate the 2-to-4 row decoder
    row_decoder_2to4 row_decoder (
        .address(row_address),
        .enable(enable),
        .word_lines(word_lines)
    );

    // Instantiate the 4x4 SRAM array
    sram_4x4 sram (
        .word_lines(word_lines),
        .bit_lines(bit_lines),
        .write_enable(write_enable),
        .data_out(data_out)
    );

    // Debug output
    assign word_lines_debug = word_lines;

    // Multiplexer placeholders (currently unused)
    assign read_mux_out = {bit_lines[3], bit_lines[2], bit_lines[1],
bit_lines[0]};
    assign write_mux_out = {bit_lines[3], bit_lines[2], bit_lines[1],
bit_lines[0]};

endmodule

//=====================================================
// Module: row_decoder_2to4
//=====================================================
module row_decoder_2to4 (
    input wire [1:0] address,
    input wire enable,
    output wire [3:0] word_lines
);

    assign word_lines[0] = enable & (address == 2'b00);
    assign word_lines[1] = enable & (address == 2'b01);
    assign word_lines[2] = enable & (address == 2'b10);
    assign word_lines[3] = enable & (address == 2'b11);

endmodule

//=====================================================
// Module: sram_4x4
```

```
//=====================================================
module sram_4x4 (
    input wire [3:0] word_lines,
    input wire [3:0] bit_lines,
    input wire write_enable,
    output wire [3:0] data_out
);

    reg [3:0] mem [0:3]; // 4x4 memory array

    // Initialize the memory with default values
    initial begin
        mem[0] = 4'h1;
        mem[1] = 4'h2;
        mem[2] = 4'h4;
        mem[3] = 4'h8;
    end

    // Read operation (incorrect indexing)
    assign data_out = mem[word_lines];

    // Write operation (asynchronous and incorrectly indexed)
    always @(*) begin
        if (write_enable) begin
            mem[word_lines] = bit_lines;
        end
    end

endmodule
```

Ground Truth Code (Version B)

```
module sram_system (
    input wire [1:0] row_address,    // 2-bit row address
    input wire enable,                // Enable signal
    input wire write_enable,          // Write enable
    input wire [3:0] data_in,        // Data input for write operations
    output wire [3:0] data_out,      // Data output for read operations
    output wire [3:0] word_lines_debug // Debug output to see word lines
);

    wire [3:0] word_lines;
    wire [3:0] bit_lines;

    // Instantiate row decoder
    row_decoder_2to4 row_decoder (
        .address(row_address),
        .enable(enable),
        .word_lines(word_lines)
    );

    // For simplicity, connect data_in directly to bit_lines
    assign bit_lines = data_in;

    // Instantiate SRAM array
    sram_4x4 sram_array (
        .word_lines(word_lines),
        .bit_lines(bit_lines),
        .write_enable(write_enable),
        .data_out(data_out)
```

```
    );

    // Debug output
    assign word_lines_debug = word_lines;

endmodule

module sram_4x4 (
    input wire [3:0] word_lines,     // Word lines from row decoder
    input wire [3:0] bit_lines,      // Bit lines for data input
    input wire write_enable,         // Write enable signal
    output wire [3:0] data_out       // Data output from selected row
);

    // 4x4 memory array - each cell stores 1 bit (16 bits total)
    reg [3:0] memory_array [3:0];

    initial begin
        memory_array[0] = 4'b0001;
        memory_array[1] = 4'b0010;
        memory_array[2] = 4'b0100;
        memory_array[3] = 4'b1000;
    end

    // Read operation - output data from selected row
    assign data_out = (word_lines[0]) ? memory_array[0] :
                      (word_lines[1]) ? memory_array[1] :
                      (word_lines[2]) ? memory_array[2] :
                      (word_lines[3]) ? memory_array[3] :
                      4'b0000; // Default when no word line is active

    // Write operation - write data to selected row
    always @(*) begin
        if (write_enable) begin
            case (word_lines)
                4'b0001: memory_array[0] = bit_lines;
                4'b0010: memory_array[1] = bit_lines;
                4'b0100: memory_array[2] = bit_lines;
                4'b1000: memory_array[3] = bit_lines;
                default: ; // No operation
            endcase
        end
    end

endmodule

module row_decoder_2to4 (
    input wire [1:0] address,        // 2-bit row address (a1, a0)
    input wire enable,               // Enable signal
    output wire [3:0] word_lines     // Word lines (w3 to w0)
);

    // Each word line is active when address matches its row AND enable is
high
    assign word_lines[0] = enable & (address == 2'b00);
    assign word_lines[1] = enable & (address == 2'b01);
    assign word_lines[2] = enable & (address == 2'b10);
    assign word_lines[3] = enable & (address == 2'b11);

endmodule
```

Error Analysis

Error 1: Incorrect Array Indexing (Critical)

Location: `sram_4x4` module

Problem Code (Generated):

```
assign data_out = mem[word_lines];
always @(*) begin
    if (write_enable) begin
        mem[word_lines] = bit_lines;
    end
end
```

Issue:

- `word_lines` is a **4-bit one-hot encoded signal** (e.g., `0001`, `0010`, `0100`, `1000`) representing which row is selected
- The generated code incorrectly uses this one-hot value directly as an array index
- This causes multiple critical failures:
    - `word_lines = 4'b0100` (decimal 4) → **out-of-bounds** access for 4-element array (valid indices: 0-3)
    - `word_lines = 4'b1000` (decimal 8) → **out-of-bounds** access
    - `word_lines = 4'b0000` (disabled) → unintentionally accesses row 0
    - `word_lines = 4'b0011` (multiple bits) → accesses row 3, not multiple rows

Impact: Simulation produces undefined (X) values; synthesized hardware would have unpredictable behavior.

Correct Implementation (Ground Truth):

```
assign data_out = (word_lines[0]) ? memory_array[0] :
                  (word_lines[1]) ? memory_array[1] :
                  (word_lines[2]) ? memory_array[2] :
                  (word_lines[3]) ? memory_array[3] :
                  4'b0000;

always @(*) begin
    if (write_enable) begin
        case (word_lines)
            4'b0001: memory_array[0] = bit_lines;
            4'b0010: memory_array[1] = bit_lines;
            4'b0100: memory_array[2] = bit_lines;
            4'b1000: memory_array[3] = bit_lines;
            default: ;
        endcase
    end
end
```

Error 2: Missing Data Path Connection (Critical)

Location: `sram_system` module

Problem Code (Generated):

```
wire [3:0] bit_lines;
// bit_lines is never assigned!
```

Issue:

- The `bit_lines` wire is declared but **never driven** by `data_in`
- The SRAM core attempts to write from `bit_lines`, resulting in writing undefined/X values
- User data input (`data_in`) never reaches the memory

Impact: All write operations fail to store actual data; memory is written with X/Z values.

Correct Implementation (Ground Truth):

```
assign bit_lines = data_in;
```

Error 3: Incorrect Enable Handling (Functional)

Location: `sram_4x4` module read/write logic

Issue: When `enable = 0`, the decoder outputs `word_lines = 4'b0000`. The generated code interprets this as:

Read Path:

```
assign data_out = mem[word_lines];  // word_lines=0 → reads mem[0]
```

- **Problem:** Still outputs data from row 0 when disabled
- **Expected:** Should output `4'b0000` or high-Z when no row is selected

Write Path:

```
if (write_enable) begin
    mem[word_lines] = bit_lines;  // word_lines=0 → writes mem[0]
end
```

- **Problem:** Still writes to row 0 when disabled
- **Expected:** Should perform no write operation when no row is selected

Impact: The enable signal is effectively ignored, allowing spurious reads/writes to row 0.

Correct Behavior (Ground Truth):

- Read returns `4'b0000` when no word line is active
- Write case statement only matches valid one-hot patterns; `default: ;` prevents writes when disabled

Error 4: Combinational Write Logic (Design Hazard)

Location: `sram_4x4` module write logic

Problem Code (Generated & Ground Truth):

```
always @(*) begin
    if (write_enable) begin
        mem[word_lines] = bit_lines;
    end
end
```

Issue:

- Uses `always @(*)` for memory writes, creating **asynchronous/combinational write semantics**
- No clock edge control
- Can produce glitches and hazards in simulation
- Poor synthesis results (most real SRAMs are synchronous)
- Combined with incorrect indexing (Error 1) and missing enable checks (Error 3), the effect is severely amplified

Note: This issue exists in both generated and ground truth code, but is more problematic in the generated version due to compounding errors.

Error 5: Dead Code / Unused Logic

Location: `sram_system` module

Problem Code (Generated):

```
wire [3:0] read_mux_out;
wire [3:0] write_mux_out;

assign read_mux_out = {bit_lines[3], bit_lines[2], bit_lines[1],
bit_lines[0]};
assign write_mux_out = {bit_lines[3], bit_lines[2], bit_lines[1],
bit_lines[0]};
```

Issue:

- Declares and assigns `read_mux_out` and `write_mux_out` but **never uses them**
- Misleading signal names suggest they're part of the data path
- Makes code harder to verify and understand
- Wastes synthesis resources (though typically optimized away)

Impact: Code confusion and maintainability issues; suggests incomplete understanding of the design requirements.

Correct Implementation: Remove unused signals (not present in ground truth).

Comparison Summary

Aspect	Generated Code (Version A)	Ground Truth (Version B)
Row Decoder	Correct	Correct
Memory Initialization	Correct	Correct
Array Indexing	Uses one-hot value directly	Decodes one-hot properly
Data Path	bit_lines unconnected	bit_lines = data_in
Enable Handling	Reads/writes row 0 when disabled	Prevents access when disabled
Code Quality	Contains unused logic	Clean, minimal

Severity Assessment

Critical Errors (Prevent Functional Operation):

1. Array indexing with one-hot encoding
2. Missing data_in to bit_lines connection
3. Incorrect enable signal handling