MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

LAB PROJECT REPORT

for

ECE2244: VLSI Design Lab

"I2C (Inter-Integrated Circuit) Controller"

Submitted by

ARNAV SHARMA
Reg No.: 230959070
Batch: A1

DEVJEET NANDA
Reg No.: 230959068
Batch: A1

DEPT.OF ELECTRONICS AND COMMUNICATION (ECE)

MANIPAL INSTITUTE OF TECHNOLOGY, MANIPAL-576104

28th March, 2025

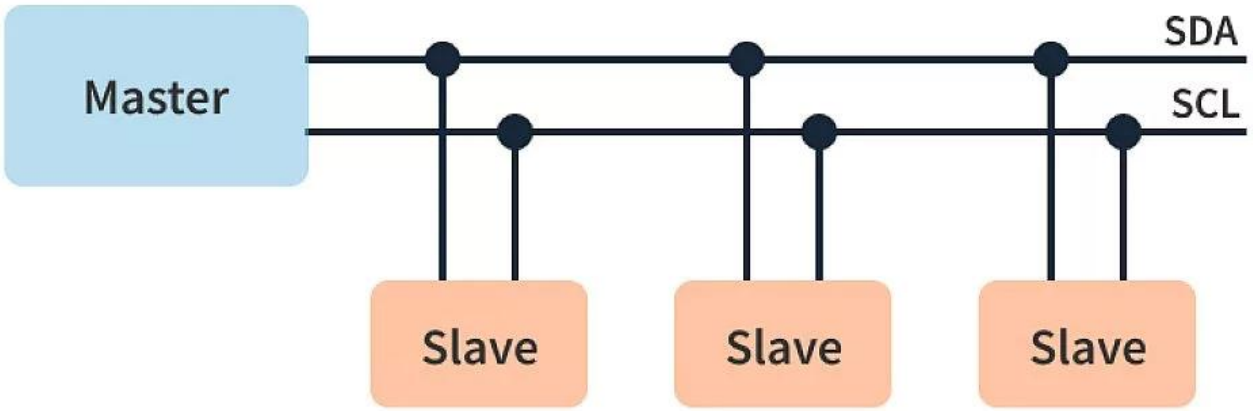
Index

OBJECTIVES:

- **This project focuses on the design, implementation, and verification of an I2C (Inter-Integrated Circuit) controller using Verilog in a VLSI environment.**
- **The objective is to develop a hardware description of an I2C master module, simulate its functionality using a testbench and analyze its performance through waveform inspection.**
- **Perform RTL synthesis and generate the synthesized design using Cadence Genus Synthesis Solution GUI.**
- **To create the physical design of the circuit using Cadence Innovus tool.**
- **To analyze area utilization, cell placement and planning and timing of the synthesized and designed circuit.**

PROJECT THEORY EXPLANATION:

I²C, or Inter-Integrated Circuit, is a widely used serial communication protocol designed for efficient, low-cost communication between microcontrollers and peripheral devices over short distances. It was developed by Philips Semiconductor (now NXP) in 1982 and is commonly referred to as "I-squared-C" or "I-two-C".



Key Features of I²C:

Two-Wire Communication: I²C uses only two lines:

- SDA (Serial Data Line): Transmits data.
- SCL (Serial Clock Line): Synchronizes data transmission.

Master-Slave Architecture: Typically, a single master (controller) initiates communication with one or more slave (peripheral) devices. However, it also supports multi-master configurations.

Addressing: Each device on the bus has a unique 7-bit or 10-bit address, allowing multiple devices to share the same bus without conflict.

Half-Duplex Communication: Data can flow in both directions but not simultaneously.

Clock Stretching: Slaves can hold the clock line low to delay communication if they need more processing time.

Arbitration: Built-in mechanisms prevent data corruption when multiple masters attempt to communicate simultaneously.

How I²C Works:

Start Condition: Communication begins when the master pulls SDA low while SCL remains high.

Address Frame: The master sends the address of the target device along with a read/write bit.

Acknowledgment (ACK): The addressed device responds with an ACK bit to confirm receipt.

Data Transmission: Data is sent in 8-bit frames, followed by an ACK/NACK bit after each frame.

Stop Condition: The master releases both lines (SDA and SCL go high), signalling the end of communication.

Advantages:

- Simplifies wiring with just two lines for multiple devices.
- Cost-effective and power-efficient.
- Supports various speeds, including standard mode (100 kbps), fast mode (400 kbps), and high-speed mode (3.4 Mbps).

Applications:

I²C is used in a wide range of applications, including:

- Embedded systems
- IoT devices
- Consumer electronics
- Automotive systems
- Industrial equipment.

I²C's simplicity, flexibility, and robust design make it a popular choice for short-distance communication in modern electronics.

Verilog Code:

The following code is for the I2C Master:

```
module i2c_master (
   input  wire      clk,       // System clock
   input  wire      rst_n,      // Active-low reset
   input  wire      start,      // Start signal for transmission
   input  wire [7:0] data_in,    // Data to transmit
   output reg       scl,       // I2C clock
   inout  wire      sda,       // I2C data (bidirectional)
   output reg       done       // Transaction complete
);

   reg [7:0] shift_reg;
   reg [3:0] bit_cnt;
   reg      sda_en;   // When asserted, drives SDA
   reg      sda_out;  // Value driven on SDA when enabled

   // Tri-state assignment: if sda_en is high, drive sda_out; else, high-Z.
   assign sda = sda_en ? sda_out : 1'bz;

   // Define states for our simple state machine.
   localparam IDLE       = 3'd0,
              START_STATE = 3'd1,
              BIT_LOW     = 3'd2,
              BIT_HIGH    = 3'd3,
              STOP_STATE  = 3'd4,
              DONE_STATE  = 3'd5;
   reg [2:0] state;

   always @(posedge clk or negedge rst_n) begin
      if (!rst_n) begin
         state    <= IDLE;
         scl      <= 1;
```

```
    sda_en    <= 0;
    sda_out   <= 1;
    bit_cnt   <= 0;
    shift_reg <= 8'd0;
    done      <= 0;
  end else begin
    case (state)
      IDLE: begin
        done   <= 0;
        scl    <= 1;
        sda_en <= 0;
        sda_out<= 1;
        bit_cnt<= 0;
        if (start) begin
          shift_reg <= data_in;
          state     <= START_STATE;
        end
      end

      START_STATE: begin
        // Start condition: while SCL is high, pull SDA low.
        sda_en  <= 1;
        sda_out <= 0;
        scl     <= 1;
        state   <= BIT_LOW;
      end

      BIT_LOW: begin
        // Drive SCL low and set SDA to the current bit.
        scl     <= 0;
        sda_out <= shift_reg[7 - bit_cnt];
        state   <= BIT_HIGH;
      end
```

```
            BIT_HIGH: begin
               // Drive SCL high so the slave can sample the bit.
               scl <= 1;
               // After SCL is high, move to next bit or finish if 8 bits sent.
               if (bit_cnt < 7) begin
                  bit_cnt <= bit_cnt + 1;
                  state   <= BIT_LOW;
               end else begin
                  state <= STOP_STATE;
               end
            end

            STOP_STATE: begin
               // Generate stop condition:
               // First, drive SCL low and SDA low; then release SDA while SCL goes high.
               scl    <= 0;
               sda_out <= 0;
               state   <= DONE_STATE;
            end

            DONE_STATE: begin
               scl    <= 1;
               sda_out <= 1;
               sda_en  <= 0;  // Release SDA
               done    <= 1;  // Signal that transmission is complete
               state   <= IDLE;
            end

            default: state <= IDLE;
         endcase
      end
   end
```

```
endmodule
```

The following code is for the I2C Slave:

```
module i2c_slave (
   input  wire      clk,        // System clock (not used for sampling here)
   input  wire      rst_n,       // Active-low reset
   inout  wire      sda,        // I2C data (bidirectional)
   input  wire      scl,        // I2C clock (driven by master)
   output reg [7:0]  data_out     // Received data
);

   reg [7:0] shift_reg;
   reg [3:0] bit_cnt;

   // The slave is not driving SDA in this example.
   assign sda = 1'bz;

   // Sample data on the falling edge of SCL.
   always @(negedge scl or negedge rst_n) begin
      if (!rst_n) begin
         bit_cnt   <= 0;
         shift_reg <= 0;
         data_out  <= 0;
      end else begin
         // Shift in the sampled bit.
         shift_reg <= {shift_reg[6:0], sda};
         bit_cnt <= bit_cnt + 1;
         if (bit_cnt == 3'd7) begin
            data_out <= {shift_reg[6:0], sda}; // After 8 bits, update data_out.
            bit_cnt <= 0;
            shift_reg <= 0;
         end
      end
```

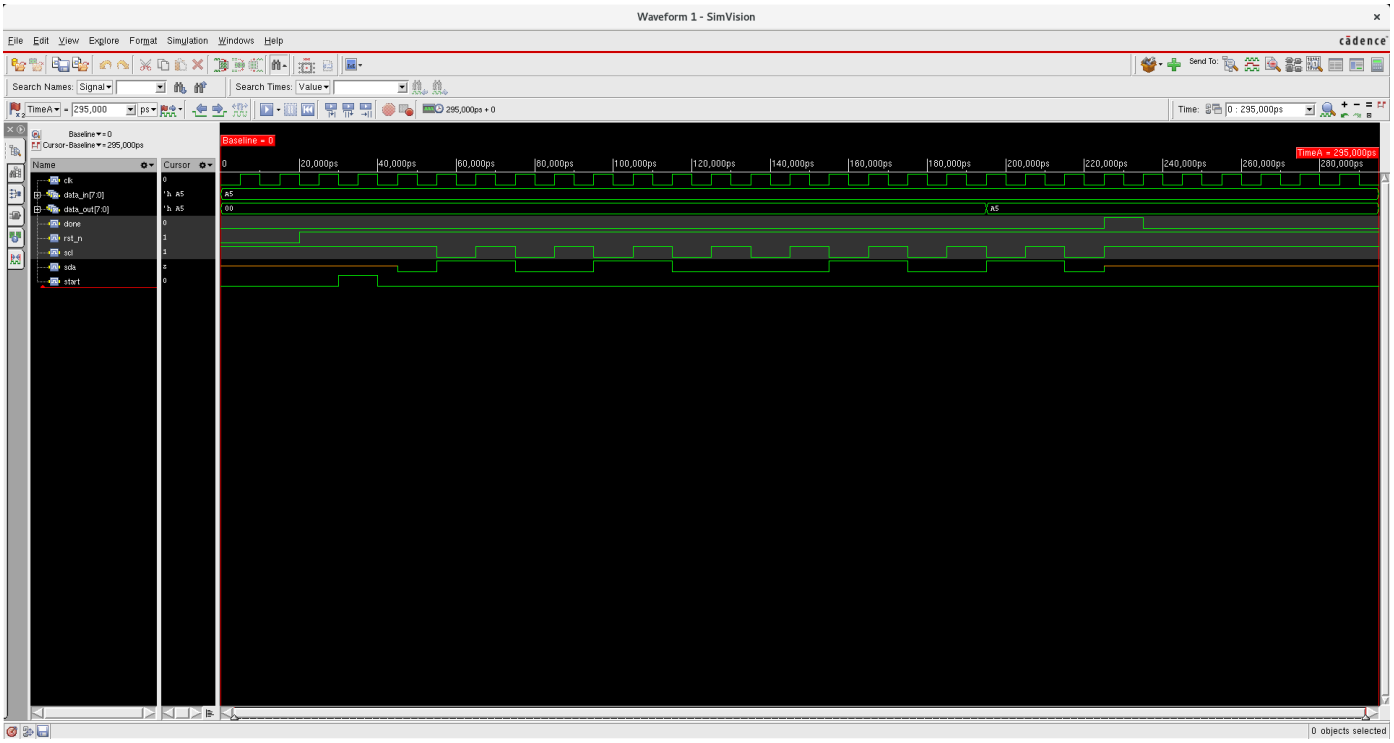
```
    end

endmodule
```

The following code is for the Test Bench:

```
`timescale 1ns/1ps
module tb_i2c;
    reg       clk;
    reg       rst_n;
    reg       start;
    reg  [7:0] data_in;
    wire      scl;
    wire      sda;
    wire      done;
    wire [7:0] data_out;

    // Instantiate I2C master.
    i2c_master master_inst (
        .clk(clk),
        .rst_n(rst_n),
        .start(start),
        .data_in(data_in),
        .scl(scl),
        .sda(sda),
        .done(done)
    );

    // Instantiate I2C slave.
    i2c_slave slave_inst (
        .clk(clk),
        .rst_n(rst_n),
        .sda(sda),
        .scl(scl),
```

```
        .data_out(data_out)
    );

    // Dump waveforms for viewing.
    initial begin
        $dumpfile("i2c_waveform.vcd");
        $dumpvars(0, tb_i2c);
    end

    // Generate 100 MHz clock (10 ns period).
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    // Reset and stimulus.
    initial begin
        rst_n   = 0;
        start   = 0;
        data_in = 8'hA5; // Example data
        #20;         // Hold reset for 20 ns
        rst_n = 1;
        #10;
        start = 1;    // Trigger the I2C transaction
        #10;
        start = 0;    // Remove the start signal
        wait(done);   // Wait for transaction to complete
        #20;
        $display("I2C Transaction complete. Slave received data: %h", data_out);
        #50;
        $finish;
    end
endmodule
```

Waveform:



Following is the code in the sdc file:

```
create_clock -name "clk" -period 2.0 -waveform {0.0 1.0} [get_ports clk]
set_clock_transition 0.1 [get_clocks clk]
set_clock_gating_check -setup 0.0
set_wire_load_mode "enclosed"
set_clock_uncertainty -setup 0.01 [get_ports clk]
set_clock_uncertainty -hold 0.01 [get_ports clk]
```

Power Report:

```
Instance: /i2c_top
Power Unit: W
PDB Frames: /stim#0/frame#0
  ------------------------------------------------------------------------
    Category         Leakage     Internal    Switching        Total    Row%
  ------------------------------------------------------------------------
      memory     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
    register     4.58422e-06  1.69530e-04  7.05556e-06  1.81170e-04  85.97%
       latch     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
       logic     1.01907e-06  1.31619e-05  5.27134e-06  1.94523e-05   9.23%
        bbox     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
       clock     0.00000e+00  0.00000e+00  1.01250e-05  1.01250e-05   4.80%
         pad     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
          pm     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
  ------------------------------------------------------------------------
    Subtotal     5.60329e-06  1.82692e-04  2.24519e-05  2.10747e-04 100.00%
  Percentage           2.66%       86.69%       10.65%      100.00% 100.00%
  ------------------------------------------------------------------------
```

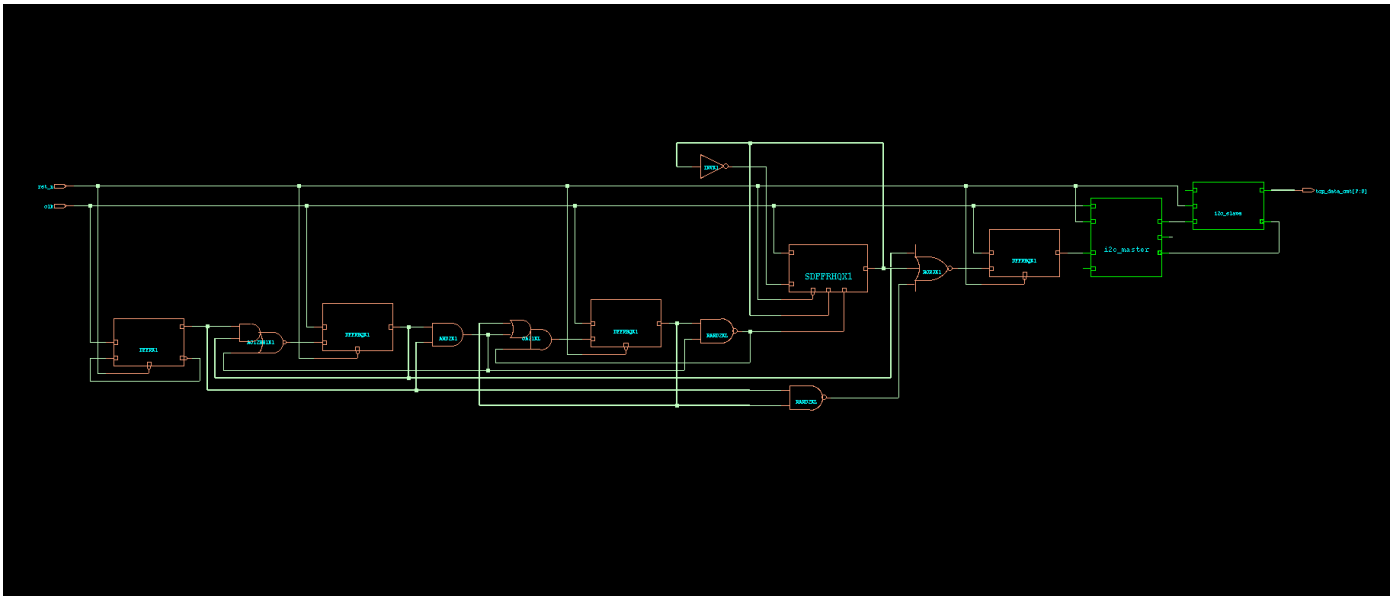
Timing Report:

```
Path 1: MET (707 ps) Setup Check with Pin master_inst/bit_cnt_reg[1]/CK->D
          Group: clk
     Startpoint: (R) master_inst/state_reg[1]/CK
          Clock: (R) clk
       Endpoint: (R) master_inst/bit_cnt_reg[1]/D
          Clock: (R) clk

                    Capture       Launch
       Clock Edge:+    2000            0
       Src Latency:+      0            0
       Net Latency:+      0 (I)        0 (I)
           Arrival:=    2000            0

              Setup:-     194
        Uncertainty:-      10
     Required Time:=    1796
      Launch Clock:-       0
         Data Path:-    1089
             Slack:=     707

#-------------------------------------------------------------------------------------------------------------
#        Timing Point          Flags    Arc    Edge   Cell     Fanout Load Trans Delay Arrival Instance
#                                                               (fF)  (ps)  (ps)   (ps)  Location
#-------------------------------------------------------------------------------------------------------------
  master_inst/state_reg[1]/CK  -       -      R     (arrival)     15    -   100     0       0    (-,-)
  master_inst/state_reg[1]/Q   -       CK->Q  R     DFFRX1         9 18.7   198   485     485    (-,-)
  master_inst/g828__6260/Y     -       A->Y   F     NAND2XL        5 12.7   326   269     754    (-,-)
  master_inst/g817__5122/Y     -       A1->Y  R     OAI32X1        1  2.8   217   214     968    (-,-)
  master_inst/g805__4319/Y     -       B0->Y  F     AOI31X1        1  1.7   129    50    1018    (-,-)
  master_inst/g796__2398/Y     -       B->Y   R     NAND2XL        1  1.7    71    71    1089    (-,-)
  master_inst/bit_cnt_reg[1]/D <<<     -      R     DFFRX1         1    -     -     0    1089    (-,-)
#-------------------------------------------------------------------------------------------------------------
```

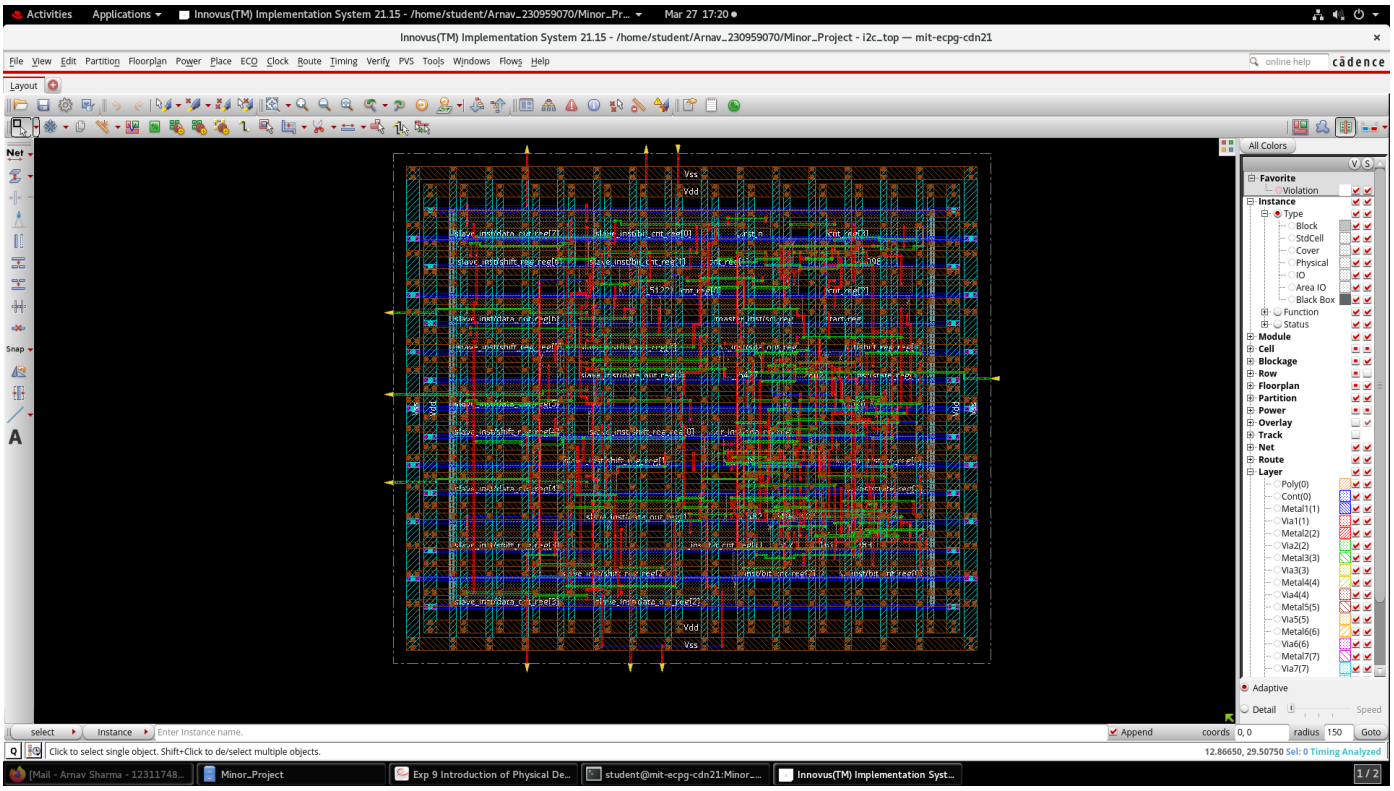
Area Report:

```
================================================================
  Generated by:           Genus(TM) Synthesis Solution 21.14-s082_1
  Generated on:           Mar 27 2025  04:20:20 pm
  Module:                 i2c_top
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
================================================================

   Instance     Module    Cell Count  Cell Area  Net Area   Total Area   Wireload
-------------------------------------------------------------------------------------
i2c_top                         86   1132.322     0.000     1132.322 <none> (D)
  master_inst i2c_master        45    376.936     0.000      376.936 <none> (D)
  slave_inst  i2c_slave         29    616.874     0.000      616.874 <none> (D)
  (D) = wireload is default in technology library
```

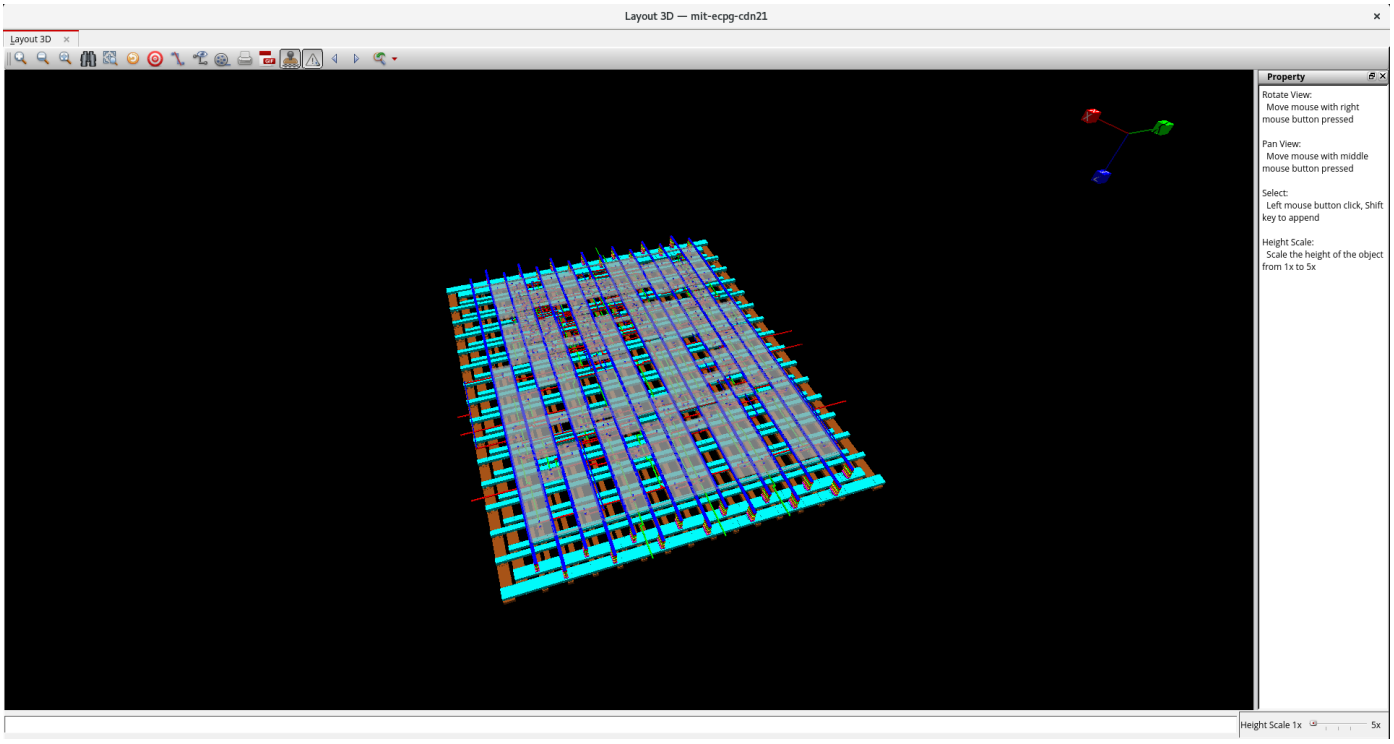
Synthesized circuit schematic:



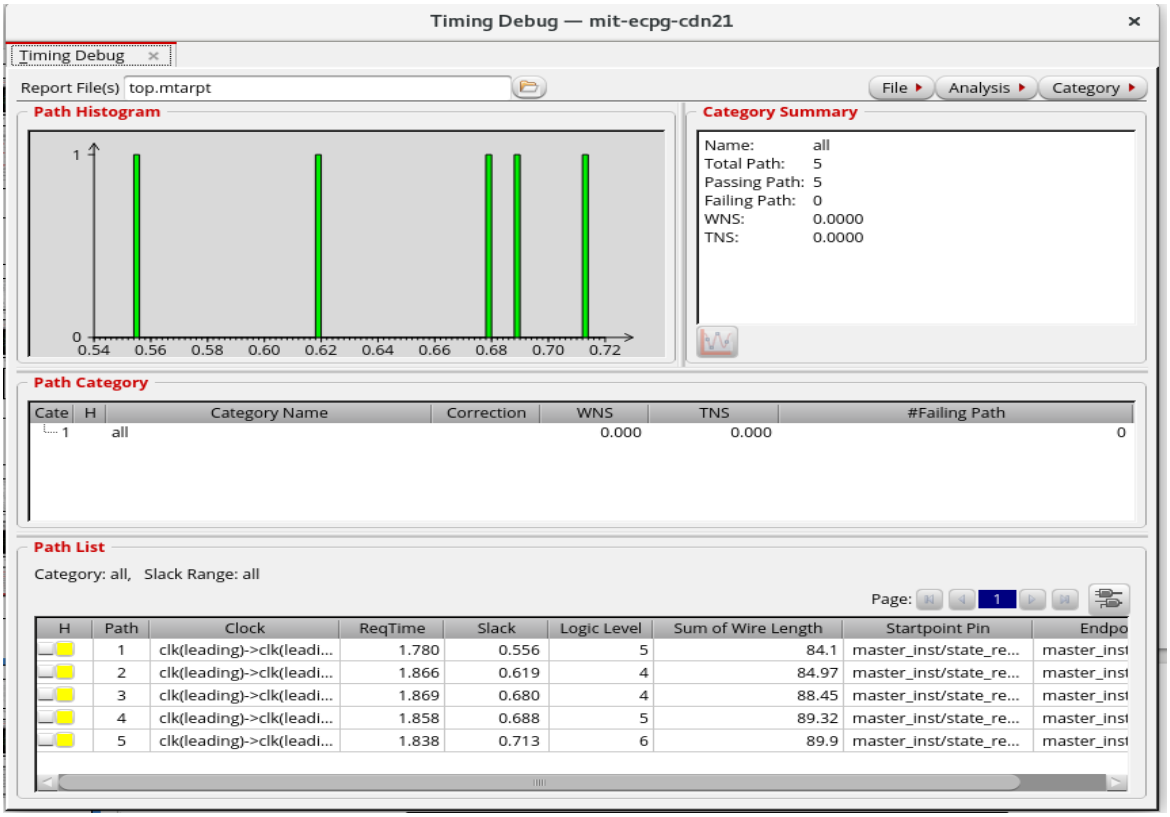
Layout:


Activities
Applications
Innovus(TM) Implementation System 21.15 - /home/student/Arnav_230959070/Minor_Pr...
Mar 27 17:20
Innovus(TM) Implementation System 21.15 - /home/student/Arnav_230959070/Minor_Project - i2c_top — mit-ecpg-cdn21
File View Edit Partition Floorplan Power Place ECO Clock Route Timing Verify PVS Tools Windows Flows Help
cadence
Layout
Net
Snap
All Colors
Favorite
Violation
Instance
Type
Block
StdCell
Cover
Physical
IO
Area IO
Black Box
Function
Status
Module
Cell
Blockage
Row
Floorplan
Partition
Power
Overlay
Track
Net
Route
Layer
Poly(0)
Cont(0)
Metal1(1)
Via1(1)
Metal2(2)
Via2(2)
Metal3(3)
Via3(3)
Metal4(4)
Via4(4)
Metal5(5)
Via5(5)
Metal6(6)
Via6(6)
Metal7(7)
Via7(7)
Adaptive
Detail
Speed
select
Instance
Enter Instance name.
Append
coords
0, 0
radius
150
Goto
Click to select single object. Shift+Click to de/select multiple objects.
12.86650, 29.50750 Sel: 0 Timing Analyzed
[Mail - Arnav Sharma - 12311748...
Minor_Project
Exp 9 Introduction of Physical De...
student@mit-ecpg-cdn21:Minor_...
Innovus(TM) Implementation Syst...
1 / 2

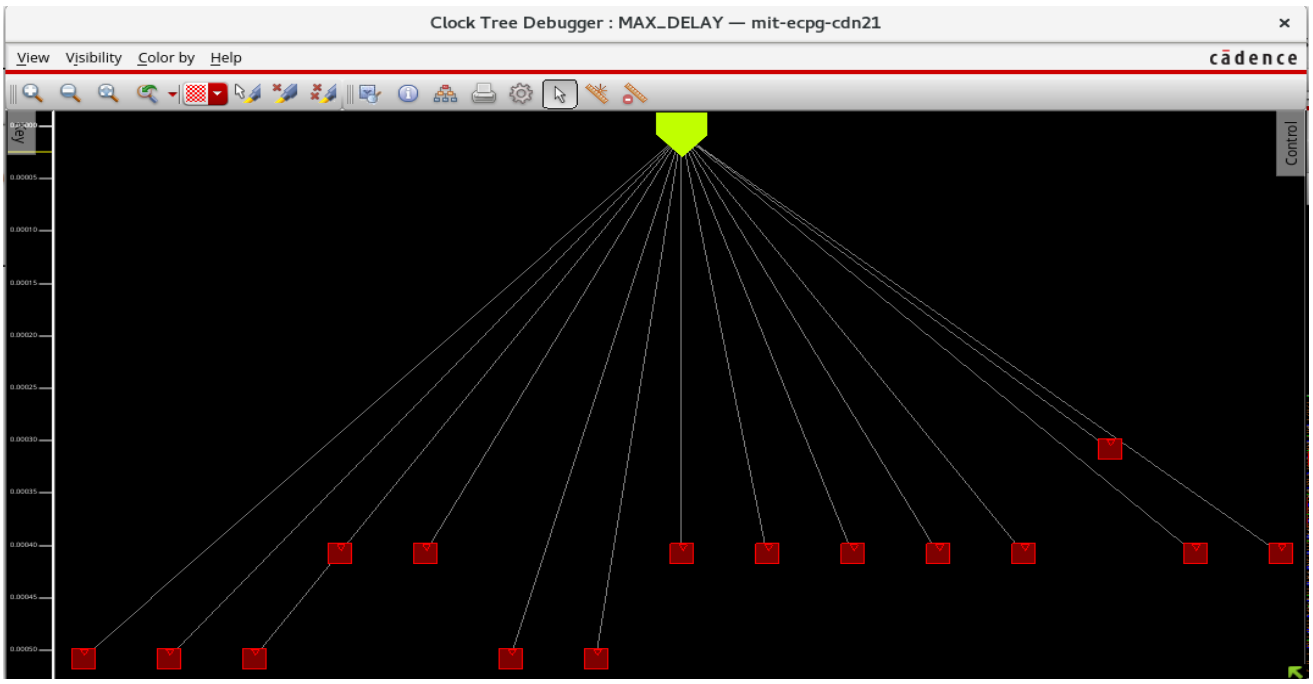
In 3D:


Layout 3D — mit-ecpg-cdn21
Layout 3D
Property
Rotate View:
Move mouse with right mouse button pressed
Pan View:
Move mouse with middle mouse button pressed
Select:
Left mouse button click, Shift key to append
Height Scale:
Scale the height of the object from 1x to 5x
Height Scale 1x
5x

Timing Debug Widow:



Clock Tree Debugger:



Global Routing Report:

```
#Max overcon = 0 track.
#Total overcon = 0.00%.
#Worst layer Gcell overcon rate = 0.00%.
#
#Global routing statistics:
#Cpu time = 00:00:00
#Elapsed time = 00:00:00
#Increased memory = 4.67 (MB)
#Total memory = 1925.55 (MB)
#Peak memory = 2048.06 (MB)
#
#Finished global routing on Thu Mar 27 17:13:05 2025
#
#
#cpu time = 00:00:00, elapsed time = 00:00:00, memory = 1925.87 (MB), peak = 2048.06 (MB)
#Start Track Assignment.
#Done with 138 horizontal wires in 1 hboxes and 154 vertical wires in 1 hboxes.
#Done with 22 horizontal wires in 1 hboxes and 32 vertical wires in 1 hboxes.
#Done with 1 horizontal wires in 1 hboxes and 1 vertical wires in 1 hboxes.
#
#Track assignment summary:
# layer   (wire length)   (overlap)     (long ovlp)     (with obs/pg/clk)
#------------------------------------------------------------------------
# Metal1         0.00     0.00%           0.00%           0.00%
# Metal2       742.18     0.00%           0.00%           0.00%
# Metal3       644.94     0.18%           0.00%           0.00%
# Metal4         0.00     0.00%           0.00%           0.00%
# Metal5         0.00     0.00%           0.00%           0.00%
# Metal6         0.00     0.00%           0.00%           0.00%
# Metal7         0.00     0.00%           0.00%           0.00%
# Metal8         0.00     0.00%           0.00%           0.00%
# Metal9         0.00     0.00%           0.00%           0.00%
#------------------------------------------------------------------------
# All        1387.12      0.08%           0.00%           0.00%
#Complete Track Assignment.
#Total wire length = 1374 um.
#Total half perimeter of net bounding box = 1297 um.
#Total wire length on LAYER Metal1 = 0 um.
#Total wire length on LAYER Metal2 = 738 um.
#Total wire length on LAYER Metal3 = 636 um.
#Total wire length on LAYER Metal4 = 0 um.
#Total wire length on LAYER Metal5 = 0 um.
#Total wire length on LAYER Metal6 = 0 um.
#Total wire length on LAYER Metal7 = 0 um.
#Total wire length on LAYER Metal8 = 0 um.
#Total wire length on LAYER Metal9 = 0 um.
#Total number of vias = 553
#Up-Via Summary (total 553):
```

CONCLUSION:

In this project, we successfully designed and implemented an I2C protocol controller for VLSI applications. Using Verilog, we developed both the master and slave modules, ensuring proper data communication through a finite state machine-based approach. The design was verified in Cadence by generating waveforms to confirm correct timing and data transfer. We further analyzed power, area, and timing using Genus, optimizing the design for efficiency. Finally, the layout was generated in Innovus, completing the full ASIC design flow.

Through this project, we gained insights into digital design, synthesis, and layout generation. The project reinforced key VLSI concepts such as sequential logic, finite state machines, and clocked communication protocols. Overall, this I2C controller serves as a foundational step toward more complex hardware communication interfaces, paving the way for future optimizations in low-power and high-performance embedded systems and IoT projects.

Sources:

1. https://docs.arduino.cc/learn/communication/wire/
2. https://www.ti.com/lit/an/sbaa565/sbaa565.pdf?ts=1743039996771&ref_url=https%253A%252F%252Fwww.google.co.in%252F/
3. https://newhavendisplay.com/blog/i2c-communication-interface/
4. https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/
5. https://www.circuitbread.com/tutorials/what-is-the-i2c-communication-protocol/
6. https://learn.sparkfun.com/tutorials/i2c/all/
7. https://learn.sparkfun.com/tutorials/i2c/all/