Project report

Name: Nandan U

USN: 4VM21EC057

CAN_ID: CAN_33844765

College Name: VIDYA VIKAS INSTITUTE OF ENGINEERING & TECHNOLOGY, MYSURU

Department: ECE

Date of Submission: 14\03\2025

Project: Design of an 8-bit magnitude comparator using Verilog/SystemVerilog/VHDL

Introduction

This report presents the RTL design and functional verification of an 8-bit magnitude comparator using Verilog. The 8-bit Magnitude Comparator is a digital circuit designed to compare two 8-bit binary numbers, A and B. It determines whether A is greater than, less than, or equal to B and provides three output signals (A_greater, A_less, A_equal) to indicate the result. This module is widely used in arithmetic operations, decision-making circuits, and control systems. Its simplicity and efficiency make it a fundamental component in digital design. The implementation was verified using EDA Playground with Aldec Riviera-PRO simulator.

Truth table of 8-Bit magnitude comparator

Below is the truth table for the 8-bit magnitude comparator:

Truth Table

A	B	A > B	A < B	A == B
00011011	00011011	0	0	1
11111111	00000000	1	0	0

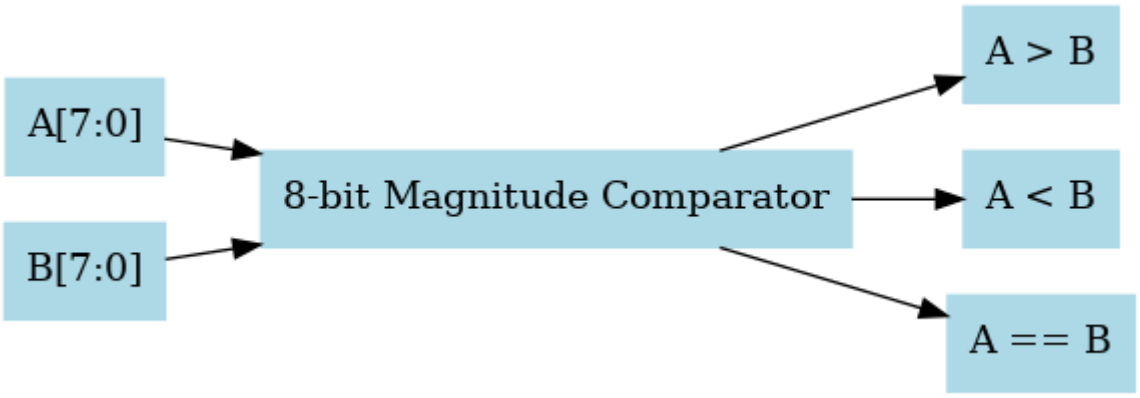
Design Architecture

The architecture consists of two 8-bit inputs (A and B) and three 1-bit outputs (A_greater, A_less, A_equal). The comparison logic is implemented using Verilog relational operators (>, <, ==). The outputs are assigned based on the comparison results, providing a clear

indication of the relationship between A and B. The design is combinational, ensuring fast and efficient operation without the need for a clock.

Block Diagram

Below is the block diagram of the 8-bit magnitude comparator:



RTL Code

```
  // Code your design here

module magnitude_comparator_8bit (
  input  [7:0] A, B,
  output A_greater, A_less, A_equal
);

  assign A_greater = (A > B) ? 1 : 0;
  assign A_less    = (A < B) ? 1 : 0;
  assign A_equal   = (A == B) ? 1 : 0;

endmodule
```

Testbench Code

```
  `timescale 1ns / 1ps

module tb_magnitude_comparator_8bit;

  // Inputs
  reg [7:0] A;
  reg [7:0] B;

  // Outputs
```

```
wire A_greater;
wire A_less;
wire A_equal;

// Instantiate the Unit Under Test (UUT)
magnitude_comparator_8bit uut (
   .A(A),
   .B(B),
   .A_greater(A_greater),
   .A_less(A_less),
   .A_equal(A_equal)
);

// Testbench logic
initial begin
   // Initialize inputs
   A = 8'b00000000;
   B = 8'b00000000;

   // Wait for global reset
   #10;

   // Test case 1: A > B
   A = 8'b00001111;
   B = 8'b00000011;
   #10;
   $display("Test Case 1: A = %b, B = %b, A_greater = %b, A_less = %b, A_equal = %b",
         A, B, A_greater, A_less, A_equal);

   // Test case 2: A < B
   A = 8'b00000011;
   B = 8'b00001111;
   #10;
   $display("Test Case 2: A = %b, B = %b, A_greater = %b, A_less = %b, A_equal = %b",
         A, B, A_greater, A_less, A_equal);

   // Test case 3: A == B
   A = 8'b10101010;
   B = 8'b10101010;
   #10;
   $display("Test Case 3: A = %b, B = %b, A_greater = %b, A_less = %b, A_equal = %b",
         A, B, A_greater, A_less, A_equal);
```

```
    // Test case 4: A > B (edge case)
    A = 8'b11111111;
    B = 8'b00000000;
    #10;
    $display("Test Case 4: A = %b, B = %b, A_greater = %b, A_less = %b, A_equal = %b",
        A, B, A_greater, A_less, A_equal);

    // Test case 5: A < B (edge case)
    A = 8'b00000000;
    B = 8'b11111111;
    #10;
    $display("Test Case 5: A = %b, B = %b, A_greater = %b, A_less = %b, A_equal = %b",
        A, B, A_greater, A_less, A_equal);

    // Test case 6: A == B (edge case)
    A = 8'b00000000;
    B = 8'b00000000;
    #10;
    $display("Test Case 6: A = %b, B = %b, A_greater = %b, A_less = %b, A_equal = %b",
        A, B, A_greater, A_less, A_equal);

    // End simulation
    $finish;
  end

endmodule
```

Simulation & Verification

Testbench Setup:

Inputs and Outputs:

A and B are 8-bit inputs to the comparator.

A_greater is 1 if A > B, otherwise 0.

A_less is 1 if A < B, otherwise 0.

A_equal is 1 if A == B, otherwise 0.

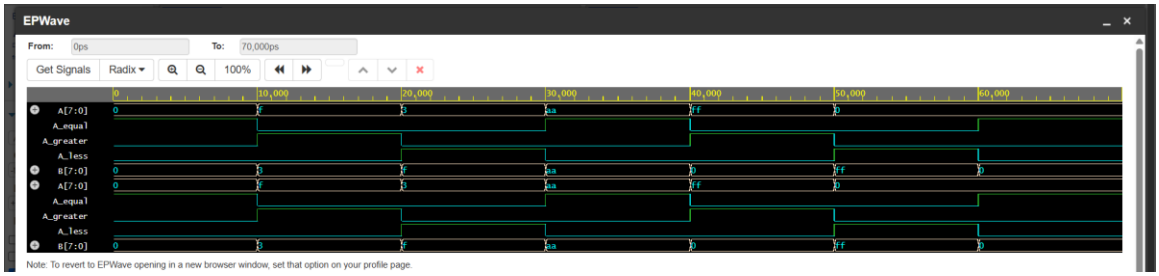
Simulation Results

Expected Output:

Test Case 1: A = 00001111, B = 00000011, A_greater = 1, A_less = 0, A_equal = 0
Test Case 2: A = 00000011, B = 00001111, A_greater = 0, A_less = 1, A_equal = 0
Test Case 3: A = 10101010, B = 10101010, A_greater = 0, A_less = 0, A_equal = 1
Test Case 4: A = 11111111, B = 00000000, A_greater = 1, A_less = 0, A_equal = 0
Test Case 5: A = 00000000, B = 11111111, A_greater = 0, A_less = 1, A_equal = 0
Test Case 6: A = 00000000, B = 00000000, A_greater = 0, A_less = 0, A_equal = 1

Simulated Input-Output Waveforms

Below are the simulated input-output waveforms:



Results and Discussion

The 8-bit magnitude comparator was successfully implemented and verified. The simulation results matched the expected behavior, confirming the correctness of the design.

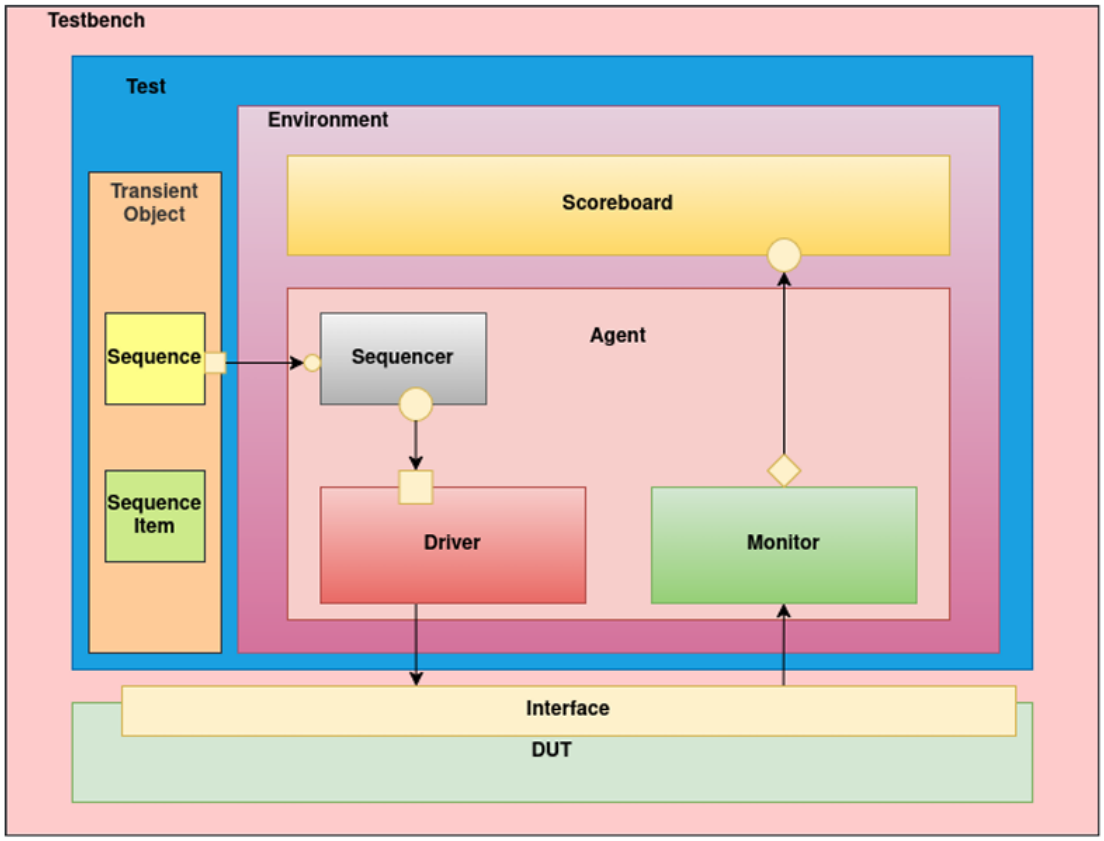
Block-Level Verification in UVM

Testbench Architecture (50%)

- Proper use of UVM components

- Adherence to the UVM factory and configuration mechanism.

- Proper use of virtual sequences and sequence layering if applicable.



Driver

```
class mag_comp_driver extends uvm_driver #(mag_comp_tx);
 `uvm_component_utils(mag_comp_driver)


  virtual mag_comp_if vif;
  mag_comp_tx tx;

  function new(string name="mag_comp_driver", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
   tx=mag_comp_tx::type_id::create("tx",this);
    if (!uvm_config_db#(virtual mag_comp_if)::get(this, "", "vif",vif))
      `uvm_fatal("DRIVER", "Could not get interface handle")
  endfunction
```

```
  virtual task run_phase(uvm_phase phase);
    forever begin
      seq_item_port.get_next_item(tx);
      vif.A <= tx.A;
      vif.B <= tx.B;
      #10;
      seq_item_port.item_done();
    end
  endtask


endclass
```

Monitor

```
class mag_comp_monitor extends uvm_monitor;
 `uvm_component_utils(mag_comp_monitor)

  virtual mag_comp_if vif;
  mag_comp_tx tx;
  uvm_analysis_port#(mag_comp_tx) mon_ap;


  function new(string name="mag_comp_monitor", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    mon_ap = new("mon_ap", this);
    tx = mag_comp_tx::type_id::create("tx");
    if (!uvm_config_db#(virtual mag_comp_if)::get(this, "", "vif", vif))
      `uvm_fatal("MONITOR", "Could not get interface handle")
  endfunction

  task run_phase(uvm_phase phase);
    forever begin
      #10;
      tx.A = vif.A;
      tx.B = vif.B;
      tx.A_greater = vif.A_greater;
```

```
        tx.A_less = vif.A_less;
        tx.A_equal = vif.A_equal;
        mon_ap.write(tx);
      end
   endtask
endclass
```

Agent

```
class mag_comp_agent extends uvm_agent;
 `uvm_component_utils(mag_comp_agent)

   mag_comp_driver drv;
   mag_comp_monitor mon;
   uvm_sequencer#(mag_comp_tx) seqr;



  function new(string name="mag_comp_agent", uvm_component parent=null);
     super.new(name, parent);
   endfunction

   virtual function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      drv = mag_comp_driver::type_id::create("drv", this);
      mon = mag_comp_monitor::type_id::create("mon", this);
      seqr = uvm_sequencer#(mag_comp_tx)::type_id::create("seqr", this);
   endfunction

   virtual function void connect_phase(uvm_phase phase);
      drv.seq_item_port.connect(seqr.seq_item_export);
   endfunction

endclass
```

Transection

```
class mag_comp_tx extends uvm_sequence_item;
```

```
 `uvm_object_utils(mag_comp_tx)

  rand bit [7:0] A, B;
  bit A_greater, A_less, A_equal;

  function new(string name = "mag_comp_tx");
    super.new(name);
  endfunction

endclass
```

Test

```
class mag_comp_test extends uvm_test;
 `uvm_component_utils(mag_comp_test)

  mag_comp_env env;
  mag_comp_seq seq;


 function new(string name="mag_comp_test", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    env = mag_comp_env::type_id::create("env", this);
    seq = mag_comp_seq::type_id::create("seq",this);
  endfunction

  task run_phase(uvm_phase phase);
    // mag_comp_seq seq = mag_comp_seq::type_id::create("seq");
    phase.raise_objection(this);
    seq.start(env.agent.seqr);
    #100;
    phase.drop_objection(this);
  endtask

endclass
```

Testbench

```
   // Code your testbench here
// or browse Examples

import uvm_pkg::*;
`include "uvm_macros.svh"

`include "package.sv"

module tb_top;

   mag_comp_if vif();

   magnitude_comparator_8bit dut (
      .A(vif.A), .B(vif.B),
      .A_greater(vif.A_greater), .A_less(vif.A_less), .A_equal(vif.A_equal)
   );

   initial
    begin
     uvm_config_db#(virtual mag_comp_if)::set(null, "*", "vif", vif);
     run_test("mag_comp_test");
    end

endmodule
```

Stimulus Generation

- Development of constrained-random and directed test sequences.

- Use of UVM sequences and transaction-based stimulus generation.

- Ability to generate different corner cases and invalid scenarios.

- Parameterization and reuse of sequences.

Generator

```
class mag_comp_seq extends uvm_sequence #(mag_comp_tx);
   `uvm_object_utils(mag_comp_seq)

   function new(string name = "mag_comp_seq");
     super.new(name);
```

```
  endfunction

 task body();
    mag_comp_tx tx;
    repeat (10)
     begin
       tx = mag_comp_tx::type_id::create("tx");
       start_item(tx);
       assert(tx.randomize());
       finish_item(tx);
    end
  endtask

endclass
```

Interface

```
interface mag_comp_if;

  logic [7:0] A, B;
  logic A_greater, A_less, A_equal;

endinterface
```

Scoreboarding and Checking

- Implementation of functional and self-checking scoreboard.
- Use of predictive models and golden reference comparison.
- Effective use of UVM phases for checking.

Scoreboard

```
class mag_comp_scoreboard extends uvm_scoreboard;
 `uvm_component_utils(mag_comp_scoreboard)

  uvm_analysis_imp#(mag_comp_tx, mag_comp_scoreboard) sb_port;


  function new(string name="mag_comp_scoreboard", uvm_component parent=null);
    super.new(name, parent);
```

```
  endfunction

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    sb_port = new("sb_port", this);
  endfunction

 function void write(mag_comp_tx tx);
  bit exp_A_greater = (tx.A > tx.B);
  bit exp_A_less = (tx.A < tx.B);
  bit exp_A_equal = (tx.A == tx.B);

  if (tx.A_greater !== exp_A_greater || tx.A_less !== exp_A_less || tx.A_equal !== exp_A_equal)
   begin
    `uvm_error(get_type_name(), $sformatf("FAIL: Mismatch detected! A=%0d B=%0d
Expected (G,L,E)=(%b,%b,%b) Got (%b,%b,%b)",
      tx.A, tx.B, exp_A_greater, exp_A_less, exp_A_equal, tx.A_greater, tx.A_less, tx.A_equal))
   end
  else
   begin
   `uvm_info(get_type_name(), $sformatf("PASS: A=%0d B=%0d Expected
(G,L,E)=(%b,%b,%b) Matched (G,L,E)=(%b,%b,%b)",
      tx.A, tx.B, exp_A_greater, exp_A_less, exp_A_equal, tx.A_greater, tx.A_less, tx.A_equal),
UVM_MEDIUM)
   end
 endfunction

endclass
```

Debugging and Logs

- Effective use of UVM messaging and verbosity levels.

- Debugging skills and ability to interpret waveforms and logs.

- Error detection.

- Documentation of issues and resolutions.

Environment

```
class mag_comp_env extends uvm_env;
 `uvm_component_utils(mag_comp_env)
```

```
  mag_comp_agent agent;
  mag_comp_scoreboard sb;


 function new(string name="mag_comp_env", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    agent = mag_comp_agent::type_id::create("agent", this);
    sb = mag_comp_scoreboard::type_id::create("sb", this);
  endfunction

  virtual function void connect_phase(uvm_phase phase);
    agent.mon.mon_ap.connect(sb.sb_port);
  endfunction

endclass
```

Waveform (In Testbench)

```
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars();
  end
```

UVM Report

```
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :   23
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[TEST_DONE]     1
[UVM/RELNOTES]     1
[mag_comp_scoreboard]    20

$finish called from file "/apps/vcsmx/vcs/U-2023.03-SP2//etc/uvm-1.2/src/base/uvm_root.svh", line 527.
$finish at simulation time                 200
           V C S   S i m u l a t i o n   R e p o r t
Time: 200 ns
CPU Time:      0.640 seconds;       Data structure size:   0.2Mb
```

Code Quality and Best Practices

- Consistency in naming conventions and coding style.

- Use of parameterized and reusable components.

- Proper comments and documentation within the code.

- Efficient and optimized coding practices.

EDA Link: https://edaplayground.com/x/6VfZ

Generate GDS using OpenROAD tool

In this section, the layout of the RTL code has been generated using the OpenROAD software tool.

Technology/Platform utilized: asap7

Instructions of the config.mk

```
export DESIGN_NAME = magnitude_comparator_8bit
export PLATFORM    = asap7

export VERILOG_FILES =
$(DESIGN_HOME)/src/$(DESIGN_NAME)/magnitude_comparator_8bit.v
export SDC_FILE      = $(DESIGN_HOME)/$(PLATFORM)/$(DESIGN_NAME)/constraint.sdc
#export ABC_AREA     = 1
export ABC_SPEED     = 1
# Adders degrade GCD
export ADDER_MAP_FILE :=

#export CORE_UTILIZATION ?= 75
export DIE_AREA = 0 0 8.5 4.5
export CORE_AREA = 0.5 0.5 8 4
export PLACE_DENSITY_LB_ADDON = 0.9
export TNS_END_PERCENT        = 100
export CELL_PAD_IN_SITES_DETAIL_PLACEMENT = 2
export CELL_PAD_IN_SITES_GLOBAL_PLACEMENT = 4
#export REMOVE_CELLS_FOR_EQY   = TAPCELL*
#export PDN_TCL = $(DESIGN_HOME)/$(PLATFORM)/$(DESIGN_NAME)/my_pdn.tcl
```

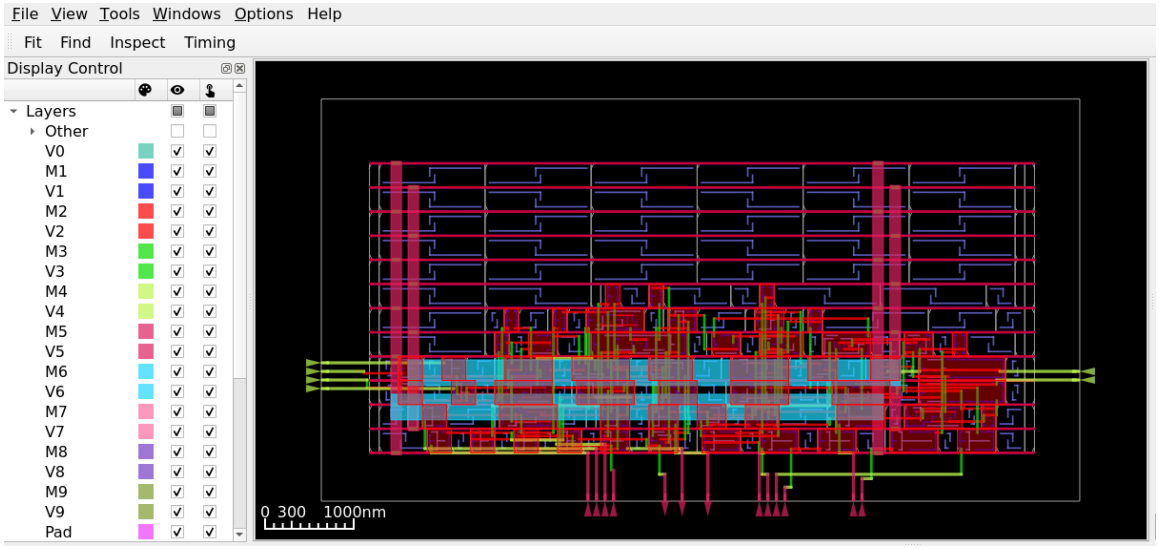
Instructions of the constraint.sdc

```
current_design magnitude_comparator_8bit
```

```
set clk_name  core_clock
set clk_port_name clk
set clk_period 2.5
set clk_io_pct 0.2

set clk_port [get_ports $clk_port_name]

create_clock -name $clk_name -period $clk_period $clk_port

set non_clock_inputs [lsearch -inline -all -not -exact [all_inputs] $clk_port]

set_input_delay  [expr $clk_period * $clk_io_pct] -clock $clk_name $non_clock_inputs
set_output_delay [expr $clk_period * $clk_io_pct] -clock $clk_name [all_outputs]
```

Layout of the Design

Below is the layout of the 8-bit magnitude comparator:



Performance Analysis

Power Measurement:

```
Group            Internal  Switching    Leakage      Total
                    Power     Power      Power      Power (Watts)
----------------------------------------------------------------
Sequential          0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
Combinational       4.83e-15   2.57e-15   4.83e-09   4.83e-09 100.0%
Clock               0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
```

Macro 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.0%
Pad 0.00e+00 0.00e+00 0.00e+00 0.00e+00 0.0%

Total 4.83e-15 2.57e-15 4.83e-09 4.83e-09 100.0%
0.0% 0.0% 100.0%

Area Measurement:
Design area 7 u^2 28% utilization.

Timing Information:

Startpoint: A[4] (input port)
Endpoint: A_less (output port)
Path Group: unconstrained
Path Type: max

Delay	Time	Description
0.00	0.00	v input external delay
0.00	0.00	v A[4] (in)
9.61	9.61	v input5/Y (BUFx12f_ASAP7_75t_R)
18.72	28.32	v _32_/Y (XOR2x1_ASAP7_75t_R)
33.87	62.20	v _36_/Y (OR4x1_ASAP7_75t_R)
8.85	71.04	^ _37_/Y (INVx1_ASAP7_75t_R)
28.04	99.08	^ _42_/Y (AND5x2_ASAP7_75t_R)
7.32	106.41	v _66_/Y (AOI21x1_ASAP7_75t_R)
12.08	118.49	v output19/Y (BUFx3_ASAP7_75t_R)
0.00	118.49	v A_less (out)
	118.49	data arrival time

(Path is unconstrained)

Clock frequency

Group	Slack

max slew

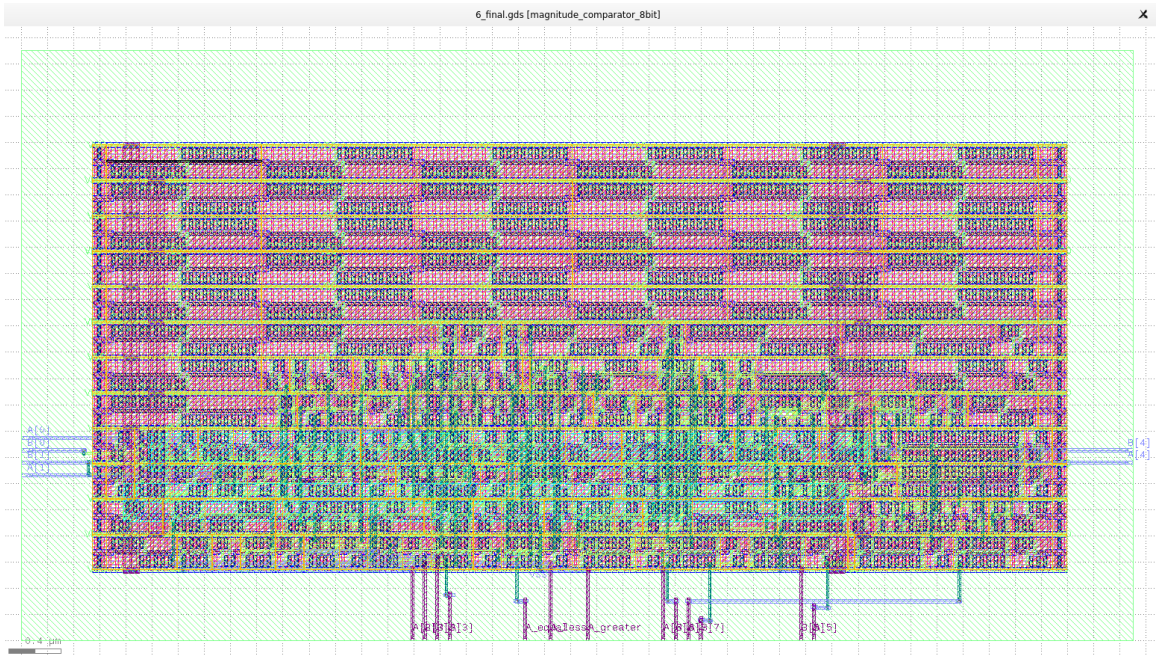
Pin	Limit	Slew	Slack
_65_/B	320.00	17.38	302.62 (MET)

max capacitance

Pin	Limit	Cap	Slack
_66_/Y	23.04	0.69	22.35 (MET)

Generated GDS

Below is the generated GDS file:



Conclusions

In this report, the RTL code of 8-bit magnitude comparator has been designed in Verilog. The code is successfully verified with the UVM with 100% test case pass. The design code is further processed in the OpenROAD tool to generate its GDS using the asap7 platform. There is no setup and hold violations.