Project report

Project-6 : Design of an 8x1 MUX using Verilog/System Verilog/VHDL

Name : Shreya Dadasaheb Yamakanmarde

USN : 2HN21EC038

Can_id : CAN_33488373

College Name : Hirasugar institute of technology, Nidasoshi-591236

Department : Electronics and communication Engineering

Date of Submission : 20/03/2025

1. Introduction

This report presents the RTL design and functional verification of an 8x1 MUX using Verilog. The 8x1 Multiplexer (8-bit) is a digital circuit that selects one of eight 8-bit input signals (D0 to D7) based on a 3-bit select signal (S). The selected input is routed to the 8-bit output (Y). This module is widely used in data routing, signal selection, and resource sharing in digital systems. Its flexibility and efficiency make it a key component in processors, communication systems, and other hardware designs. The implementation was verified using EDA Playground with Aldec Riviera-PRO simulator.

Truth table of 8x1 MUX

Below is the truth table for the 8x1 MUX:

Truth Table for 8-bit 8-to-1 MUX

Select (S)	Y
000	D0
001	D1
010	D2
011	D3
100	D4
101	D5
110	D6
111	D7

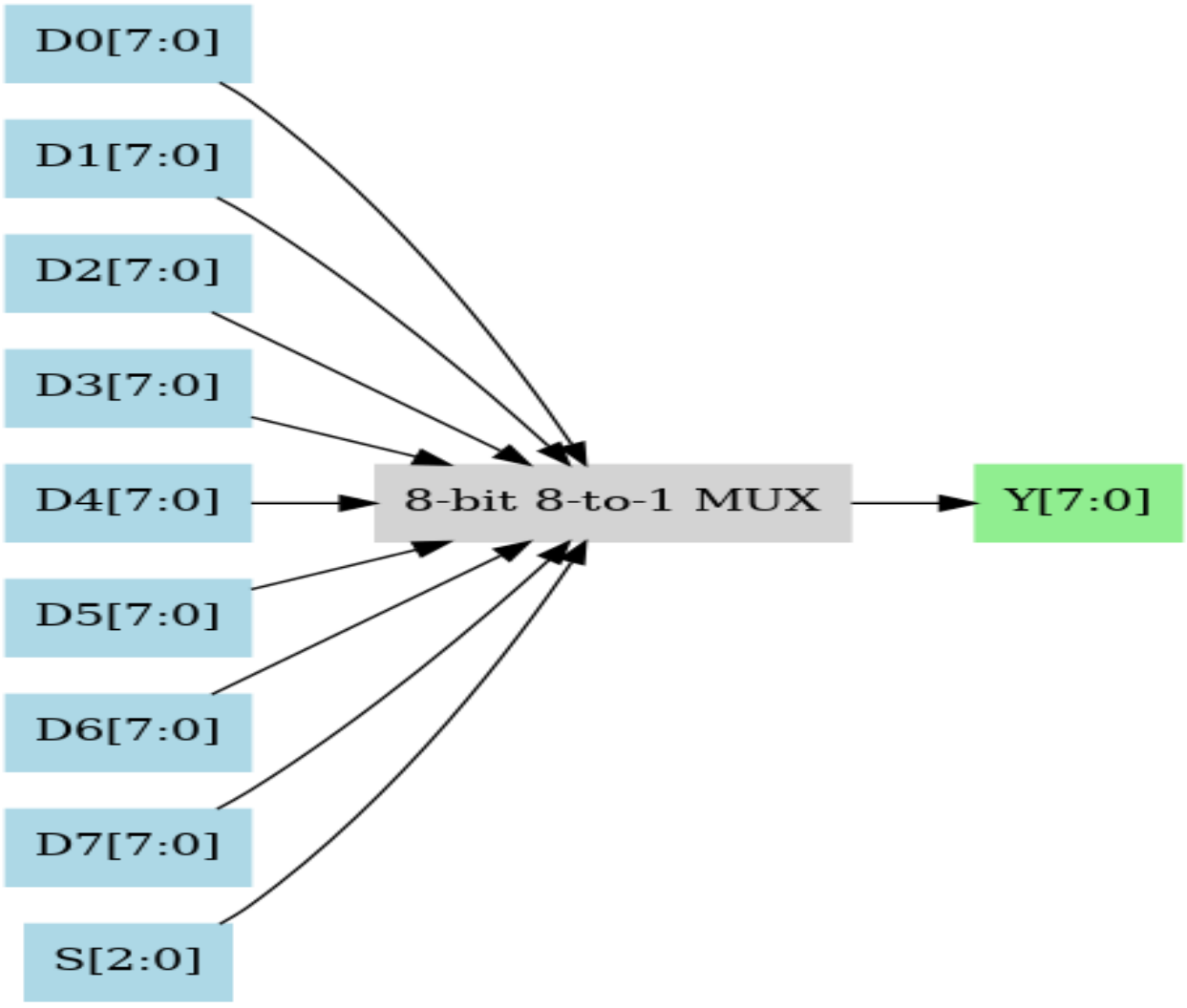
2. Specifications

Design Architecture

- The architecture consists of eight 8-bit inputs (**D0** to **D7**), a 3-bit select signal (**S**), and an 8-bit output (**Y**).
- A case statement maps each value of **S** to the corresponding input, which is then assigned to **Y**.
- A default case ensures that **Y** is set to **8'b00000000** for undefined states.
- The design is combinational, ensuring immediate output response to changes in the select signal.

Block Diagram

Below is the block diagram of the 8x1 MUX:



3. RTL Code

```
module mux8x1_8bit (
input  [7:0] D0, D1, D2, D3, D4, D5, D6, D7,  // 8-bit inputs
input  [2:0] S,  // 3-bit select
output reg [7:0] Y  // 8-bit output
);
always @(*) begin
   case (S)
      3'b000: Y = D0;
      3'b001: Y = D1;
      3'b010: Y = D2;
      3'b011: Y = D3;
      3'b100: Y = D4;
      3'b101: Y = D5;
      3'b110: Y = D6;
      3'b111: Y = D7;
      default: Y = 8'b00000000;
   endcase
end
endmodule
```

4. Testbench Code

```
`timescale 1ns / 1ps

module tb_mux8x1_8bit;

// Inputs
reg [7:0] D0, D1, D2, D3, D4, D5, D6, D7;
reg [2:0] S;

// Outputs
wire [7:0] Y;

// Instantiate the Unit Under Test (UUT)
mux8x1_8bit uut (
   .D0(D0),
   .D1(D1),
   .D2(D2),
   .D3(D3),
   .D4(D4),
   .D5(D5),
```

```
    .D6(D6),
    .D7(D7),
    .S(S),
    .Y(Y)
);

// Testbench logic
initial begin
    // Initialize inputs
    D0 = 8'b00000000;
    D1 = 8'b00000001;
    D2 = 8'b00000010;
    D3 = 8'b00000011;
    D4 = 8'b00000100;
    D5 = 8'b00000101;
    D6 = 8'b00000110;
    D7 = 8'b00000111;
    S = 3'b000;

    // Wait for global reset
    #10;

    // Test case 1: Select D0
    S = 3'b000;
    #10;
    $display("Test Case 1: S = %b, Y = %b", S, Y);

    // Test case 2: Select D1
    S = 3'b001;
    #10;
    $display("Test Case 2: S = %b, Y = %b", S, Y);

    // Test case 3: Select D2
    S = 3'b010;
    #10;
    $display("Test Case 3: S = %b, Y = %b", S, Y);

    // Test case 4: Select D3
    S = 3'b011;
    #10;
    $display("Test Case 4: S = %b, Y = %b", S, Y);

    // Test case 5: Select D4
    S = 3'b100;
    #10;
    $display("Test Case 5: S = %b, Y = %b", S, Y);
```

```
    // Test case 6: Select D5
    S = 3'b101;
    #10;
    $display("Test Case 6: S = %b, Y = %b", S, Y);

    // Test case 7: Select D6
    S = 3'b110;
    #10;
    $display("Test Case 7: S = %b, Y = %b", S, Y);

    // Test case 8: Select D7
    S = 3'b111;
    #10;
    $display("Test Case 8: S = %b, Y = %b", S, Y);

    // End simulation
    $finish;
  end

endmodule
```

5. Simulation & Verification

Testbench Setup:

- **D0** to **D7** are the 8-bit input data lines.
- **S** is the 3-bit select signal.
- **Y** is the 8-bit output.
- Multiple test cases are used to verify functionality.

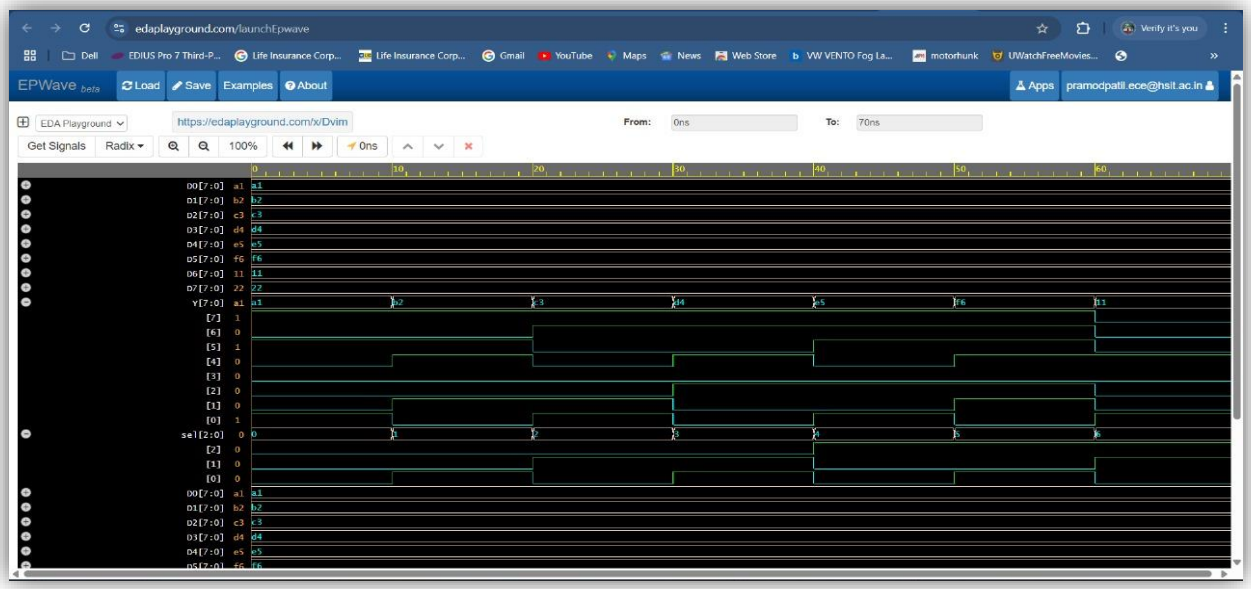
6. Simulation Results

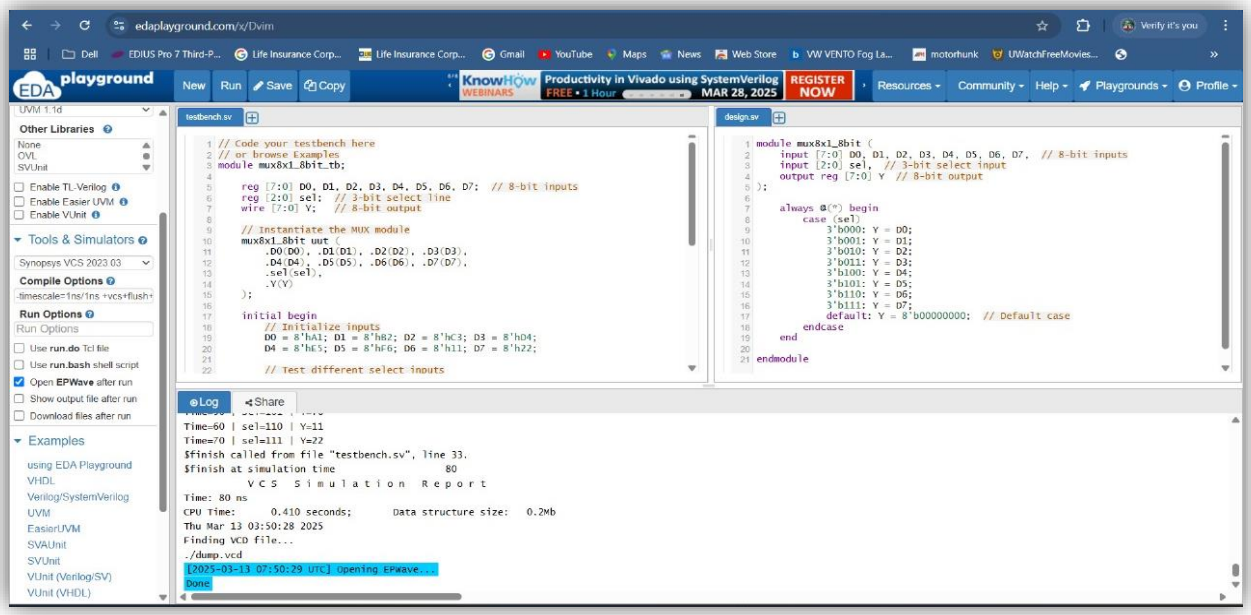
Expected Output:

```
Test Case 1:  S = 000,  Y = 00000000
Test Case 2:  S = 001,  Y = 00000001
Test Case 3:  S = 010,  Y = 00000010
Test Case 4:  S = 011,  Y = 00000011
Test Case 5:  S = 100,  Y = 00000100
Test Case 6:  S = 101,  Y = 00000101
Test Case 7:  S = 110,  Y = 00000110
Test Case 8:  S = 111,  Y = 00000111
```

Simulated Input-Output Waveforms

Below are the simulated input-output waveforms:





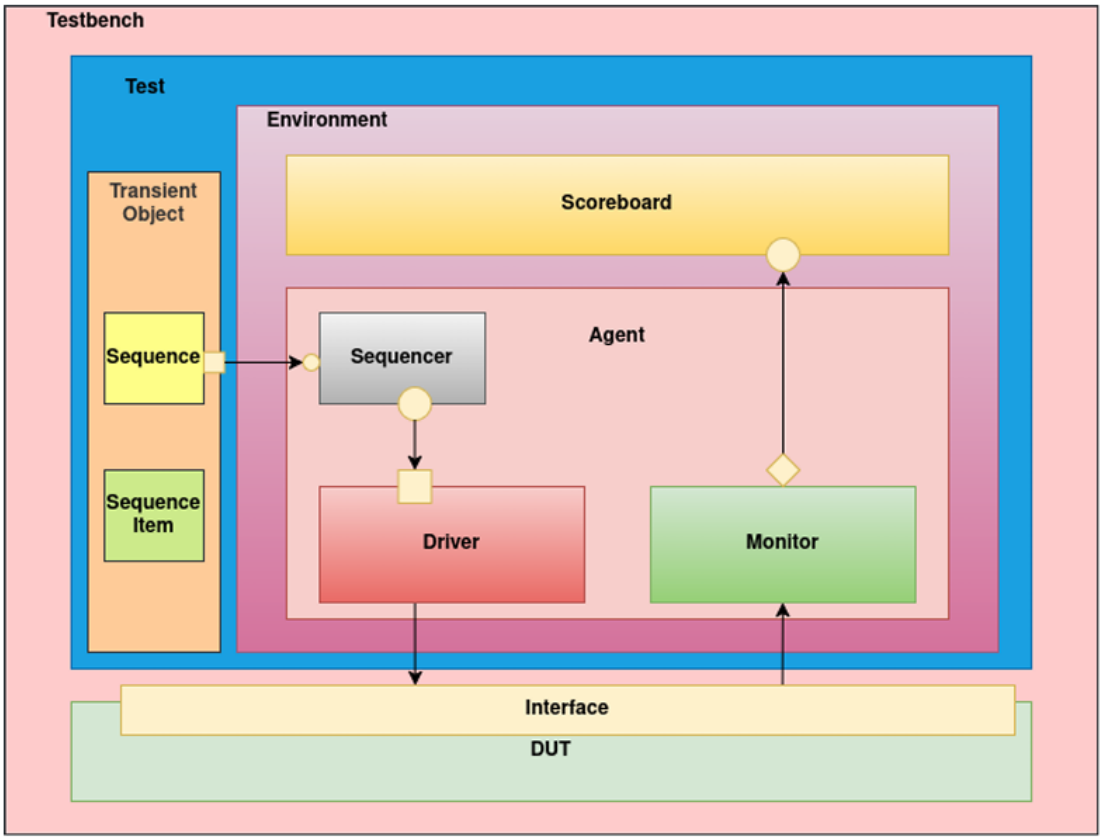
7. Results and Discussion

The 8x1 MUX was successfully implemented and verified. The simulation results matched the expected behavior, confirming the correctness of the design.

Evaluation Criteria for Block-Level Verification in UVM

1) 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 max_8x1driver extends uvm_driver #(max_8x1sequence_item);
`uvm_component_utils(max_8x1driver)

virtual max_8x1if vif;
max_8x1sequence_item item;

function new(string name="max_8x1driver", uvm_component parent);
  super.new(name, parent);
```

```
    `uvm_info("max_8x1driver", "Inside constructor of max_8x1driver", UVM_HIGH)
  endfunction : new

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    `uvm_info(get_name(), "Inside build phase", UVM_HIGH)

    if (!uvm_config_db#(virtual max_8x1if)::get(this, "*", "max_8x1vif", vif))
      `uvm_error(get_name(), "Failed to get Virtual IF from database.")
  endfunction : build_phase

  task run_phase(uvm_phase phase);
    super.run_phase(phase);
    `uvm_info(get_name(), "Inside run phase", UVM_HIGH)

    forever begin
      item = max_8x1sequence_item::type_id::create("item");
      seq_item_port.get_next_item(item);
      drive(item);
      seq_item_port.item_done();
    end
  endtask : run_phase

  task drive(max_8x1sequence_item item);
    `uvm_info(get_name(), "Drive...", UVM_HIGH)
    #1
    vif.D0 = item.D0;
    vif.D1 = item.D1;
    vif.D2 = item.D2;
    vif.D3 = item.D3;
    vif.D4 = item.D4;
    vif.D5 = item.D5;
    vif.D6 = item.D6;
    vif.D7 = item.D7;
    vif.S = item.S;
  endtask : drive

endclass : max_8x1driver
```

- **Monitor**

```
    class max_8x1monitor extends uvm_monitor;
  `uvm_component_utils(max_8x1monitor)

  virtual max_8x1if vif;
  max_8x1sequence_item item;
```

```
uvm_analysis_port #(max_8x1sequence_item) mon_port;

function new(string name="max_8x1monitor", uvm_component parent);
  super.new(name, parent);
  `uvm_info("max_8x1monitor", "Inside constructor of max_8x1monitor", UVM_HIGH)
endfunction : new

function void build_phase(uvm_phase phase);
  super.build_phase(phase);
  `uvm_info(get_name(), "Inside build phase", UVM_HIGH)
  mon_port = new("mon_port", this);

  if (!uvm_config_db#(virtual max_8x1if)::get(this, "*", "max_8x1vif", vif))
    `uvm_error(get_name(), "Failed to get Virtual IF from database.")
endfunction : build_phase

function void connect_phase(uvm_phase phase);
  super.connect_phase(phase);
  `uvm_info(get_name(), "Inside connect phase", UVM_HIGH)
endfunction : connect_phase

task run_phase(uvm_phase phase);
  super.run_phase(phase);
  `uvm_info(get_name(), "Inside run phase", UVM_HIGH)

  forever begin
    item = max_8x1sequence_item::type_id::create("item");
    sample(item);
    `uvm_info(get_name(), "Item received!!", UVM_HIGH)
    mon_port.write(item);
  end
endtask : run_phase

task sample(max_8x1sequence_item item);
  #1;
  item.D0 = vif.D0;
  item.D1 = vif.D1;
  item.D2 = vif.D2;
  item.D3 = vif.D3;
  item.D4 = vif.D4;
  item.D5 = vif.D5;
  item.D6 = vif.D6;
  item.D7 = vif.D7;
  item.S = vif.S;
  item.Y = vif.Y;
endtask : sample
```

endclass : max_8x1monitor

❖ **Agent**

```
class max_8x1agent extends uvm_agent;
`uvm_component_utils(max_8x1agent)
      max_8x1driver drv;
      max_8x1monitor mon;
      max_8x1sequencer seqr;

function new(string name="max_8x1agent",uvm_component parent);
   super.new(name,parent);
   `uvm_info("max_8x1agent", "Inside constructor of max_8x1agent", UVM_HIGH)
endfunction

function void build_phase(uvm_phase phase);
   super.build_phase(phase);
   `uvm_info(get_name(), "Inside build phase", UVM_HIGH)

      drv=max_8x1driver::type_id::create("drv",this);
      mon=max_8x1monitor::type_id::create("mon",this);
      seqr=max_8x1sequencer::type_id::create("seqr",this);

endfunction: build_phase

function void connect_phase(uvm_phase phase);
   super.connect_phase(phase);
   `uvm_info(get_name(), "Inside connect phase", UVM_HIGH)
   drv.seq_item_port.connect(seqr.seq_item_export);
endfunction

task run_phase(uvm_phase phase);
   super.run_phase(phase);
   `uvm_info(get_name(), "Inside run phase", UVM_HIGH)
endtask

endclass: max_8x1agent
```

❖ **Environment**

```
class max_8x1env extends uvm_env;
`uvm_component_utils(max_8x1env)

max_8x1agent agent;
max_8x1scoreboard scb;
```

```
  function new(string name = "max_8x1env", uvm_component parent = null);
    super.new(name, parent);
    `uvm_info("max_8x1env", "Inside constructor of max_8x1env", UVM_HIGH)
  endfunction

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    `uvm_info(get_name(), "Inside build phase", UVM_HIGH)
    agent = max_8x1agent::type_id::create("agent", this);
    scb = max_8x1scoreboard::type_id::create("scb", this);

  endfunction : build_phase

  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    `uvm_info(get_name(), "Inside connect phase", UVM_HIGH)
    agent.mon.mon_port.connect(scb.scb_port);

  endfunction : connect_phase

endclass : max_8x1env
```

❖ **Test**

```
 class max_8x1test extends uvm_test;
  `uvm_component_utils(max_8x1test)

       max_8x1env env;

  function new(string name="max_8x1test",uvm_component parent);
    super.new(name,parent);
    `uvm_info("max_8x1test", "Inside constructor of max_8x1test", UVM_HIGH)
  endfunction

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    `uvm_info(get_name(), "Inside build phase", UVM_HIGH)
        env=max_8x1env::type_id::create("env",this);
  endfunction: build_phase

  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    `uvm_info(get_name(), "Inside connect phase", UVM_HIGH)
  endfunction
```

```
endclass: max_8x1test

class max_8x1mul_test extends max_8x1test;
   `uvm_component_utils(max_8x1mul_test)

   max_8x1mul_seq mul_seq;

   function new(string name="max_8x1mul_test",uvm_component parent);
      super.new(name,parent);
      `uvm_info("max_8x1mul_test", "Inside constructor of max_8x1mul_test", UVM_HIGH)
   endfunction

   function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      `uvm_info(get_name(), "Inside build phase", UVM_HIGH)

   endfunction: build_phase

   function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      `uvm_info(get_name(), "Inside connect phase", UVM_HIGH)
   endfunction

   task run_phase(uvm_phase phase);
     super.run_phase(phase);
     `uvm_info(get_name(), "Inside run phase", UVM_HIGH)

     phase.raise_objection(this);

     repeat(`TEST_COUNT) begin
     // forever begin
       mul_seq=max_8x1mul_seq::type_id::create("mul_seq");
       mul_seq.start(env.agent.seqr);
     end
     wait(env.scb.test_cnt==`TEST_COUNT);

     phase.drop_objection(this);

   endtask

endclass: max_8x1mul_test
```

❖ Testbench

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

```
`define TEST_COUNT 200

`include "interface.sv"
`include "sequence_items.sv"
`include "sequencer.sv"
`include "sequence.sv"
`include "driver.sv"
`include "monitor.sv"
`include "scoreboard.sv"
`include "agent.sv"
`include "environment.sv"
`include "test.sv"
`timescale 1ns/1ns
`include "uvm_macros.svh"
//import uvm_pkg::*;

module top;

   // bit clk=0;
  //  bit rst;

    max_8x1if top_if();

 mux8x1_8bit dut(
      .D0       (top_if.D0),
      .D1       (top_if.D1),
      .D2       (top_if.D2),
      .D3       (top_if.D3),
      .D4       (top_if.D4),
      .D5       (top_if.D5),
      .D6       (top_if.D6),
      .D7       (top_if.D7),
      .S        (top_if.S),
      .Y        (top_if.Y)
                  );

//clock generation
   // initial forever #0.5 clk=~clk;

initial begin
  //  rst = 1;
  //  #2 rst =0;
  end

          initial begin
```

```
    uvm_config_db #(virtual max_8x1if) :: set(null,"*","max_8x1vif",top_if);
    `uvm_info("TOP","Configured database for interface...",UVM_LOW)
  end

  initial begin
    run_test("max_8x1test");
  end

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


 initial begin
  #10000000;
  $finish();
 end
endmodule
// TODO: Receiving items from monitor in scoreboard
```

2) Stimulus Generation(15%)

- 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.

❖ **Sequence Item**

```
  class max_8x1sequence_item extends uvm_sequence_item;
  randc logic [7:0] D0, D1, D2, D3, D4, D5, D6, D7; // 8-bit inputs
  randc logic  [2:0] S;  // 3-bit select
  logic [7:0] Y ; // 8-bit output

 function new(string name="max_8x1sequence_item");
  super.new(name);
 endfunction

endclass
```

❖ **Sequence**

```
  class max_8x1sequencer extends uvm_sequencer #(max_8x1sequence_item);
```

```
  `uvm_component_utils(max_8x1sequencer)
  function new(string name="max_8x1sequencer",uvm_component parent);
     super.new(name,parent);
    `uvm_info("max_8x1sequencer", "Inside constructor of max_8x1sequencer", UVM_HIGH)
  endfunction


  function void build_phase(uvm_phase phase);
     super.build_phase(phase);
     `uvm_info(get_name(), "Inside build phase", UVM_HIGH)
  endfunction: build_phase

  function void connect_phase(uvm_phase phase);
     super.connect_phase(phase);
     `uvm_info(get_name(), "Inside connect phase", UVM_HIGH)
  endfunction

endclass: max_8x1sequencer
```

3) Scoreboarding and Checking(25%)

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

❖ **Scoreboard**

```
  class max_8x1scoreboard extends uvm_scoreboard;
`uvm_component_utils(max_8x1scoreboard)

uvm_analysis_imp #(max_8x1sequence_item, max_8x1scoreboard) scb_port;

max_8x1sequence_item item[$];
max_8x1sequence_item s_item;
int test_cnt = 0;
int test_valid = 0;
int test_invalid = 0;

function new(string name = "max_8x1scoreboard", uvm_component parent);
  super.new(name, parent);
```

```
    `uvm_info("SCB_CLASS", "Inside Constructor!", UVM_HIGH)
  endfunction : new

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    `uvm_info("SCB_CLASS", "Build Phase!", UVM_HIGH)
    scb_port = new("scb_port", this);
  endfunction : build_phase

  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    `uvm_info("SCB_CLASS", "Connect Phase!", UVM_HIGH)
  endfunction : connect_phase

  function void write(max_8x1sequence_item rx_item);
    item.push_front(rx_item);
  endfunction : write

  task run_phase(uvm_phase phase);
    super.run_phase(phase);
    forever begin
      s_item = max_8x1sequence_item::type_id::create("s_item");
      wait(item.size() != 0);
      s_item = item.pop_front();
      compare(s_item);
      test_cnt++;
    end
  endtask : run_phase

  function void compare(max_8x1sequence_item item);
    logic [7:0] ex_res;

    ex_res = mux8x1_function(item.D0, item.D1, item.D2, item.D3, item.D4, item.D5, item.D6, item.D7,
item.S);
    if (ex_res == item.Y) begin
      `uvm_info(get_name(), $sformatf("[%0d/%0d] Test Passed", test_cnt, `TEST_COUNT),
UVM_HIGH);
      test_analysis(item, ex_res, 1);
      test_valid++;
    end else begin
      `uvm_error(get_name(), $sformatf("[%0d/%0d] Test failed", test_cnt, `TEST_COUNT));
```

```
      test_analysis(item, ex_res, 1);
      test_invalid++;
    end
  endfunction : compare

  function void test_analysis(max_8x1sequence_item item, logic [31:0] ex_res, bit flag);
    if (flag) begin
      $display("--------------------------------------------------------------------------------");
      $display("D0 = %0b, D1 = %0b, D2 = %0b, D3 = %0b, D4 = %0b, D5 = %0b, D6 = %0b, D7 = %0b,
S = %0b, Y = %0b, expected = %0b", item.D0, item.D1, item.D2, item.D3, item.D4, item.D5, item.D6,
item.D7, item.S, item.Y, ex_res);
    end
  endfunction : test_analysis

  function automatic [7:0] mux8x1_function(
    input [7:0] D0, D1, D2, D3, D4, D5, D6, D7,  // 8-bit data inputs
    input [2:0] S  // 3-bit select signal
  );
    case (S)
      3'b000: mux8x1_function = D0;
      3'b001: mux8x1_function = D1;
      3'b010: mux8x1_function = D2;
      3'b011: mux8x1_function = D3;
      3'b100: mux8x1_function = D4;
      3'b101: mux8x1_function = D5;
      3'b110: mux8x1_function = D6;
      3'b111: mux8x1_function = D7;
      default: mux8x1_function = 8'b00000000;
    endcase
  endfunction : mux8x1_function

  function void report_phase(uvm_phase phase);
    super.report_phase(phase);
    `uvm_info(get_name(), $sformatf("Total tests: %0d", test_cnt), UVM_LOW)
    `uvm_info(get_name(), $sformatf("Passed tests: %0d", test_valid), UVM_LOW)
    `uvm_info(get_name(), $sformatf("Failed tests: %0d", test_invalid), UVM_LOW)
  endfunction : report_phase

endclass : max_8x1scoreboard
```

4) Debugging and Logs(5%)

- Effective use of UVM messaging and verbosity levels.
- Debugging skills and ability to interpret waveforms and logs.
- Error detection.
- Documentation of issues and resolutions.

Waveform (In Testbench)

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

UVM Report

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

** Report counts by severity
UVM_INFO :    6
UVM_WARNING :    0
UVM_ERROR :    1
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[TEST_DONE]     1
[TOP]     1
[scb]     4
$finish called from file "/apps/vcsmx/vcs/U-2023.03-SP2//etc/uvm-1.1/src/base/uvm_root.svh", line 437.
$finish at simulation time                 200
          V C S   S i m u l a t i o n   R e p o r t
```

5) Code Quality and Best Practices(5%)

- 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/TZ7J

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: nangate45

Instructions of the config.mk

```
export DESIGN_NICKNAME = ripple_carry_adder_8bit
export DESIGN_NAME = ripple_carry_adder_8bit
export PLATFORM = gf180

export VERILOG_FILES = $(sort $(wildcard
$(DESIGN_HOME)/src/$(DESIGN_NICKNAME)/ripple_carry_adder_8bit.v))
export SDC_FILE = $(DESIGN_HOME)/$(PLATFORM)/$(DESIGN_NICKNAME)/constraint.sdc

export CORE_UTILIZATION = 0.5
export PLACE_DENSITY = 0.1
export TNS_END_PERCENT = 100
```

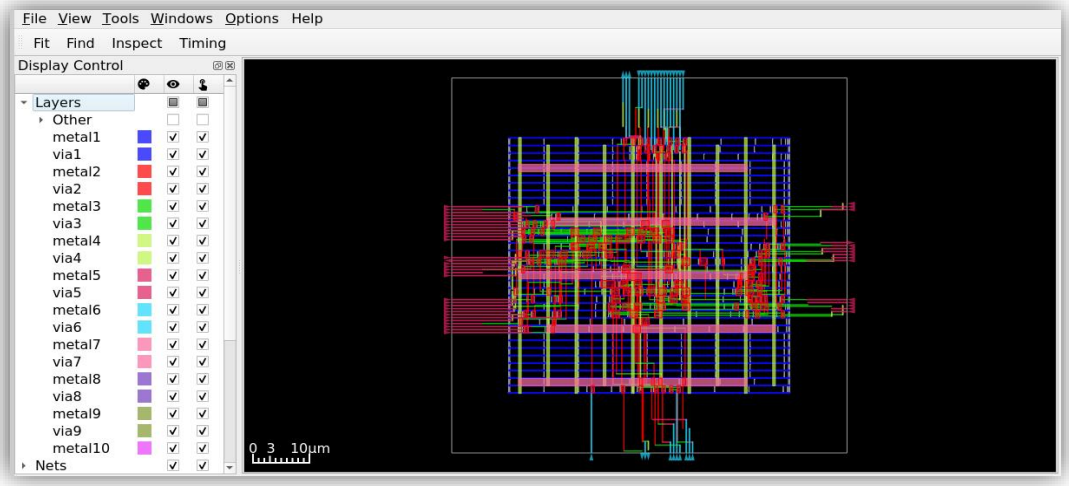
Instructions of the constraint.sdc

```
current_design ripple_carry_adder_8bit
set clk_name v_clk
set clk_period 2.5
set clk_io_pct 0.2
create_clock -name $clk_name -period $clk_period

set non_clock_inputs [lsearch -inline -all -not [all_inputs]]
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 8x1 MUX:



Performance Analysis

- **Power Measurement:**

Group	Internal Power	Switching Power	Leakage Power	Total Power (Watts)	
Sequential	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.0%
Combinational	5.88e-14	1.95e-14	4.58e-06	4.58e-06	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	5.88e-14	1.95e-14	4.58e-06	4.58e-06	100.0%
	0.0%	0.0%		100.0%	

- **Area Measurement:**

Design area 205 u^2 9% utilization.

- **Timing Information:**

Startpoint: S[0] (input port)
Endpoint: Y[5] (output port)
Path Group: unconstrained
Path Type: max

Delay	Time	Description
0.00	0.00	^ input external delay
0.00	0.00	^ S[0] (in)
0.02	0.02	^ _066_/Z (BUF_X8)
0.02	0.05	^ _089_/Z (BUF_X8)
0.06	0.10	v _119_/Z (MUX2_X1)
0.06	0.16	v _120_/Z (MUX2_X1)
0.06	0.21	v _121_/Z (MUX2_X1)
0.02	0.24	v output71/Z (BUF_X1)
0.00	0.24	v Y[5] (out)
	0.24	data arrival time

(Path is unconstrained)

- **Clock frequency:**

Group Slack
---

max slew

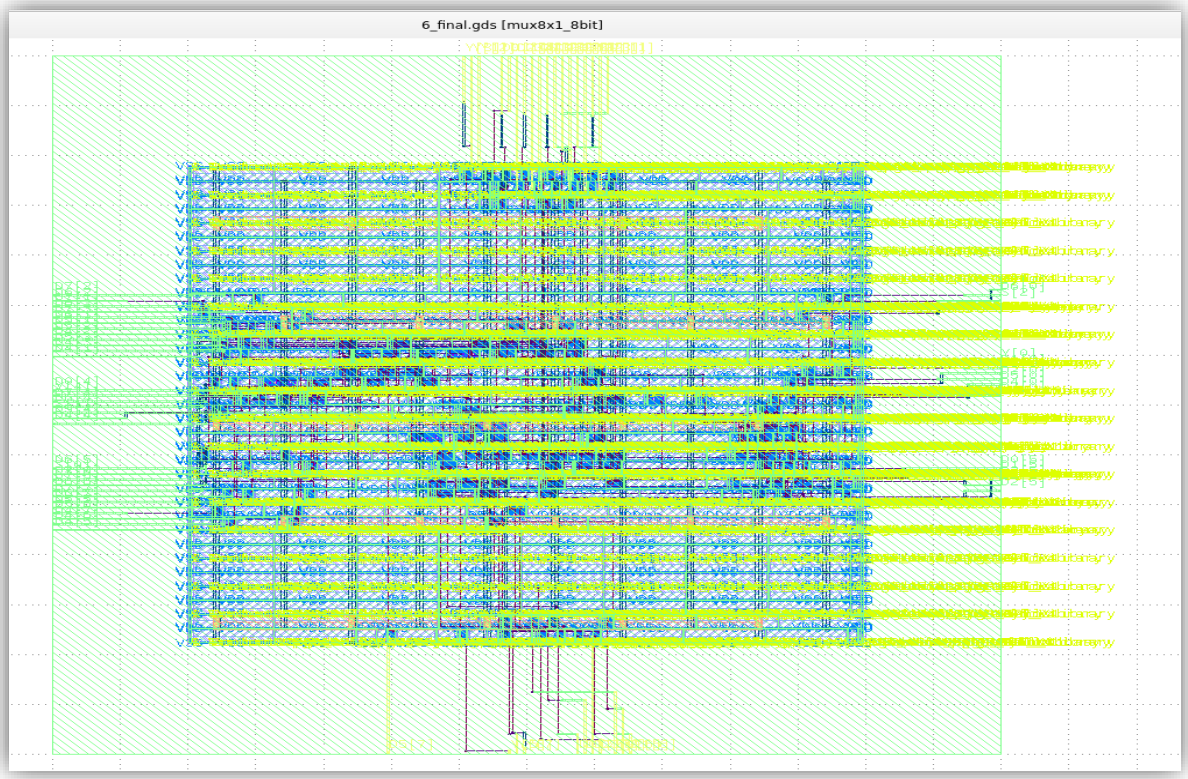
Pin	Limit	Slew	Slack
_083_/ZN	0.20	0.03	0.16 (MET)

max capacitance

Pin	Limit	Cap	Slack
_083_/ZN	22.16	1.62	20.54 (MET)

Generated GDS

Below is the generated GDS file:



Google Drive Link of Tar file:

https://drive.google.com/drive/folders/1qQoWmnCpCrWfMK32PI0jZaK6w5VdgUdn?usp=drive_link

Conclusions

In this report, the RTL code of 8x1 MUX 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 nangate45 platform. There is no setup and hold violations.