BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE 468 (January 2024)
VLSI Laboratory

Final Project Report

Section: G2 Group: 04

12-bit Synchronous Counter

Course Instructors:

Nafis Sadik, Lecturer
Rafid Hasan Palash, Part-Time Lecturer

Signature of Instructor: ______________________________

Academic Honesty Statement:

IMPORTANT! Please carefully read and sign the Academic Honesty Statement, below. Type the student ID and name, and put your signature. *You will not receive credit for this project experiment unless this statement is signed in the presence of your lab instructor.*

"In signing this statement, We hereby certify that the work on this project is our own and that we have not copied the work of any other students (past or present), and cited all relevant sources while completing this project. We understand that if we fail to honor this agreement, We will each receive a score of ZERO for this project and be subject to failure of this course."

Signature: ____________ **Full Name: Rokon Uddin Mahmud** **Student ID: 1906140**	**Signature:** ____________ **Full Name: Rabeya Salam Munira** **Student ID: 1906141**
Signature: ____________ **Full Name: Aliva Sadnim Mahmud** **Student ID: 1906146**	**Signature:** ____________ **Full Name: Abrar Faiaz Eram** **Student ID: 1906167**

Contents

1 Abstract

This project presents a 12-bit synchronous counter designed to perform up/down counting operations based on a control signal. The counter incorporates several control inputs: Load, Up/Down, Reset, and Enable. When the Load signal is high, the counter will be loaded with a 12-bit value from the input bus data[11:0]. The Up/Down signal determines the direction of the count: up for high, down for low. The Reset signal forces the counter to zero. The Enable signal controls the overall operation of the counter, allowing it to be paused when low. The counter is driven by a clock signal and operates synchronously with the positive edge of the clock. This project includes directed and layered testbench design, synthesis optimization and physical design with PnR optimization of the designed 12-bit synchronous counter.

2 Introduction

A 12-bit synchronous counter is a digital circuit capable of counting up or down in binary code, synchronized to a common clock signal. This project focuses on designing and implementing such a counter with several control inputs to enhance its functionality.

By incorporating control signals like Load, Up/Down, Reset, and Enable, the counter can be configured to perform various counting operations. This flexibility makes it suitable for a wide range of applications, including timing circuits, frequency dividers, and digital signal processing systems.

3 Design & Implementation

3.1 Problem Formulation

- The counter is controlled by four input signals Load, Up/Down, Reset, and Enable. There is also a clock signal that will operate the counter at a positive edge.
- When the Load signal is high (1), the counter will load a specified 12-bit number provided on the input lines data[11:0] (parallel loading). data[11:0] represents the 12-bit binary value that will be loaded into the counter. Load signal Forces the counter to start counting from the provided binary value rather than continuing its current count.
- If Up/Down is high (1), the counter increments (counts up) on every positive clock edge, and If Up/Down is low (0), the counter decrements (counts down).
- Reset Signal Clears the counter immediately. If it is high (1), the counter is reset synchronously to zero (000000000000 in binary).
- Enable Controls on whether the counter is active or frozen. If Enable is low (0), the counter is disabled, and its current count value will remain unchanged, irrespective of the clock signal. If Enable is high (1), the counter operates normally according to the Up/Down and Load inputs.

3.2 RTL Design Code

```
module counter(clk, reset, enable, up_down, load, data, count);

  // Define input and output ports
  input clk, reset, load, enable, up_down;
  input [11:0] data;            // 12-bit data input for loading
  output reg [11:0] count;      // 12-bit counter output

  // Always block will be executed at each positive edge of the clock
  always @(posedge clk) begin
    if (reset)
      count <= 12'b0;             // Set Counter to Zero (12-bit)
    else if (enable) begin        // Only operate if enable is high
      if (load)
        count <= data;            // Load the counter with 12-bit data value
      else if (up_down)
        count <= count + 1;       // Count up
      else
        count <= count - 1;       // Count down
    end
    // If enable is low, counter retains its current value
  end

endmodule
```

3.3 Directed Testbench

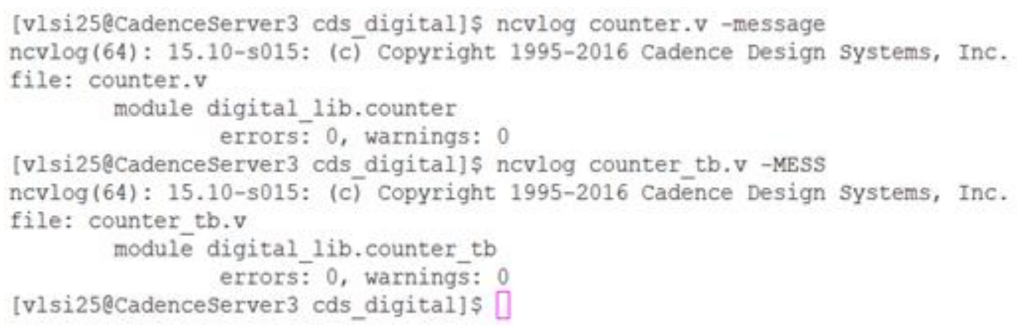
Directed testbench code was written to verify the strategy used in RTL design code.

3.3.1 Directed Testbench Code

```
module counter_tb;
  reg clk, reset, load, enable, up_down;
  reg [11:0] data;      // 12-bit data input
  wire [11:0] count;    // 12-bit counter output

  // Instantiate the 12-bit counter
  counter dut (
    .clk(clk),
    .reset(reset),
    .enable(enable),
    .load(load),
    .up_down(up_down),
    .data(data),
    .count(count)
  );

  // Clock generator (50 MHz)
  initial begin
    clk = 1'b0;
    forever #3 clk = ~clk;  // Toggle the clock every 3 time units
  end

  // Apply reset signals
  initial begin
    reset = 1'b1; #7
    reset = 1'b0; #35
    reset = 1'b1; #10
    reset = 1'b0;
  end
  // Enable signal timing
  initial begin
    enable = 1'b0; #15  // Start with counter disabled
    enable = 1'b1; #25  // Enable counter
    enable = 1'b0; #20  // Disable counter again
    enable = 1'b1;      // Re-enable counter
  end

  // Load signal timing
  initial begin
    #12
    load = 1'b1; #5
    load = 1'b0;
  end

  // Up/Down signal timing
  initial begin
    #5
    up_down = 1'b1; #24
    up_down = 1'b0;
  end

  // Data input stimulus (12-bit values)
  initial begin
    data = 12'b000000010000; #14  // Load 12-bit value 0x100
    data = 12'b000000011101; #2   // Load 12-bit value 0x1D
    data = 12'b000000111111;      // Load 12-bit value 0xFF
  end

  // Monitor input and output signals
  initial begin
    $monitor("time=%0d, reset=%b, enable=%b, load=%b, up_down=%b, data=%d, count=%d",
             $time, reset, enable, load, up_down, data, count);
  end

  // Create a VCD dump for waveform viewing
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars();
    #100 $finish;  // Stop the simulation after 100 time units
  end

endmodule
```

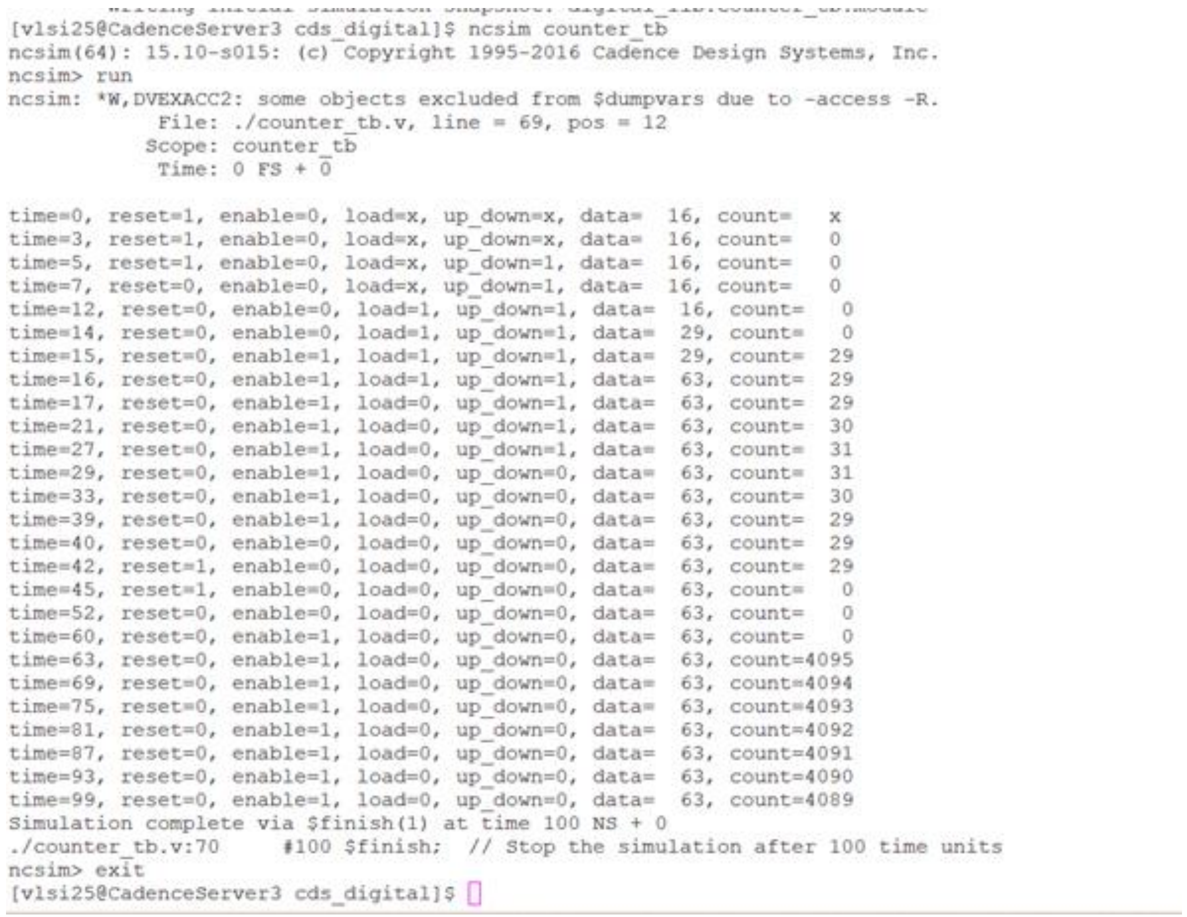
3.3.2 Directed Testbench Analysis

The following figure shows a successful compilation of the design file and the testbench file.



```
[vlsi25@CadenceServer3 cds_digital]$ ncvlog counter.v -message
ncvlog(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
file: counter.v
        module digital_lib.counter
                errors: 0, warnings: 0
[vlsi25@CadenceServer3 cds_digital]$ ncvlog counter_tb.v -MESS
ncvlog(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
file: counter_tb.v
        module digital_lib.counter_tb
                errors: 0, warnings: 0
[vlsi25@CadenceServer3 cds_digital]$
```

Fig 1: Compilation



```
Writing initial simulation snapshot: digital_lib.counter_tb:module
[vlsi25@CadenceServer3 cds_digital]$ ncsim counter_tb
ncsim(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
ncsim> run
ncsim: *W,DVEXACC2: some objects excluded from $dumpvars due to -access -R.
            File: ./counter_tb.v, line = 69, pos = 12
           Scope: counter_tb
            Time: 0 FS + 0

time=0, reset=1, enable=0, load=x, up_down=x, data=  16, count=   x
time=3, reset=1, enable=0, load=x, up_down=x, data=  16, count=   0
time=5, reset=1, enable=0, load=x, up_down=1, data=  16, count=   0
time=7, reset=0, enable=0, load=x, up_down=1, data=  16, count=   0
time=12, reset=0, enable=0, load=1, up_down=1, data=  16, count=   0
time=14, reset=0, enable=0, load=1, up_down=1, data=  29, count=   0
time=15, reset=0, enable=1, load=1, up_down=1, data=  29, count=  29
time=16, reset=0, enable=1, load=1, up_down=1, data=  63, count=  29
time=17, reset=0, enable=1, load=0, up_down=1, data=  63, count=  29
time=21, reset=0, enable=1, load=0, up_down=1, data=  63, count=  30
time=27, reset=0, enable=1, load=0, up_down=1, data=  63, count=  31
time=29, reset=0, enable=1, load=0, up_down=0, data=  63, count=  31
time=33, reset=0, enable=1, load=0, up_down=0, data=  63, count=  30
time=39, reset=0, enable=1, load=0, up_down=0, data=  63, count=  29
time=40, reset=0, enable=0, load=0, up_down=0, data=  63, count=  29
time=42, reset=1, enable=0, load=0, up_down=0, data=  63, count=  29
time=45, reset=1, enable=0, load=0, up_down=0, data=  63, count=   0
time=52, reset=0, enable=0, load=0, up_down=0, data=  63, count=   0
time=60, reset=0, enable=1, load=0, up_down=0, data=  63, count=   0
time=63, reset=0, enable=1, load=0, up_down=0, data=  63, count=4095
time=69, reset=0, enable=1, load=0, up_down=0, data=  63, count=4094
time=75, reset=0, enable=1, load=0, up_down=0, data=  63, count=4093
time=81, reset=0, enable=1, load=0, up_down=0, data=  63, count=4092
time=87, reset=0, enable=1, load=0, up_down=0, data=  63, count=4091
time=93, reset=0, enable=1, load=0, up_down=0, data=  63, count=4090
time=99, reset=0, enable=1, load=0, up_down=0, data=  63, count=4089
Simulation complete via $finish(1) at time 100 NS + 0
./counter_tb.v:70     #100 $finish;  // Stop the simulation after 100 time units
ncsim> exit
[vlsi25@CadenceServer3 cds_digital]$
```

Fig 2: Output display of directed testbench

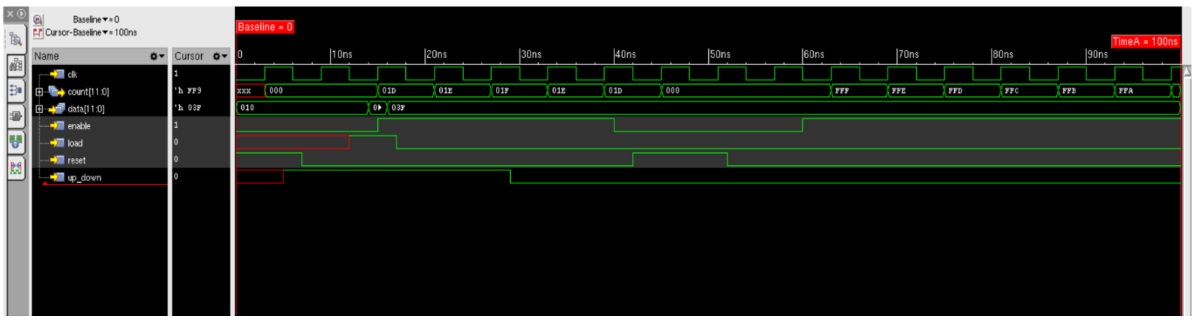


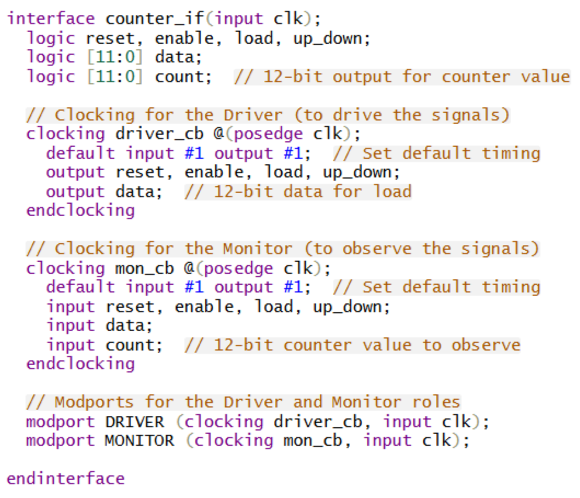
Fig 3: Output waveform of directed testbench

The output indicates successful verification of the design RTL code.

4 Layered Verification

4.1 Code

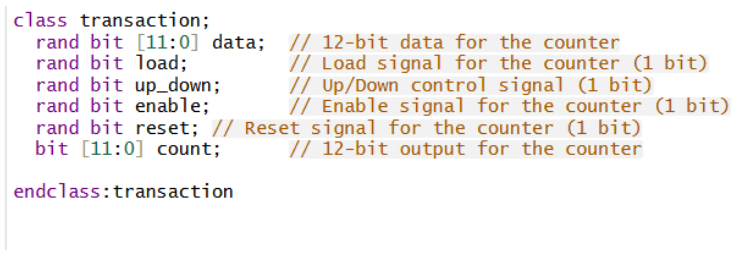
4.1.1 Interface



```
interface counter_if(input clk);
  logic reset, enable, load, up_down;
  logic [11:0] data;
  logic [11:0] count;  // 12-bit output for counter value

  // Clocking for the Driver (to drive the signals)
  clocking driver_cb @(posedge clk);
    default input #1 output #1;  // Set default timing
    output reset, enable, load, up_down;
    output data;  // 12-bit data for load
  endclocking

  // Clocking for the Monitor (to observe the signals)
  clocking mon_cb @(posedge clk);
    default input #1 output #1;  // Set default timing
    input reset, enable, load, up_down;
    input data;
    input count;  // 12-bit counter value to observe
  endclocking

  // Modports for the Driver and Monitor roles
  modport DRIVER (clocking driver_cb, input clk);
  modport MONITOR (clocking mon_cb, input clk);

endinterface
```

The interface (`counter_if`) defines input (`a`, `load`, `up_down`, `reset`, `enable`, `clk`) and output (`count`) signals, along with clocking blocks (`driver_cb` for driving signals and `mon_cb` for observation) to ensure seamless communication between the DUT and testbench.

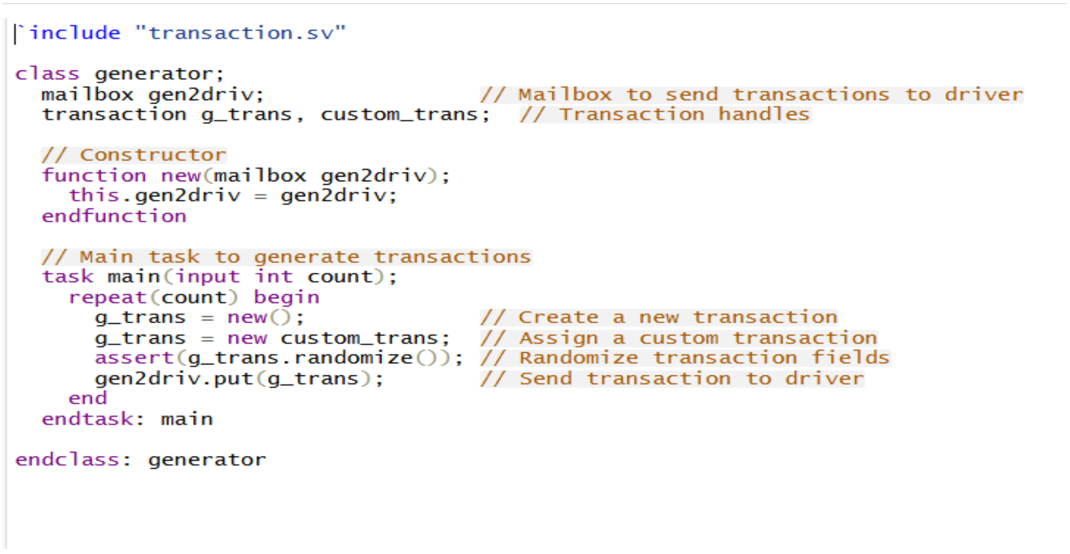
4.1.2 Transaction



```
class transaction;
  rand bit [11:0] data;   // 12-bit data for the counter
  rand bit load;          // Load signal for the counter (1 bit)
  rand bit up_down;       // Up/Down control signal (1 bit)
  rand bit enable;        // Enable signal for the counter (1 bit)
  rand bit reset; // Reset signal for the counter (1 bit)
  bit [11:0] count;       // 12-bit output for the counter

endclass:transaction
```

The transaction class encapsulates inputs and outputs into reusable objects, abstracting signal-level details.

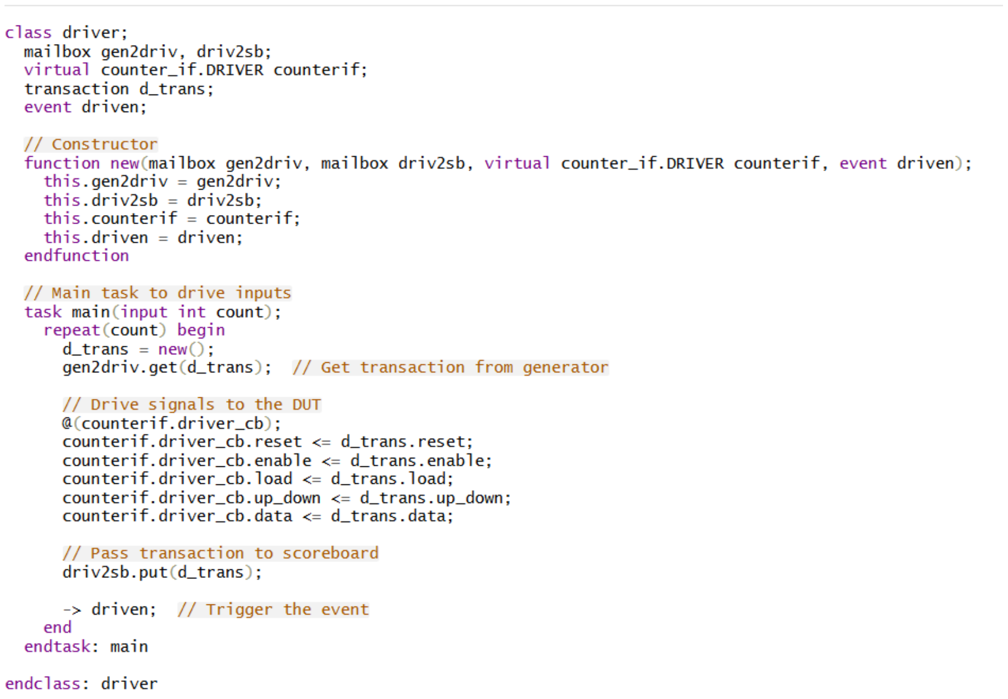
4.1.3 Generator



```
`include "transaction.sv"

class generator;
  mailbox gen2driv;                    // Mailbox to send transactions to driver
  transaction g_trans, custom_trans;   // Transaction handles

  // Constructor
  function new(mailbox gen2driv);
    this.gen2driv = gen2driv;
  endfunction

  // Main task to generate transactions
  task main(input int count);
    repeat(count) begin
      g_trans = new();                 // Create a new transaction
      g_trans = new custom_trans;      // Assign a custom transaction
      assert(g_trans.randomize());     // Randomize transaction fields
      gen2driv.put(g_trans);           // Send transaction to driver
    end
  endtask: main

endclass: generator
```

The generator creates diverse randomized transactions, covering edge cases, and feeds them to the driver.

4.1.4 Driver



```
class driver;
  mailbox gen2driv, driv2sb;
  virtual counter_if.DRIVER counterif;
  transaction d_trans;
  event driven;

  // Constructor
  function new(mailbox gen2driv, mailbox driv2sb, virtual counter_if.DRIVER counterif, event driven);
    this.gen2driv = gen2driv;
    this.driv2sb = driv2sb;
    this.counterif = counterif;
    this.driven = driven;
  endfunction

  // Main task to drive inputs
  task main(input int count);
    repeat(count) begin
      d_trans = new();
      gen2driv.get(d_trans);  // Get transaction from generator

      // Drive signals to the DUT
      @(counterif.driver_cb);
      counterif.driver_cb.reset <= d_trans.reset;
      counterif.driver_cb.enable <= d_trans.enable;
      counterif.driver_cb.load <= d_trans.load;
      counterif.driver_cb.up_down <= d_trans.up_down;
      counterif.driver_cb.data <= d_trans.data;

      // Pass transaction to scoreboard
      driv2sb.put(d_trans);

      -> driven;  // Trigger the event
    end
  endtask: main

endclass: driver
```

The driver class translates transactions into pin-level signal activity, synchronizing with the DUT via the `driver_cb` block.

4.1.5 Monitor

```
class monitor;
  mailbox mon2sb;                 // Mailbox to send monitored transactions to scoreboard
  virtual counter_if.MONITOR counterif;  // Virtual interface for counter monitor
  transaction m_trans;            // Transaction to store monitored data
  event driven;                   // Event to sync with other components

  // Constructor
  function new(mailbox mon2sb, virtual counter_if.MONITOR counterif, event driven);
    this.mon2sb = mon2sb;
    this.counterif = counterif;
    this.driven = driven;
  endfunction

  // Main task to monitor counter signals
  task main(input int count);
    @(driven);  // Wait for event triggered by the driver (ensure driven event is triggered correctly)
    @(counterif.clk);  // Wait for the clock edge (make sure clock is synchronized)
    @(counterif.mon_cb); // Wait for the clocking block (ensure it's synchronized with clk)

    repeat(count) begin
      m_trans = new();  // Create a new transaction for each cycle

      // Wait for the next rising edge of the clock
      @(posedge counterif.clk);
       // Capture the counter signals (reset, enable, load, up_down, data, count)
       m_trans.reset = counterif.mon_cb.reset;
       m_trans.enable = counterif.mon_cb.enable;
       m_trans.load = counterif.mon_cb.load;
       m_trans.up_down = counterif.mon_cb.up_down;
       m_trans.data = counterif.mon_cb.data;
       m_trans.count = counterif.mon_cb.count;  // 12-bit counter value


       // Send the transaction to the scoreboard
       mon2sb.put(m_trans);
    end
  endtask: main

endclass: monitor
```

The monitor observes DUT outputs through the `mon_cb` block, creating observed transactions and passing them to the scoreboard.

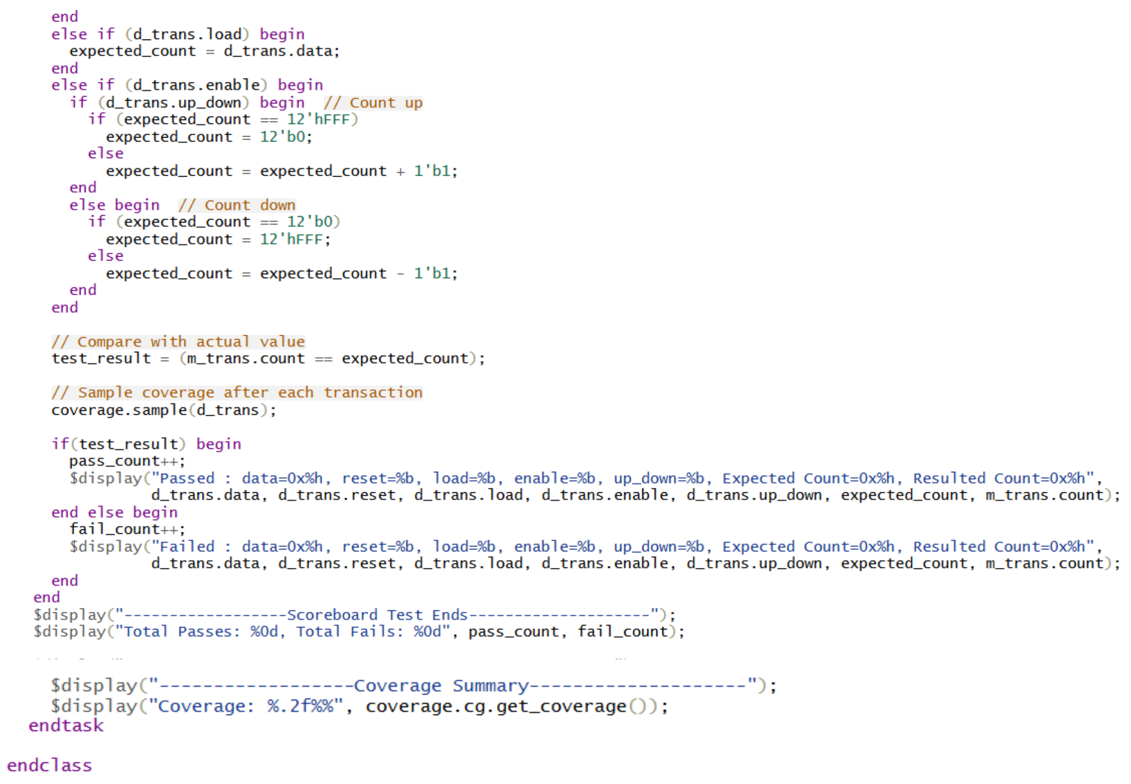
4.1.6 Coverage

```
class counter_coverage;
  // Variables to sample
  bit [11:0] data, count;
  bit load, reset, enable, up_down;

  covergroup cg;
    option.per_instance = 1;
    option.comment = "Counter coverage";

    // Cover basic input values
    data_cp: coverpoint data {
      bins min_edge = {12'h000};
      bins max_edge = {12'hFFF};
      bins others[4] = {[12'h001:12'hFFE]};  // Split other values into 4 bins
    }

    load_cp: coverpoint load;
    reset_cp: coverpoint reset;
    enable_cp: coverpoint enable;
    up_down_cp: coverpoint up_down;

    // Cross coverage
    load_data_cross: cross load_cp, data_cp {
      option.weight = 2;  // Important to verify load with different data values
    }

  endgroup
  function new();
    cg = new;
    cg.start();  // Explicitly start coverage collection
  endfunction

  function void sample(transaction trans);
    this.data = trans.data;
    this.count = trans.count;
    this.load = trans.load;
    this.reset = trans.reset;
    this.enable = trans.enable;
    this.up_down = trans.up_down;
    cg.sample();
  endfunction
endclass
```

The code defines a coverage model for a 12-bit synchronous counter to ensure comprehensive testing of its inputs and control signals. It tracks key variables, including the 12-bit input data, the counter output count (not used directly for coverage), and control signals such as `load`, `reset`, `enable`, and `up_down`. Coverage points are defined for each variable, with `data` divided into edge cases (`0x000` and `0Xfff`) and four intermediate bins (`0x001` to `0xFFE`). Control signals are monitored individually, and cross coverage is added for load and data to test combinations, assigning higher importance (`weight = 2`) to this scenario. The `new()` method initializes and starts the coverage collection, while `sample(transaction trans)` updates variables from the transaction object and records coverage data. This ensures thorough testing of individual signal activity, critical edge cases, intermediate values, and interactions between signals to validate the counter's behavior comprehensively.

4.1.7 Scoreboard

```
class scoreboard;
  mailbox driv2sb;
  mailbox mon2sb;

  transaction d_trans;
  transaction m_trans;

  event driven;

  int pass_count, fail_count;
  bit test_result;  // 1 for pass, 0 for fail

  // Add state tracking
  bit [11:0] expected_count;
  counter_coverage coverage;

  function new(mailbox driv2sb, mon2sb);
    this.driv2sb = driv2sb;
    this.mon2sb = mon2sb;
    this.pass_count = 0;
    this.fail_count = 0;
    this.expected_count = 0; // Initialize counter
    coverage = new();        // Initialize coverage
  endfunction

  task main(input int count);
    $display("------------------Scoreboard Test Starts--------------------");
    repeat(count) begin
      d_trans = new();
      m_trans = new();

      // Get both transactions
      driv2sb.get(d_trans);
      mon2sb.get(m_trans);

      // Calculate expected value based on inputs
      if (d_trans.reset) begin
        expected_count = 12'b0;
      end
```



```
      end
      else if (d_trans.load) begin
        expected_count = d_trans.data;
      end
      else if (d_trans.enable) begin
        if (d_trans.up_down) begin  // Count up
          if (expected_count == 12'hFFF)
            expected_count = 12'b0;
          else
            expected_count = expected_count + 1'b1;
        end
        else begin  // Count down
          if (expected_count == 12'b0)
            expected_count = 12'hFFF;
          else
            expected_count = expected_count - 1'b1;
        end
      end

      // Compare with actual value
      test_result = (m_trans.count == expected_count);

      // Sample coverage after each transaction
      coverage.sample(d_trans);

      if(test_result) begin
        pass_count++;
        $display("Passed : data=0x%h, reset=%b, load=%b, enable=%b, up_down=%b, Expected Count=0x%h, Resulted Count=0x%h",
                 d_trans.data, d_trans.reset, d_trans.load, d_trans.enable, d_trans.up_down, expected_count, m_trans.count);
      end else begin
        fail_count++;
        $display("Failed : data=0x%h, reset=%b, load=%b, enable=%b, up_down=%b, Expected Count=0x%h, Resulted Count=0x%h",
                 d_trans.data, d_trans.reset, d_trans.load, d_trans.enable, d_trans.up_down, expected_count, m_trans.count);
      end
    end
    $display("------------------Scoreboard Test Ends--------------------");
    $display("Total Passes: %0d, Total Fails: %0d", pass_count, fail_count);

    $display("------------------Coverage Summary--------------------");
    $display("Coverage: %.2f%%", coverage.cg.get_coverage());
  endtask

endclass
```

This code implements a **scoreboard class** to validate the functionality of a 12-bit counter by comparing expected and actual counter values during a simulation. It utilizes two mailboxes (`driv2sb` and `mon2sb`) to receive transactions from the driver and monitor. The `main` task executes the test by processing a specified number of transactions. It calculates the expected counter value based on the inputs (`reset`, `load`, `enable`, and `up_down`), including handling edge cases for counting up or down at the limits (`0x000` and `0xFFF`). The actual counter value from the monitor is compared with the expected value to determine whether the test passes or fails.

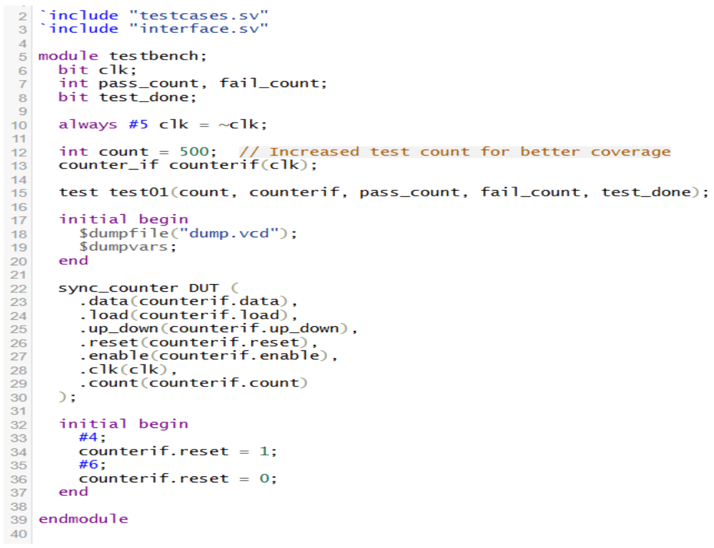
Pass and fail counts are tracked, and coverage is sampled after each transaction using the `counter_coverage` instance. Test results are printed for each transaction, showing the inputs, expected count, and resulting count. At the end, the scoreboard displays the total passes and failures along with a coverage summary, which reports how well the test exercised the specified conditions. This design ensures thorough validation of the counter's functionality and provides insightful metrics for test effectiveness.

4.1.8 Environment

```
`include "generator.sv"
`include "driver.sv"
`include "monitor.sv"
`include "scoreboard.sv"

class environment;
  mailbox gen2driv;    // Mailbox between generator and driver
  mailbox driv2sb;     // Mailbox between driver and scoreboard
  mailbox mon2sb;      // Mailbox between monitor and scoreboard

  generator gen;       // Generator instance
  driver drv;          // Driver instance
  monitor mon;         // Monitor instance
  scoreboard scb;      // Scoreboard instance

  event driven;        // Event to sync driver and monitor

  virtual counter_if counterif;  // Virtual interface for the counter

  // Constructor
  function new(virtual counter_if counterif);
    this.counterif = counterif;
    gen2driv = new();    // Initialize mailboxes
    driv2sb = new();
    mon2sb = new();

    gen = new(gen2driv);                            // Instantiate generator
    drv = new(gen2driv, driv2sb, counterif.DRIVER, driven);  // Instantiate driver
    mon = new(mon2sb, counterif.MONITOR, driven);            // Instantiate monitor
    scb = new(driv2sb, mon2sb);                     // Instantiate scoreboard
  endfunction

  // Main task
  task main(input int count);
    fork
      gen.main(count);  // Start generator
      drv.main(count);  // Start driver
      mon.main(count);  // Start monitor
      scb.main(count);  // Start scoreboard
    join
    $finish;  // Finish simulation after tasks are complete
  endtask: main

endclass: environment
```

The environment integrates all components, connecting them via mailboxes and the interface, and manages the simulation by coordinating their execution.

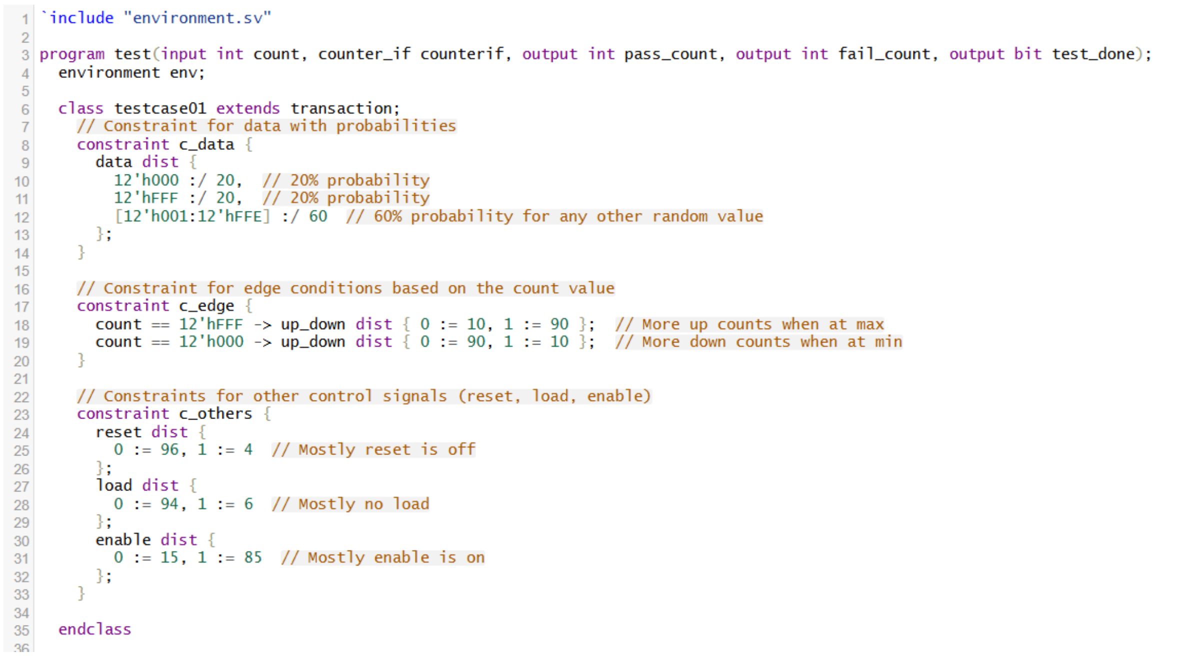
4.1.9 Testbench



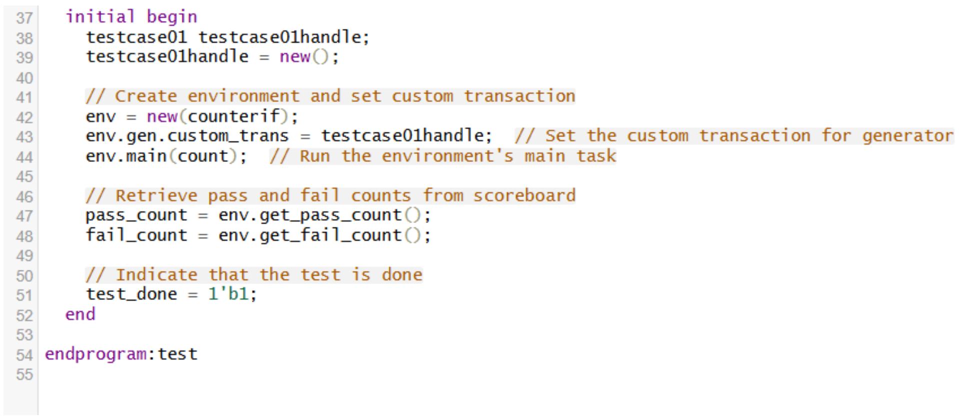
```
`include "testcases.sv"
`include "interface.sv"

module testbench;
  bit clk;
  int pass_count, fail_count;
  bit test_done;

  always #5 clk = ~clk;

  int count = 500;  // Increased test count for better coverage
  counter_if counterif(clk);

  test test01(count, counterif, pass_count, fail_count, test_done);

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

  sync_counter DUT (
    .data(counterif.data),
    .load(counterif.load),
    .up_down(counterif.up_down),
    .reset(counterif.reset),
    .enable(counterif.enable),
    .clk(clk),
    .count(counterif.count)
  );

  initial begin
    #4;
    counterif.reset = 1;
    #6;
    counterif.reset = 0;
  end

endmodule
```

The testbench sets up a complete environment to verify a 12-bit synchronous counter, generating a clock signal that toggles every 5 time units and instantiating the `counter_if` interface to connect the DUT. It configures the test with 500 transactions for improved coverage, running the `test01` instance to validate the counter's functionality while tracking `pass_count`, `fail_count`, and `test_done`. The DUT (`sync_counter`) is instantiated with its inputs (`data`, `load`, `up_down`, `reset`, `enable`, and `clk`) connected to the interface and its output (`count`) monitored. To enable waveform analysis, a VCD file is generated using `$dumpfile` and `$dumpvars`. A reset is applied at the start of the simulation, setting `counterif.reset` high for 6 time units before releasing it, ensuring the DUT initializes correctly. This testbench integrates testcases, scoreboard, and interface to comprehensively verify the counter.

4.1.10 Testcases



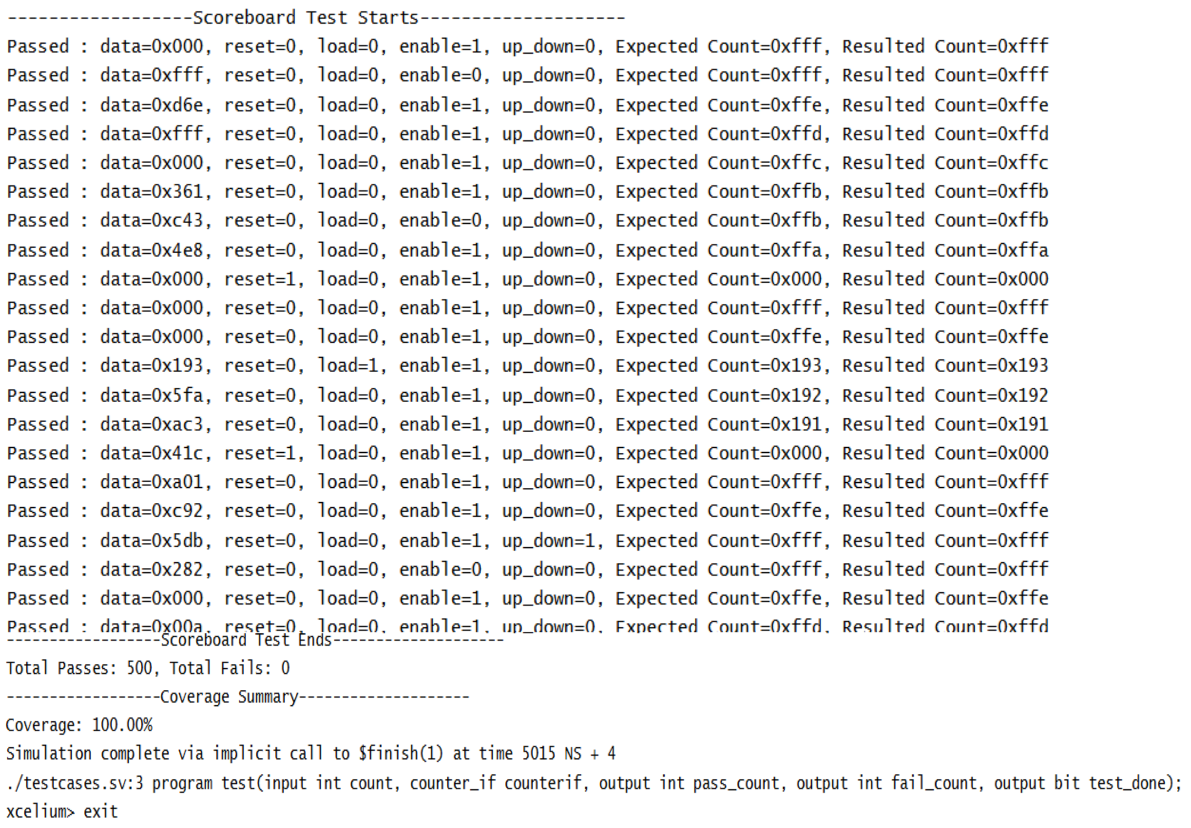
```
`include "environment.sv"

program test(input int count, counter_if counterif, output int pass_count, output int fail_count, output bit test_done);
  environment env;

  class testcase01 extends transaction;
    // Constraint for data with probabilities
    constraint c_data {
      data dist {
        12'h000 :/ 20,  // 20% probability
        12'hFFF :/ 20,  // 20% probability
        [12'h001:12'hFFE] :/ 60  // 60% probability for any other random value
      };
    }

    // Constraint for edge conditions based on the count value
    constraint c_edge {
      count == 12'hFFF -> up_down dist { 0 := 10, 1 := 90 };  // More up counts when at max
      count == 12'h000 -> up_down dist { 0 := 90, 1 := 10 };  // More down counts when at min
    }

    // Constraints for other control signals (reset, load, enable)
    constraint c_others {
      reset dist {
        0 := 96, 1 := 4  // Mostly reset is off
      };
      load dist {
        0 := 94, 1 := 6  // Mostly no load
      };
      enable dist {
        0 := 15, 1 := 85  // Mostly enable is on
      };
    }

  endclass
```



```
  initial begin
    testcase01 testcase01handle;
    testcase01handle = new();

    // Create environment and set custom transaction
    env = new(counterif);
    env.gen.custom_trans = testcase01handle;  // Set the custom transaction for generator
    env.main(count);  // Run the environment's main task

    // Retrieve pass and fail counts from scoreboard
    pass_count = env.get_pass_count();
    fail_count = env.get_fail_count();

    // Indicate that the test is done
    test_done = 1'b1;
  end

endprogram:test
```

This is the test program for testing the 12-bit counter design with custom constraints. It includes a `testcase01` class that defines distributions for the `data` input, favoring edge values (`0x000` and `0xFFF`) at 20% probability each, while other values in the range `0x001` to `0xFFE` have 60% probability. It also specifies edge conditions where at `count == 0xFFF`, the counter favors `up_down = 1` (up count), and at `count == 0x000`, it favors `up_down = 0` (down count). Additionally, it defines constraints for control signals (`reset`, `load`, `enable`) with specific probabilities, mostly keeping `reset` and `load` off and `enable` on. The test program creates an instance of the `testcase01` class, assigns it to the environment's generator, and runs the test using `env.main(count)`. After executing the test, it retrieves the pass and fail counts from the scoreboard and indicates test completion by setting `test_done` to `1`.

4.2 Output



```
------------------Scoreboard Test Starts--------------------
Passed : data=0x000, reset=0, load=0, enable=1, up_down=0, Expected Count=0xfff, Resulted Count=0xfff
Passed : data=0xfff, reset=0, load=0, enable=0, up_down=0, Expected Count=0xfff, Resulted Count=0xfff
Passed : data=0xd6e, reset=0, load=0, enable=1, up_down=0, Expected Count=0xffe, Resulted Count=0xffe
Passed : data=0xfff, reset=0, load=0, enable=1, up_down=0, Expected Count=0xffd, Resulted Count=0xffd
Passed : data=0x000, reset=0, load=0, enable=1, up_down=0, Expected Count=0xffc, Resulted Count=0xffc
Passed : data=0x361, reset=0, load=0, enable=1, up_down=0, Expected Count=0xffb, Resulted Count=0xffb
Passed : data=0xc43, reset=0, load=0, enable=0, up_down=0, Expected Count=0xffb, Resulted Count=0xffb
Passed : data=0x4e8, reset=0, load=0, enable=1, up_down=0, Expected Count=0xffa, Resulted Count=0xffa
Passed : data=0x000, reset=1, load=0, enable=1, up_down=0, Expected Count=0x000, Resulted Count=0x000
Passed : data=0x000, reset=0, load=0, enable=1, up_down=0, Expected Count=0xfff, Resulted Count=0xfff
Passed : data=0x000, reset=0, load=0, enable=1, up_down=0, Expected Count=0xffe, Resulted Count=0xffe
Passed : data=0x193, reset=0, load=1, enable=1, up_down=0, Expected Count=0x193, Resulted Count=0x193
Passed : data=0x5fa, reset=0, load=0, enable=1, up_down=0, Expected Count=0x192, Resulted Count=0x192
Passed : data=0xac3, reset=0, load=0, enable=1, up_down=0, Expected Count=0x191, Resulted Count=0x191
Passed : data=0x41c, reset=1, load=0, enable=1, up_down=0, Expected Count=0x000, Resulted Count=0x000
Passed : data=0xa01, reset=0, load=0, enable=1, up_down=0, Expected Count=0xfff, Resulted Count=0xfff
Passed : data=0xc92, reset=0, load=0, enable=1, up_down=0, Expected Count=0xffe, Resulted Count=0xffe
Passed : data=0x5db, reset=0, load=0, enable=1, up_down=1, Expected Count=0xfff, Resulted Count=0xfff
Passed : data=0x282, reset=0, load=0, enable=0, up_down=0, Expected Count=0xfff, Resulted Count=0xfff
Passed : data=0x000, reset=0, load=0, enable=1, up_down=0, Expected Count=0xffe, Resulted Count=0xffe
Passed : data=0x00a, reset=0, load=0, enable=1, up_down=0, Expected Count=0xffd, Resulted Count=0xffd
------------------Scoreboard Test Ends--------------------
Total Passes: 500, Total Fails: 0
------------------Coverage Summary-------------------
Coverage: 100.00%
Simulation complete via implicit call to $finish(1) at time 5015 NS + 4
./testcases.sv:3 program test(input int count, counter_if counterif, output int pass_count, output int fail_count, output bit test_done);
xcelium> exit
```

Fig 4: Output display of layered testbench

The output shows that all testcases are passed and there is zero failed cases. So, our design works fine. Also, the coverage summary shows it has 100% coverage since it covers all the test cases.

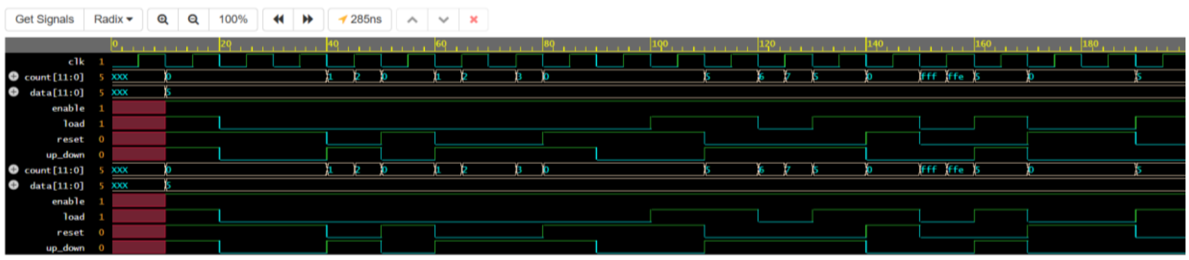


Fig 4: Output display of layered testbench (contd)

Here, the `count` output from DUT and monitor matches. Thus, the output indicates successful verification.

5 Synthesis Optimization

5.1 Clock

For area and power optimization, synthesis is done for five separate values of clock period as per the following tables in section 5.1. the respective plots show an approximately inverse relationship between area and power, as the increase of either results in the decrease of the other. As such, in this instance the power optimization is considered to be more important, and the optimized clock period is taken as 40ns. Further increase of clock period would reduce the power consumption more, but the tradeoff of reduced operation speed would be a hindrance.

5.1.1 Area

Effort	Clock Period	Total Area
medium	5	456
	10	679
	20	662
	40	702
	80	700

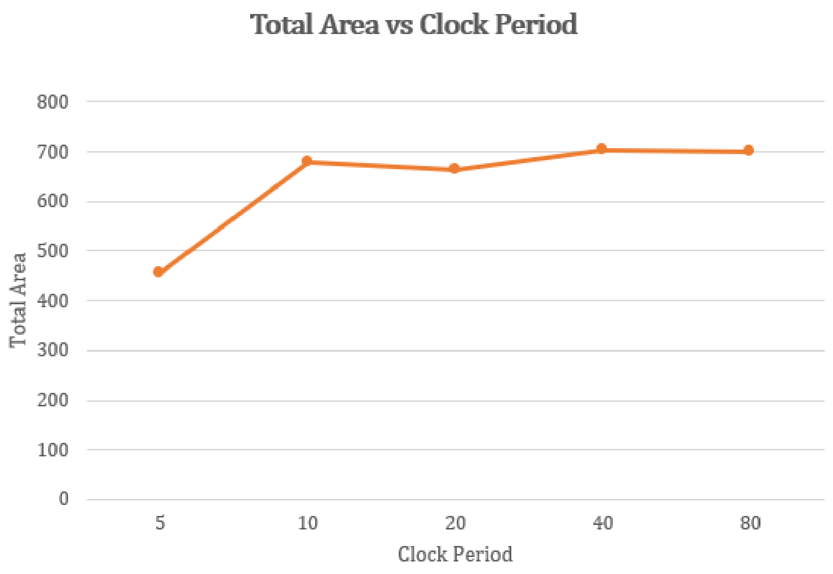


Fig 5: Total Area vs Clock Period (Synthesis)

5.1.2 Power

Effort	Clock Period	Leakage Power	Internal Power	Net Power	Dynamic (Switching) Power
medium	5	4.417	5531.831	4237.819	9769.65
	10	4.291	2804.541	4607.761	7412.302
	20	4.213	1569.535	2609.212	4178.747
	40	4.396	1042.881	1845.028	2887.909
	80	4.45	992.223	1786.303	2778.526

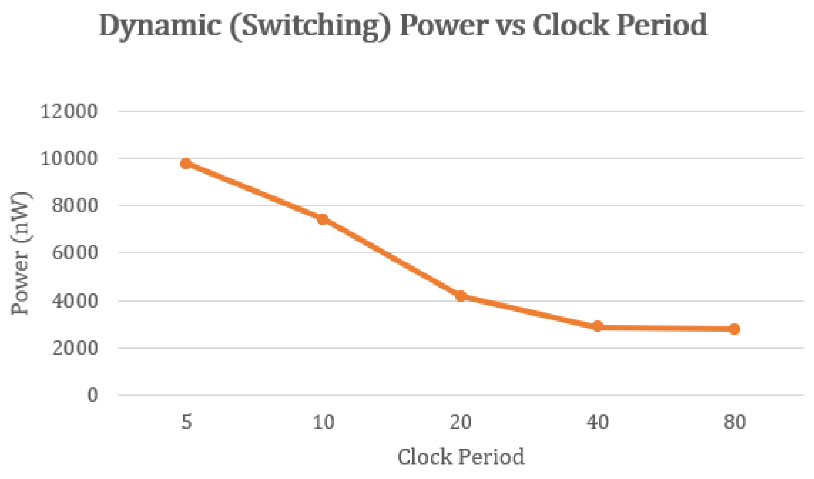


Fig 6: Dynamic Power vs Clock Period (Synthesis)

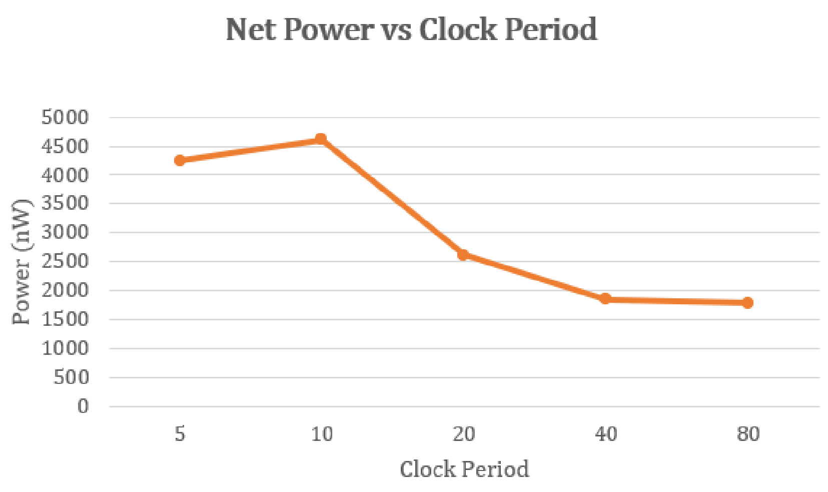


Fig 7: Net Power vs Clock Period (Synthesis)

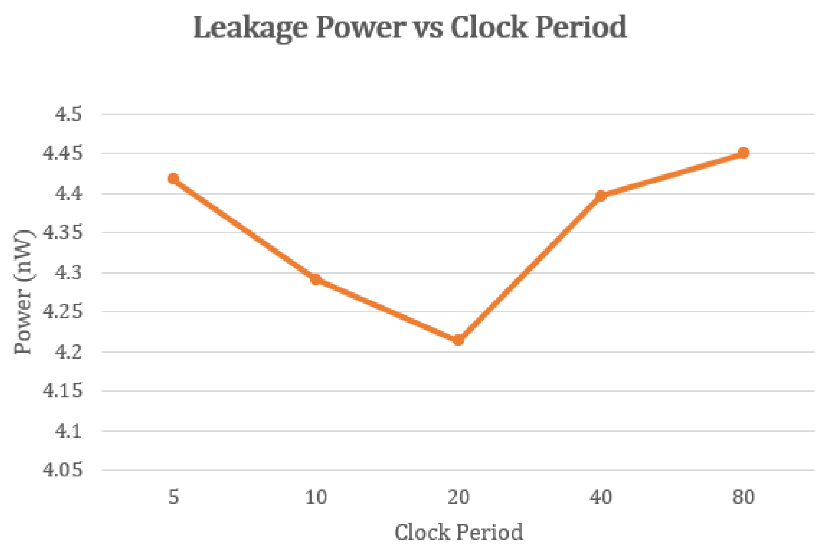


Fig 8: Leakage Power vs Clock Period (Synthesis)

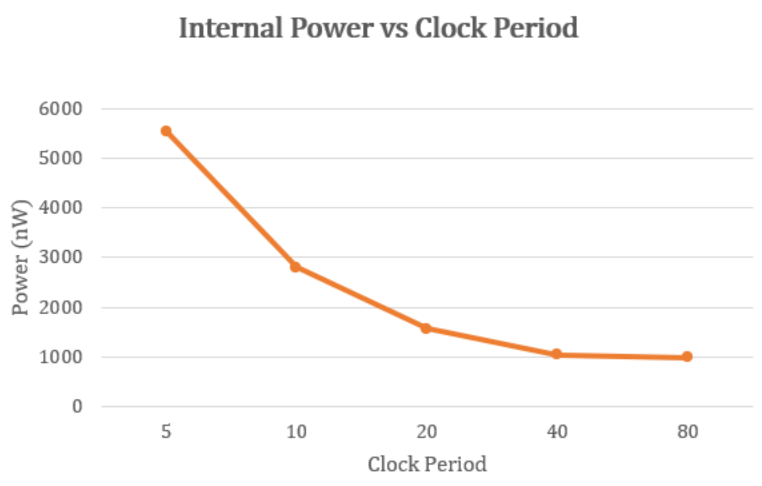


Fig 9: Internal Power vs Clock Period (Synthesis)

5.2 Input Delay

After the selection of 40ns as the clock period, the input delay is varied discretely with relation to the current clock period. The values presented in the table and the corresponding figures represent multipliers of clock period. For example, here, input delay 0.3 corresponds to an input delay of 0.3 * 40 ns = 12 ns. This is the optimized value in this regard.

5.2.1 Area

Effort	Input Delay	Total Area
medium	0.1	456
	0.2	700
	0.3	674
	0.4	690

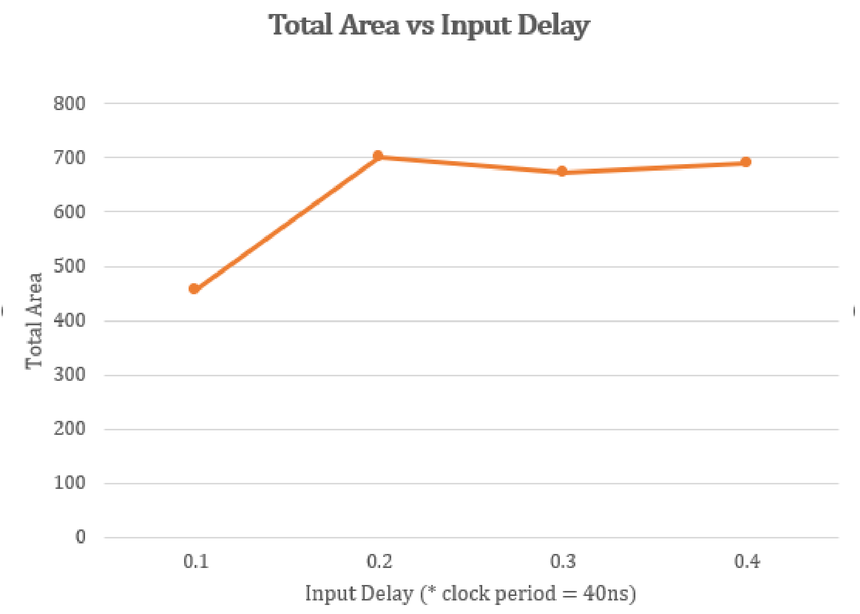


Fig 10: Total Area vs Input Delay (Synthesis)

5.2.2 Power

Effort	Input Delay	Leakage Power	Internal Power	Net Power	Dynamic (Switching) Power
medium	0.1	4.417	1581.529	1381.809	2963.338
	0.2	4.455	1036.95	1827.309	2864.259
	0.3	4.269	1000.395	1712.137	2712.532
	0.4	4.375	1025.11	1793.137	2818.248

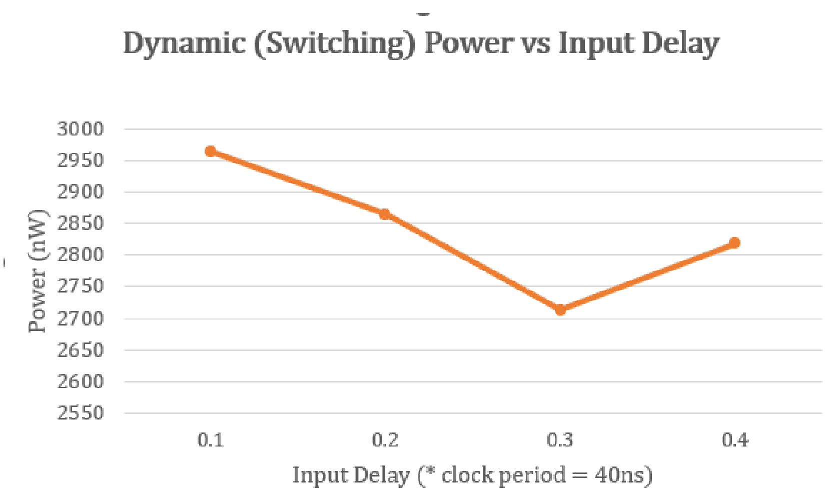


Fig 11: Dynamic Power vs Input Delay (Synthesis)

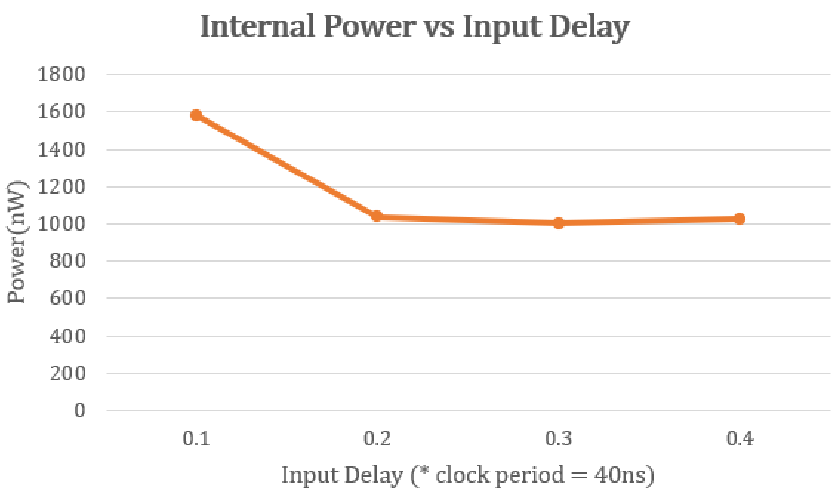


Fig 12: Internal Power vs Input Delay (Synthesis)

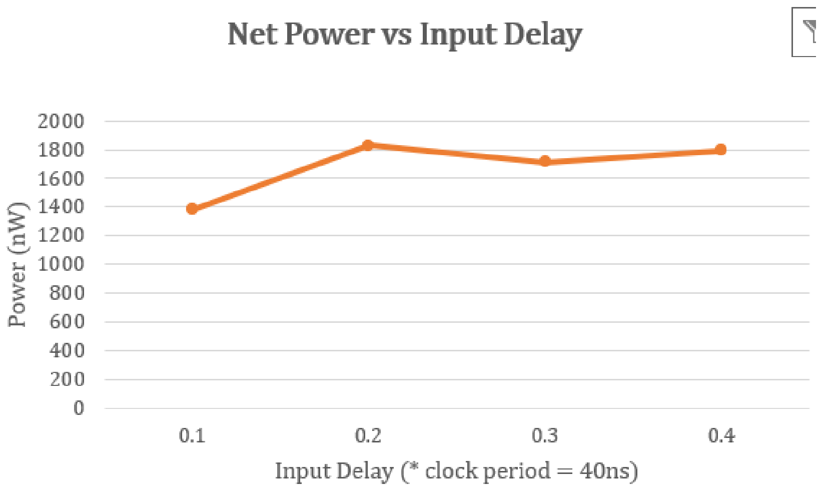


Fig 13: Net Power vs Input Delay (Synthesis)

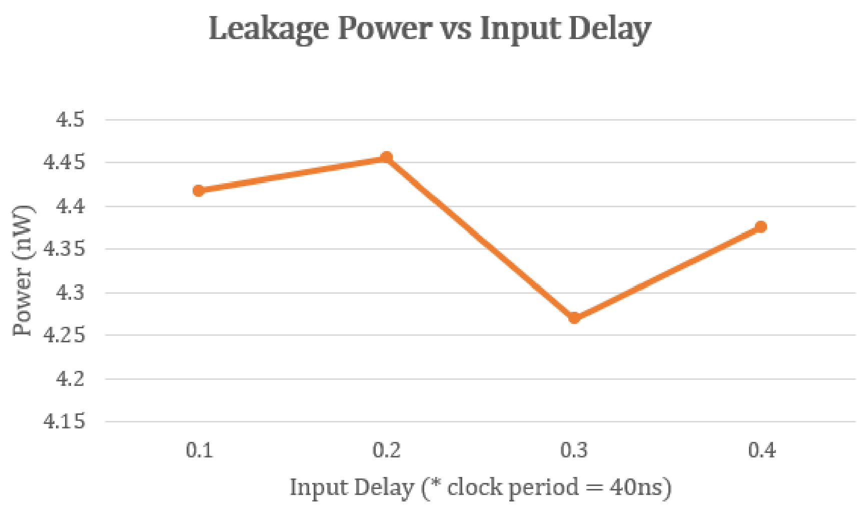


Fig 14: Leakage Power vs Input Delay (Synthesis)

5.3 Output Delay

Output delay was varied for the set values of clock period and input delay. Here, the output delays are also the multipliers of clock period. The optimized output delay is 24 ns.

5.3.1 Area

Effort	Output Delay	Total Area
medium	0.4	456
	0.5	700
	0.6	674
	0.7	690

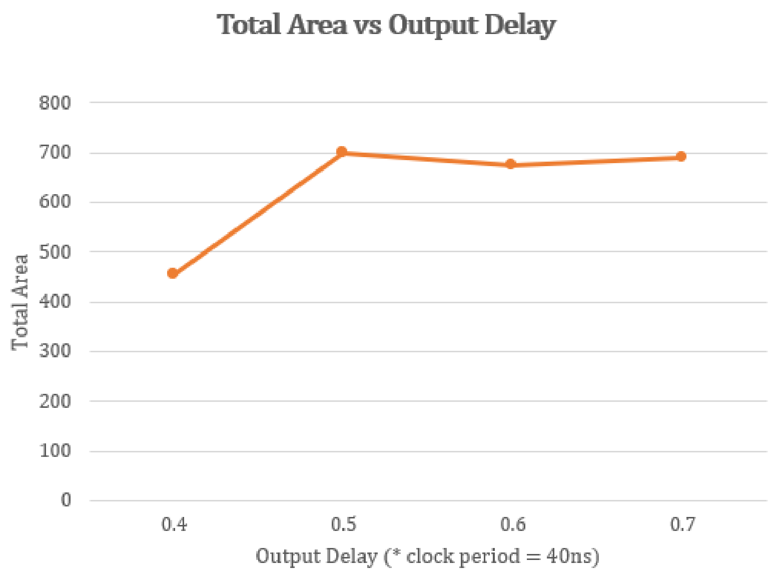


Fig 15: Total Area vs Output Delay (Synthesis)

5.3.2 Power

Effort	Output Delay	Leakage Power	Internal Power	Net Power	Dynamic (Switching) Power
medium	0.4	4.417	1581.529	1381.809	2963.338
	0.5	4.455	1036.95	1827.309	2864.259
	0.6	4.269	1000.395	1712.137	2712.532
	0.7	4.375	1025.11	1793.137	2818.248

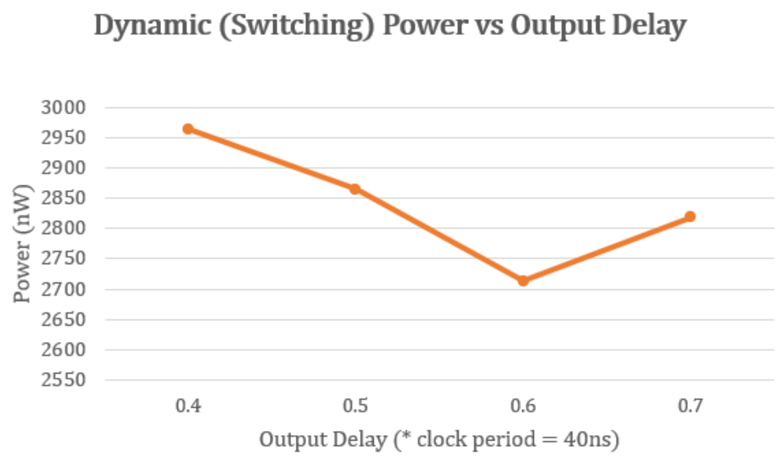


Fig 16: Dynamic Power vs Output Delay (Synthesis)

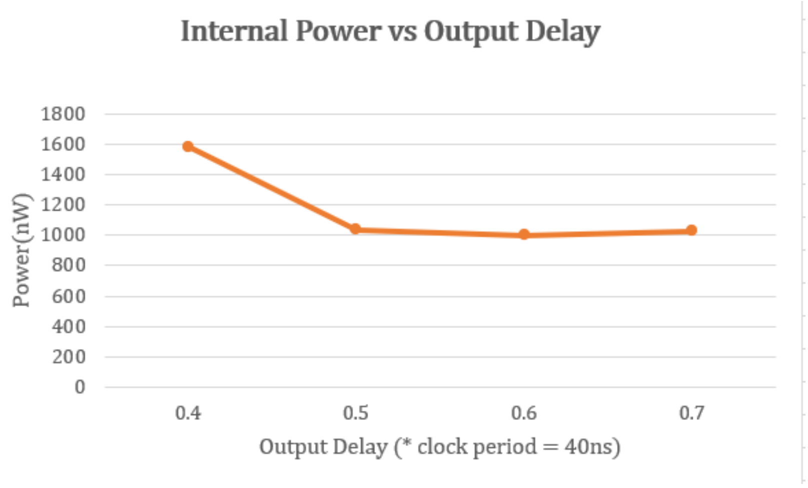


Fig 17: Internal Power vs Output Delay (Synthesis)

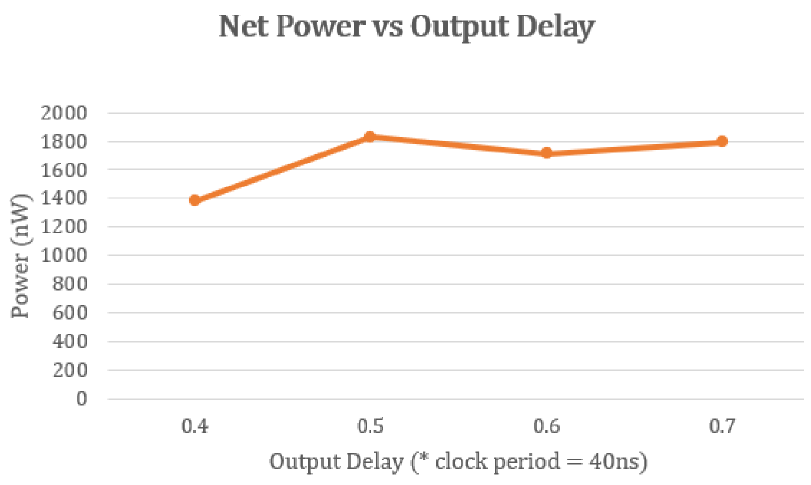


Fig 18: Net Power vs Output Delay (Synthesis)

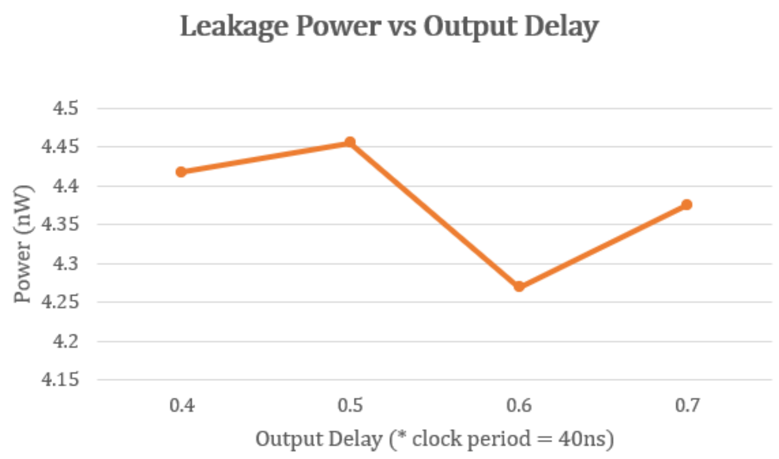


Fig 19: Leakage Power vs Output Delay (Synthesis)

5.4 Setup Time

Setup time was varied for the set values of clock period, input delay and output delay. The optimized setup time is 2 ns.

5.4.1 Area

Effort	Setup Time	Total Area
medium	0.5	456
	1	700
	2	674
	4	690

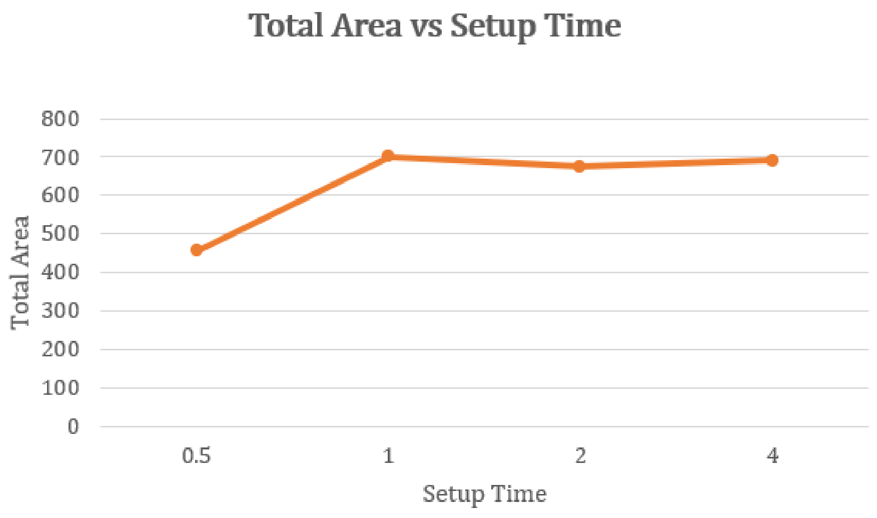


Fig 20: Total Area vs Setup Time (Synthesis)

5.4.2 Power

Effort	Setup Time	Leakage Power	Internal Power	Net Power	Dynamic (Switching) Power
medium	0.5	4.417	1581.529	1381.809	2963.338
	1	4.455	1036.95	1827.309	2864.259
	2	4.269	1000.395	1712.137	2712.532
	4	4.375	1025.11	1793.137	2818.248

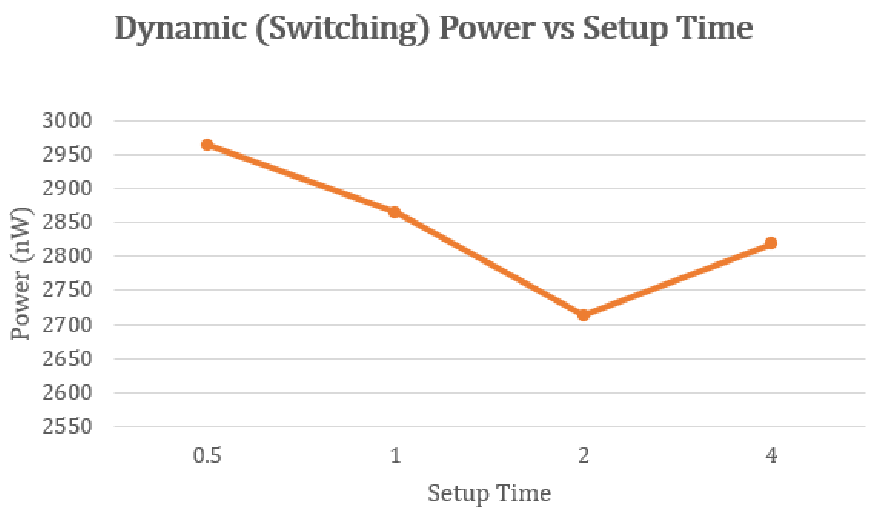


Fig 21: Dynamic Power vs Setup Time (Synthesis)

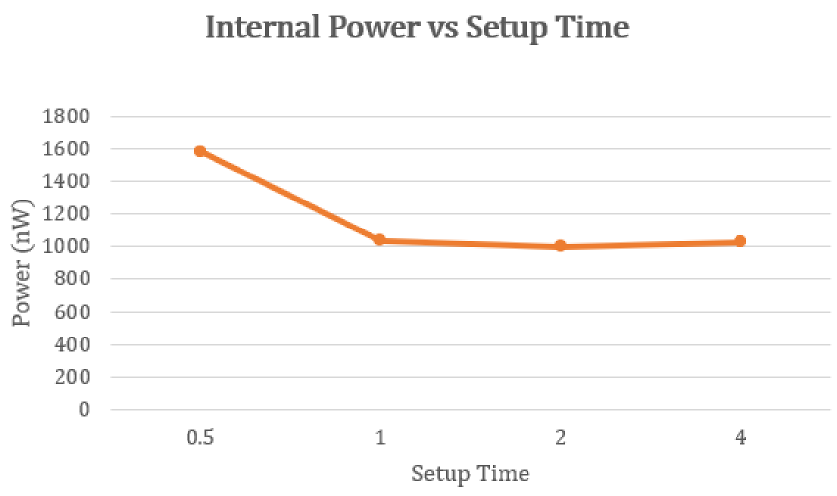


Fig 22: Internal Power vs Setup Time (Synthesis)

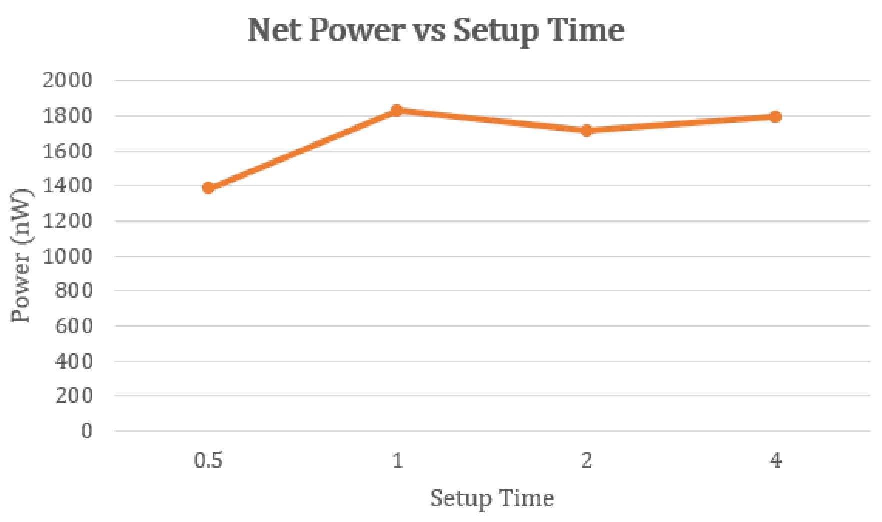


Fig 23: Net Power vs Setup Time (Synthesis)

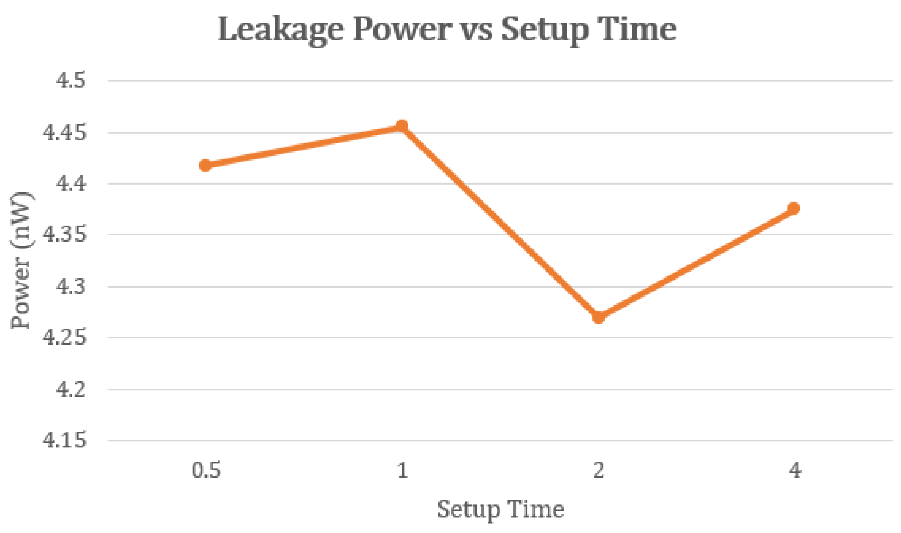


Fig 24: Leakage Power vs Setup Time (Synthesis)

5.5 Hold Time

Hold time was varied for the set values of clock period, input delay, output delay, and setup time. The optimized hold time is 0.8 ns.

5.5.1 Area

Effort	Hold Time	Total Area
medium	0.4	456
	0.6	700
	0.8	674
	1	690

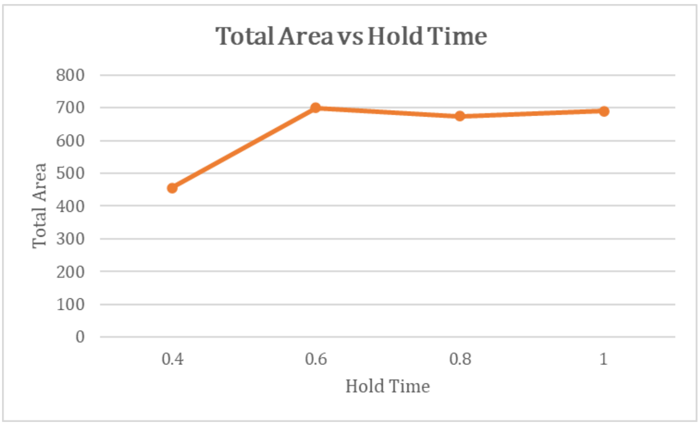


Fig 25: Total Area vs Hold Time (Synthesis)

5.5.2 Power

Effort	Hold Time	Leakage Power	Internal Power	Net Power	Dynamic (Switching) Power
medium	0.4	4.417	1581.529	1381.809	2963.338
	0.6	4.455	1036.95	1827.309	2864.259
	0.8	4.269	1000.395	1712.137	2712.532
	1	4.375	1025.11	1793.137	2818.248

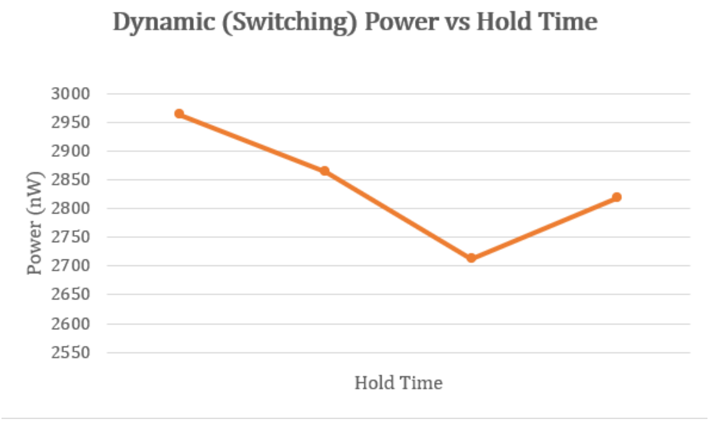


Fig 26: Dynamic Power vs Hold Time (Synthesis)

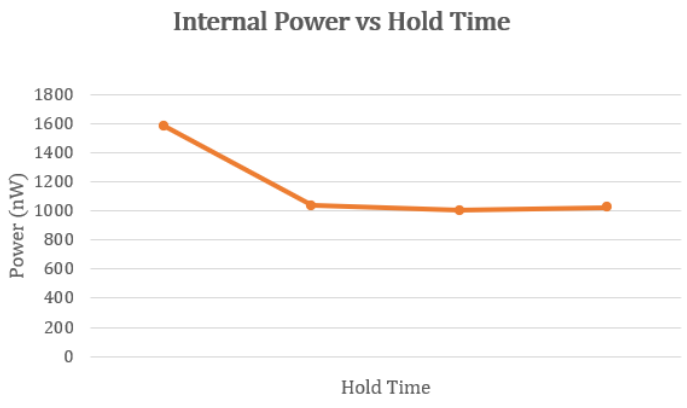


Fig 27: Internal Power vs Hold Time (Synthesis)

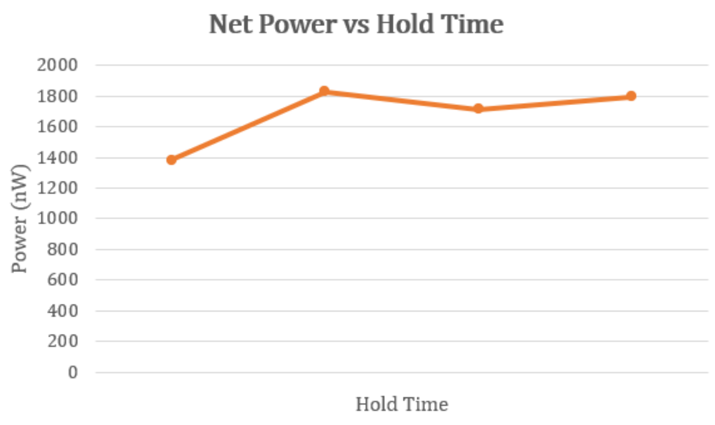


Fig 28: Net Power vs Hold Time (Synthesis)

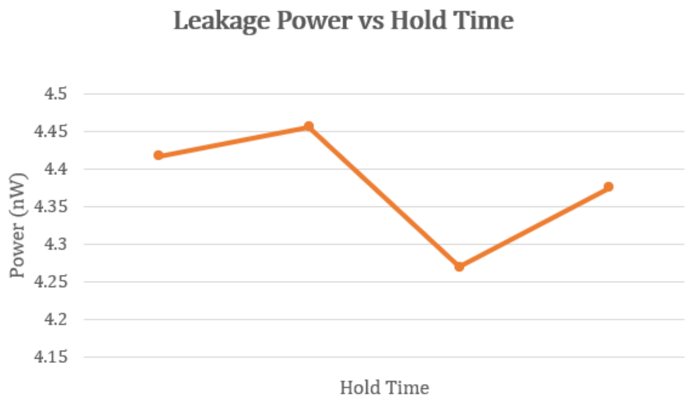


Fig 29: Leakage Power vs Hold Time (Synthesis)

6 Synthesis

6.1 Optimized Synthesis

Synthesis was performed using the optimized parameters where the effort was medium.

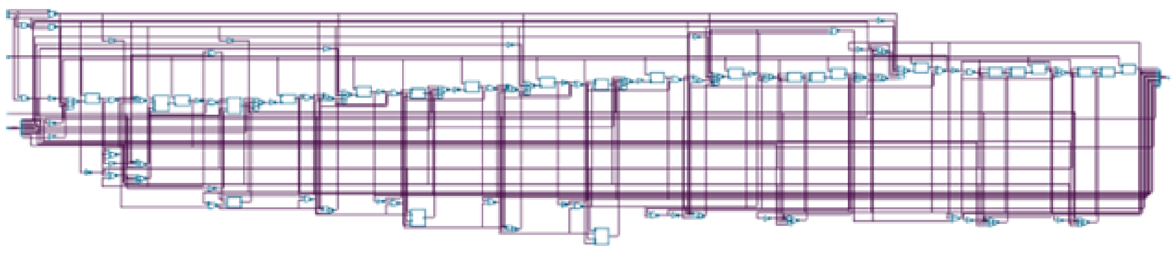
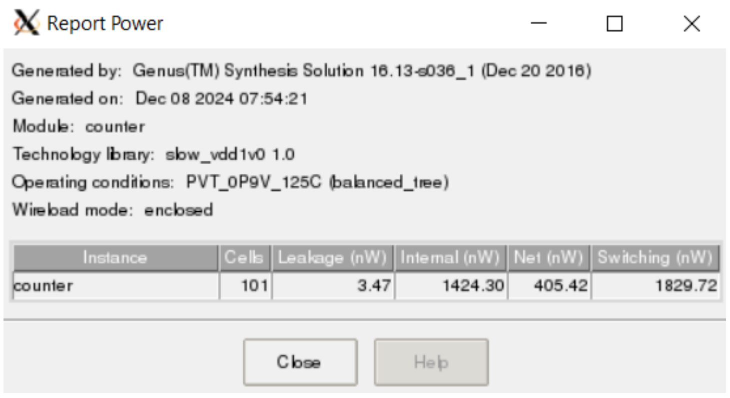


Fig 30: Schematic (Synthesis)

6.2 Synthesis Reports

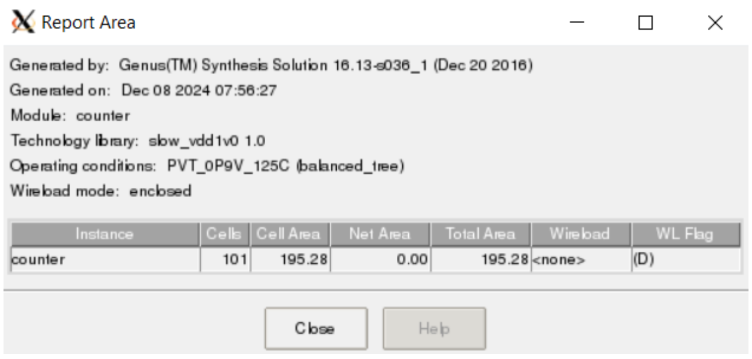


Report Power

Generated by: Genus(TM) Synthesis Solution 16.13-s036_1 (Dec 20 2016)
Generated on: Dec 08 2024 07:54:21
Module: counter
Technology library: slow_vdd1v0 1.0
Operating conditions: PVT_0P9V_125C (balanced_tree)
Wireload mode: enclosed

Instance	Cells	Leakage (nW)	Internal (nW)	Net (nW)	Switching (nW)
counter	101	3.47	1424.30	405.42	1829.72

Fig 31: Power Report for medium effort (Synthesis)

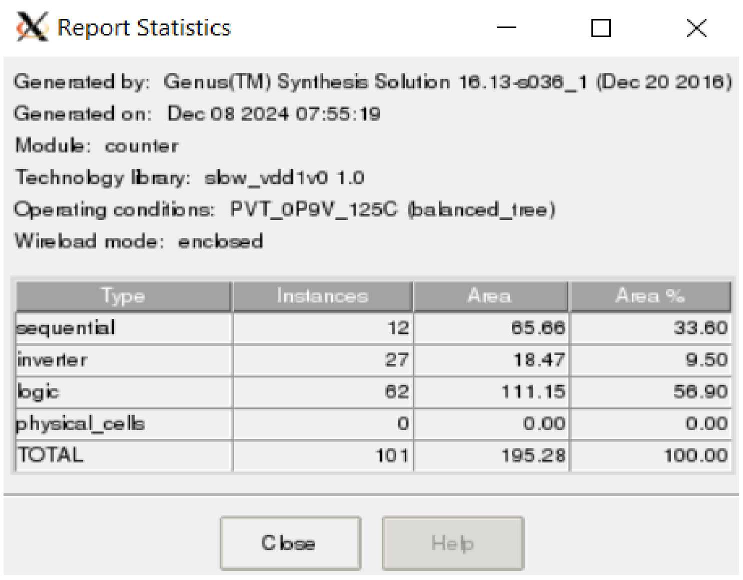


Report Area

Generated by: Genus(TM) Synthesis Solution 16.13-s036_1 (Dec 20 2016)
Generated on: Dec 08 2024 07:56:27
Module: counter
Technology library: slow_vdd1v0 1.0
Operating conditions: PVT_0P9V_125C (balanced_tree)
Wireload mode: enclosed

Instance	Cells	Cell Area	Net Area	Total Area	Wireload	WL Flag
counter	101	195.28	0.00	195.28	<none>	(D)

Fig 32: Area Report for medium effort (Synthesis)

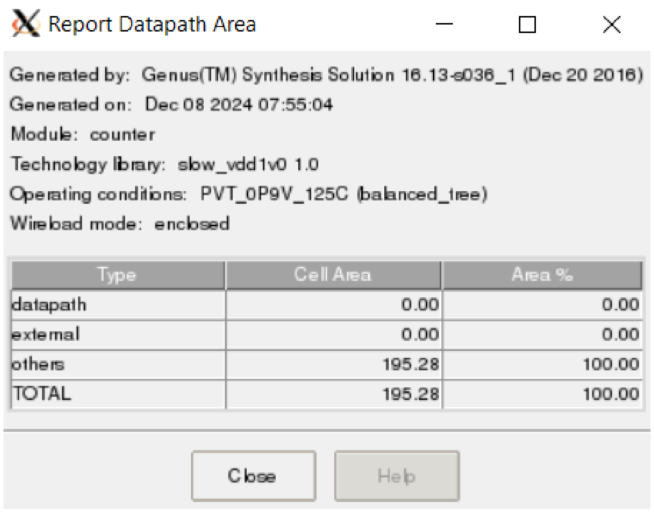


Report Statistics

Generated by: Genus(TM) Synthesis Solution 16.13-s036_1 (Dec 20 2016)
Generated on: Dec 08 2024 07:55:19
Module: counter
Technology library: slow_vdd1v0 1.0
Operating conditions: PVT_0P9V_125C (balanced_tree)
Wireload mode: enclosed

Type	Instances	Area	Area %
sequential	12	65.66	33.60
inverter	27	18.47	9.50
logic	62	111.15	56.90
physical_cells	0	0.00	0.00
TOTAL	101	195.28	100.00

Fig 33: Statistics Report for medium effort (Synthesis)

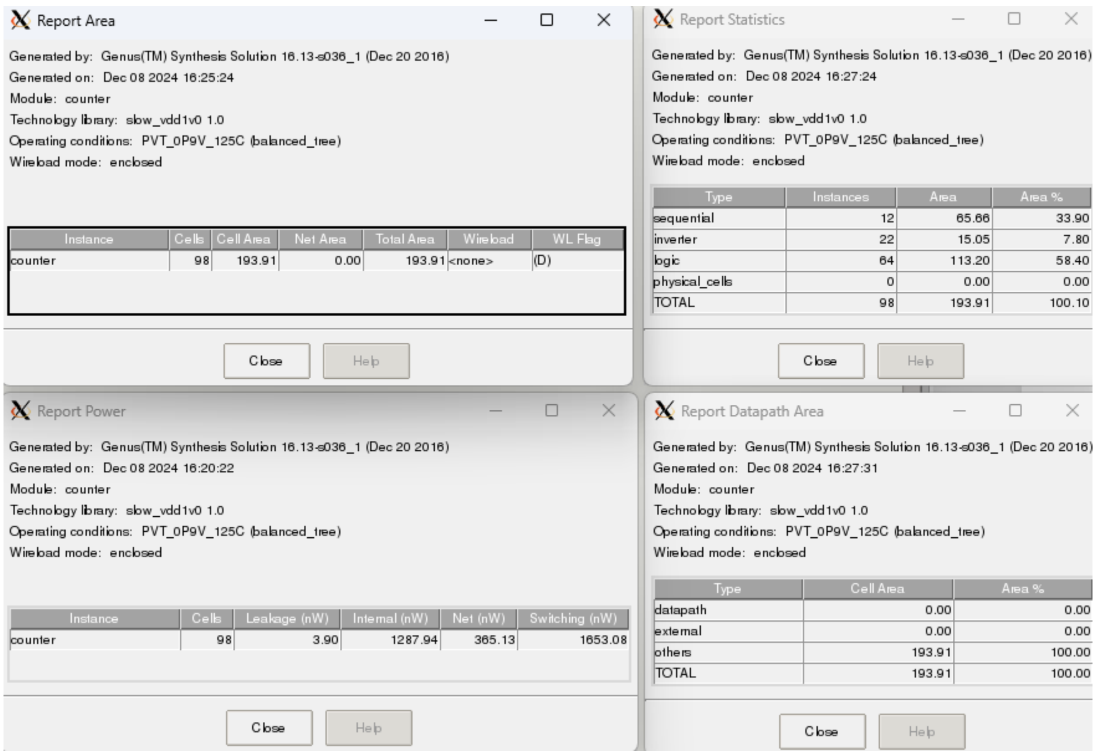

Report Datapath Area

Generated by: Genus(TM) Synthesis Solution 16.13-s036_1 (Dec 20 2016)
Generated on: Dec 08 2024 07:55:04
Module: counter
Technology library: slow_vdd1v0 1.0
Operating conditions: PVT_0P9V_125C (balanced_tree)
Wireload mode: enclosed

Type	Cell Area	Area %
datapath	0.00	0.00
external	0.00	0.00
others	195.28	100.00
TOTAL	195.28	100.00

Fig 34: Datapath Area Report for medium effort (Synthesis)

6.3 Final Optimization


Report Area

Generated by: Genus(TM) Synthesis Solution 16.13-s036_1 (Dec 20 2016)
Generated on: Dec 08 2024 16:25:24
Module: counter
Technology library: slow_vdd1v0 1.0
Operating conditions: PVT_0P9V_125C (balanced_tree)
Wireload mode: enclosed

Instance	Cells	Cell Area	Net Area	Total Area	Wireload	WL Flag
counter	98	193.91	0.00	193.91	<none>	(D)

Report Statistics

Generated by: Genus(TM) Synthesis Solution 16.13-s036_1 (Dec 20 2016)
Generated on: Dec 08 2024 16:27:24
Module: counter
Technology library: slow_vdd1v0 1.0
Operating conditions: PVT_0P9V_125C (balanced_tree)
Wireload mode: enclosed

Type	Instances	Area	Area %
sequential	12	65.66	33.90
inverter	22	15.05	7.80
logic	64	113.20	58.40
physical_cells	0	0.00	0.00
TOTAL	98	193.91	100.10

Report Power

Generated by: Genus(TM) Synthesis Solution 16.13-s036_1 (Dec 20 2016)
Generated on: Dec 08 2024 16:20:22
Module: counter
Technology library: slow_vdd1v0 1.0
Operating conditions: PVT_0P9V_125C (balanced_tree)
Wireload mode: enclosed

Instance	Cells	Leakage (nW)	Internal (nW)	Net (nW)	Switching (nW)
counter	98	3.90	1287.94	365.13	1653.08

Report Datapath Area

Generated by: Genus(TM) Synthesis Solution 16.13-s036_1 (Dec 20 2016)
Generated on: Dec 08 2024 16:27:31
Module: counter
Technology library: slow_vdd1v0 1.0
Operating conditions: PVT_0P9V_125C (balanced_tree)
Wireload mode: enclosed

Type	Cell Area	Area %
datapath	0.00	0.00
external	0.00	0.00
others	193.91	100.00
TOTAL	193.91	100.00

Fig 35: Reports for high effort (Synthesis)

Furthermore, the synthesis was done for all the same parameters while keeping the effort high. The high effort with optimized parameters gives the most optimized synthesis.

7 PnR Optimization

Design	Floorplan			Core to io boundaries				Density (%)	Routing overflow (%)		Power (uW)			
	Width	Height	Area	Core to Left	Core to Right	Core to Top	Core to Bottom		Horizontal	Vertical	Internal	Switching	Leakage	Total
v01	30	30.00	900.00	1.20	1.20	1.71	1.71	27.585	8.08	0	21.99087	7.15901	0.01063	29.16051
v02	25	19.95	498.75	1.20	1.20	1.71	1.71	56.146	0	0	21.99099	6.30091	0.01063	28.30253
v03	20	19.95	399.00	1.20	1.20	1.71	1.71	72.096	0	0	21.99293	6.00114	0.01063	28.0047
v04	18	20.00	360.00	0.70	0.70	0.70	0.70	67.976	0	0	21.99325	6.02615	0.01063	28.03003
v05	17	16.91	287.47	0.60	0.60	0.76	0.76	80.309	0	0	21.99262	5.93908	0.01063	27.94234

The table shows the parameters for each version of the design. The optimizations are explained as follows.

7.1 Optimization Plots

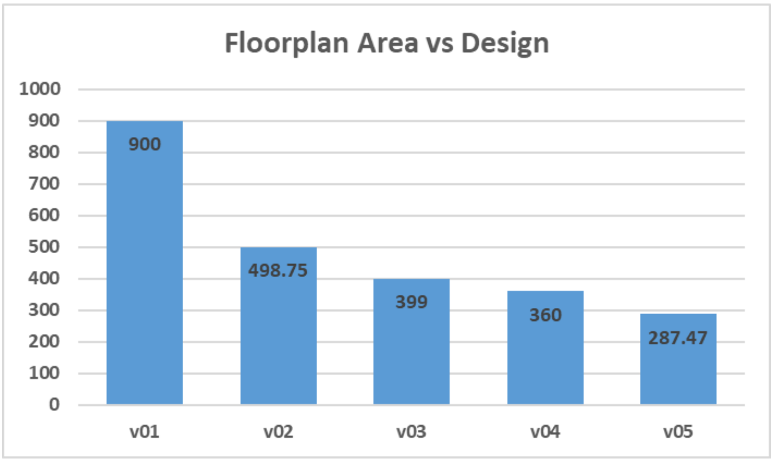


Fig 36: Floorplan Area vs Design

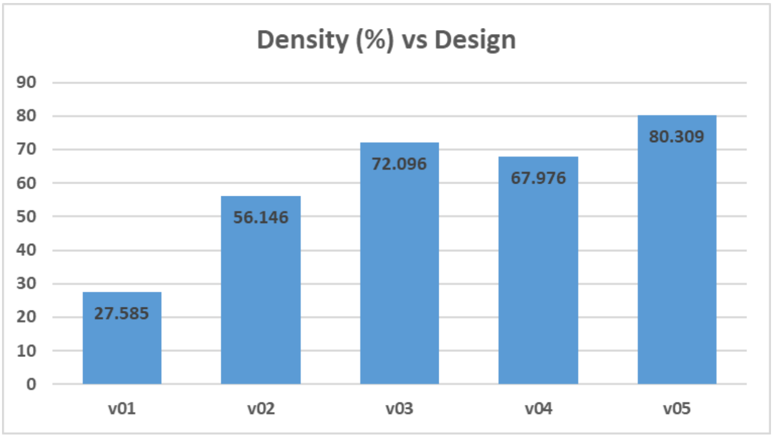


Fig 37: Density vs Design

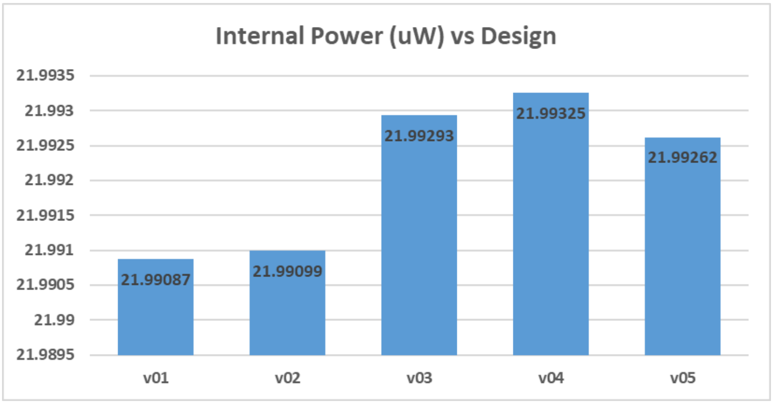


Fig 38: Internal Power vs Design

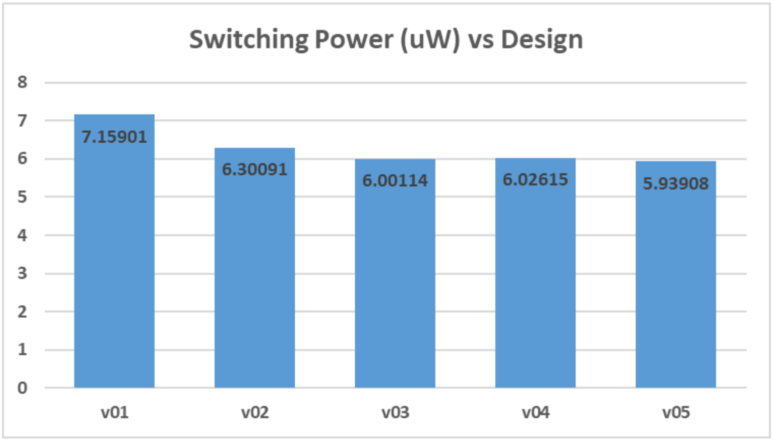


Fig 39: Switching Power vs Design

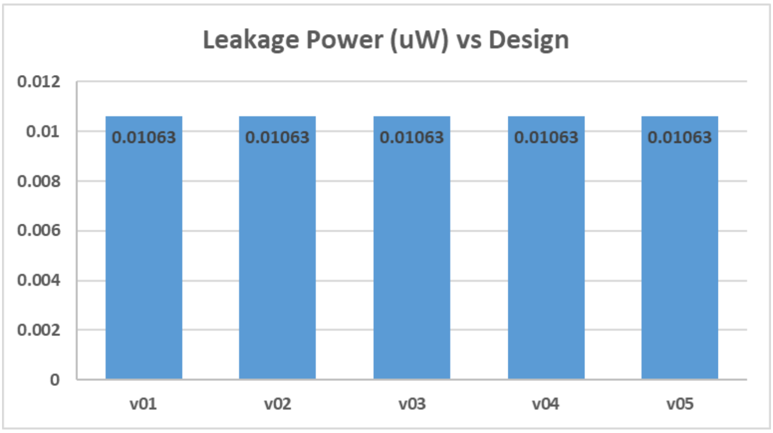


Fig 40: Leakage Power vs Design

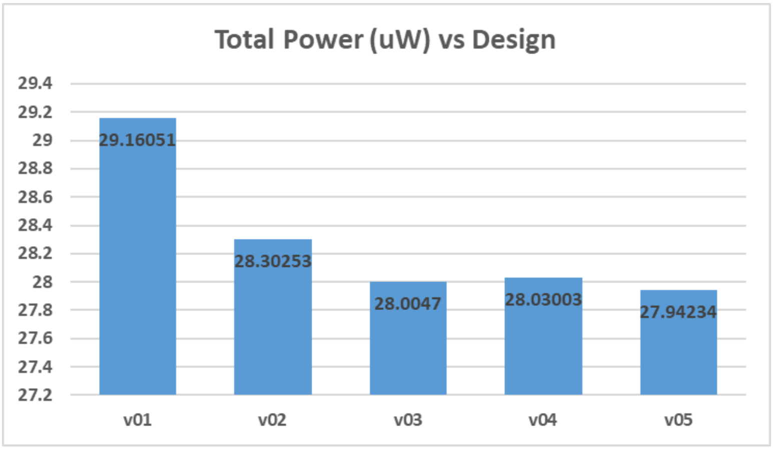


Fig 41: Total Power vs Design

- With each progressive design, the floorplan area is reduced, where design v05 has an area of 287.47 as opposed to the starting area of 900 for design v01.
- This corresponds to a 68% decrease in floorplan area. Furthermore, reduction in area results in a density improvement from 27.585% in design v01 to 80.309% in design v05.
- The internal power shows an increase with each design, but the increase is in the scale of nanowatts (~2.5 nW). As such, this increase is considered to be negligible.
- Switching power shows improvement by decreasing from 7.16 uW in design v01 to 5.94 uW in design v05.
- The leakage power remains constant at 10.63 nW throughout each design.
- Finally, the total power shows an overall decreasing trend as the designs progress, starting from 29.16 uW and settling at 27.94 uW.

8 Physical Design with PnR Optimization

The layouts using the optimized specifications are shown in this section.

8.1 Layout for v02

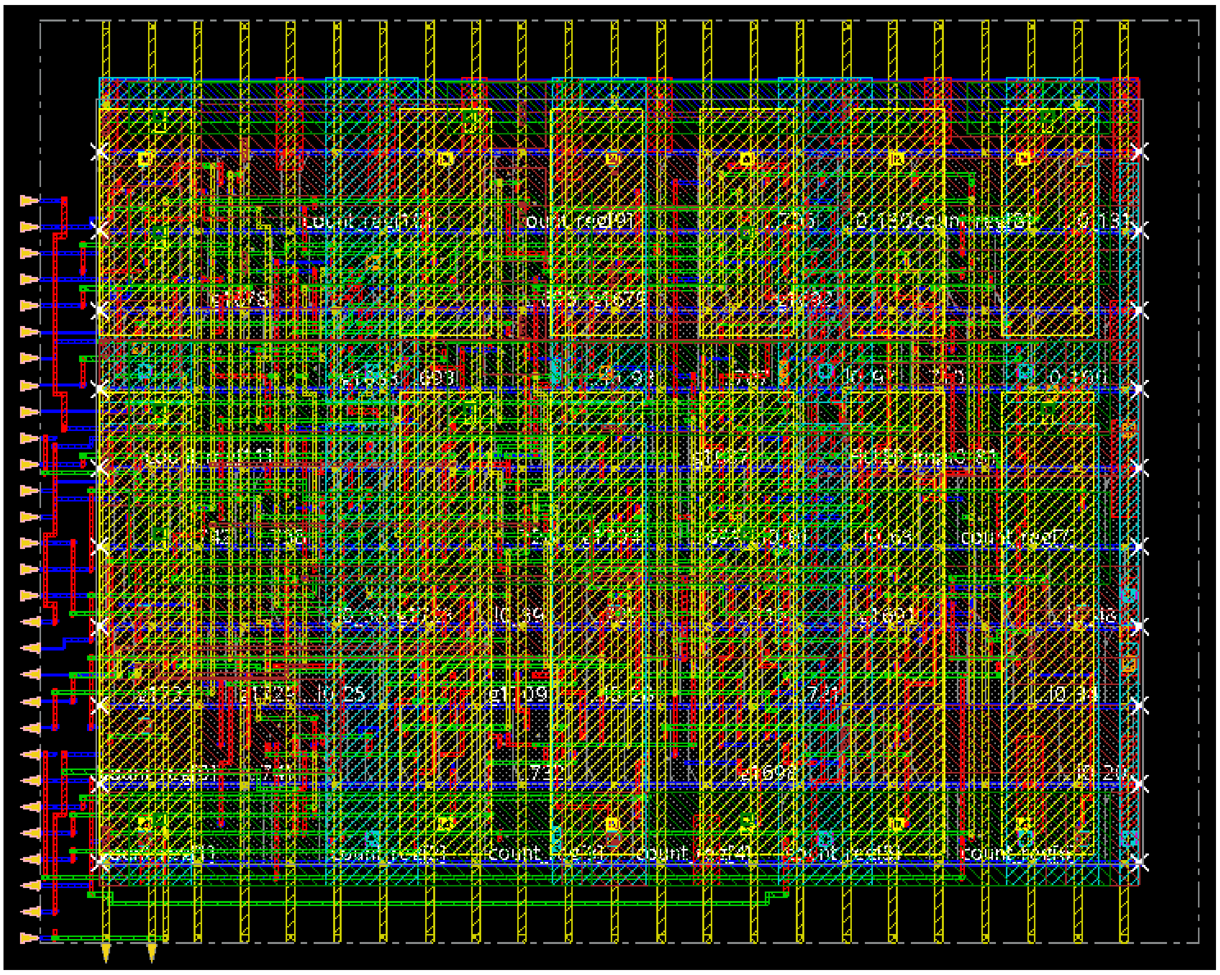
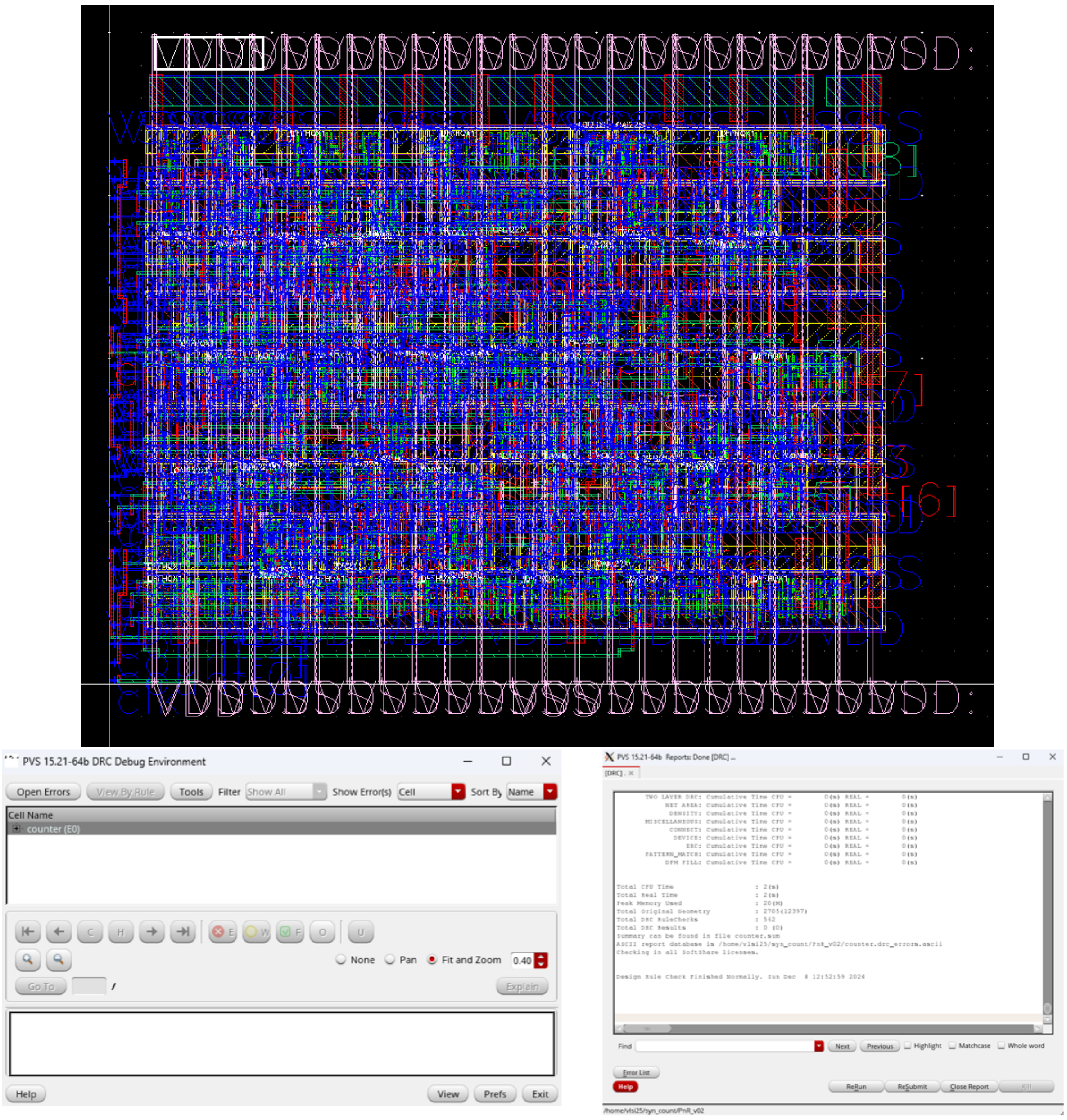


Fig 42: Layout for design v02

8.2 DRC Check for v02



```
       TWO LAYER DRC: Cumulative Time CPU =        0(s) REAL =        0(s)
            NET AREA: Cumulative Time CPU =        0(s) REAL =        0(s)
             DENSITY: Cumulative Time CPU =        0(s) REAL =        0(s)
       MISCELLANEOUS: Cumulative Time CPU =        0(s) REAL =        0(s)
             CONNECT: Cumulative Time CPU =        0(s) REAL =        0(s)
              DEVICE: Cumulative Time CPU =        0(s) REAL =        0(s)
                 ERC: Cumulative Time CPU =        0(s) REAL =        0(s)
       PATTERN_MATCH: Cumulative Time CPU =        0(s) REAL =        0(s)
            DFM FILL: Cumulative Time CPU =        0(s) REAL =        0(s)


Total CPU Time                    : 2(s)
Total Real Time                   : 2(s)
Peak Memory Used                  : 20(M)
Total Original Geometry           : 2705(12397)
Total DRC RuleChecks              : 562
Total DRC Results                 : 0 (0)
Summary can be found in file counter.sum
ASCII report database is /home/vlsi25/syn_count/PnR_v02/counter.drc_errors.ascii
Checking in all SoftShare licenses.


Design Rule Check Finished Normally. Sun Dec  8 12:52:59 2024
```

Fig 43: DRC check output for design v02

8.3 Final Optimized Layout for v05

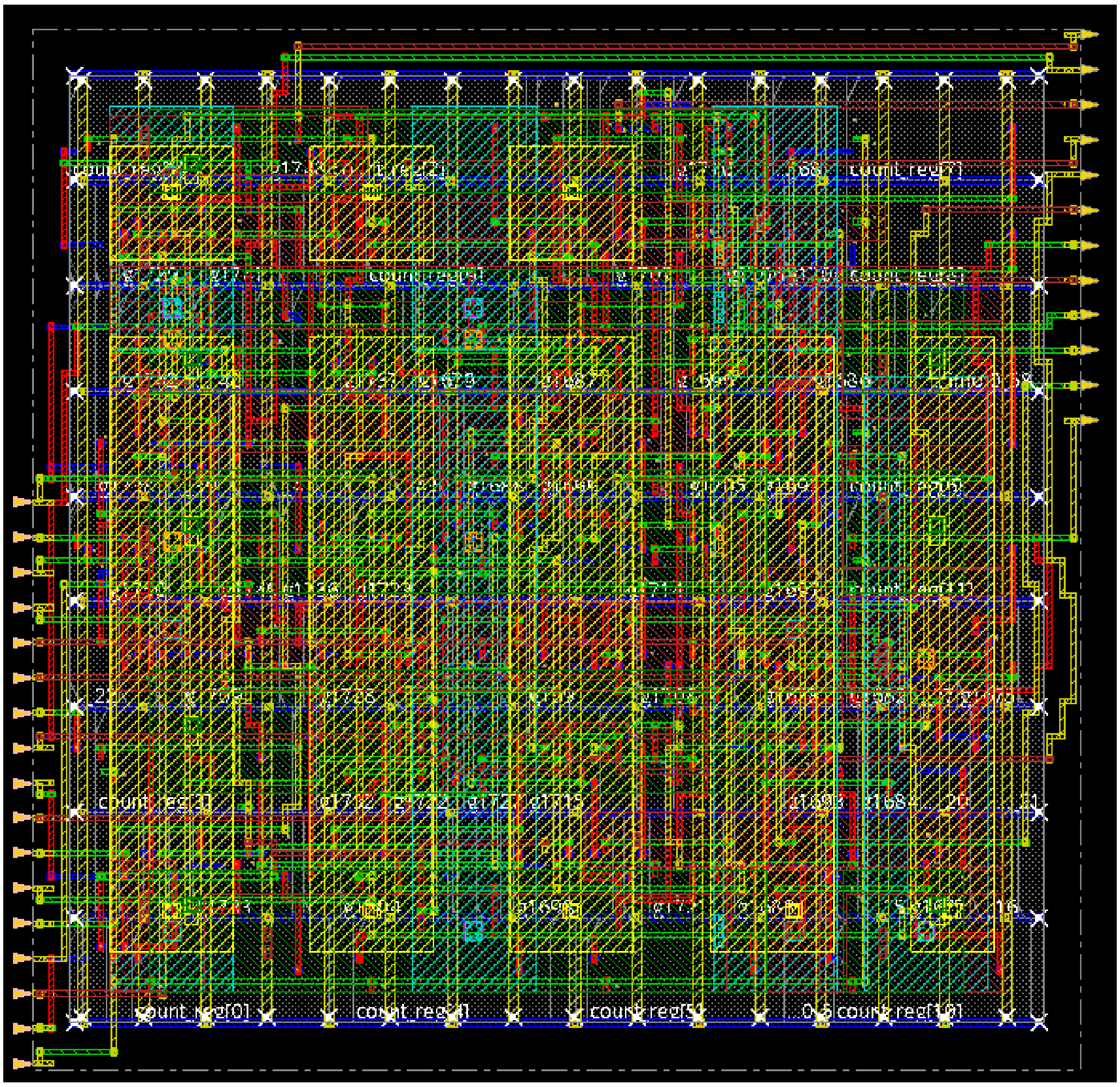


Fig 44: Layout for design v05

9 Reflection on Individual and Team work

Task	ID
Directed Testbench	1906141
Layered Testbench	1906140
Synthesis and Optimization	1906146
PnR and Optimization	1906167

10 Executive Summary

This project develops a 12-bit synchronous counter with up/down counting capabilities, controlled by key inputs: Load, Up/Down, Reset, and Enable. The Load signal allows loading a 12-bit value from the input bus (data[11:0]), while the Up/Down signal specifies the counting direction. The Reset input initializes the counter to zero, and the Enable input controls its operation, pausing the count when low. Synchronized with the clock's positive edge, the counter ensures precise performance. The project also encompasses directed and layered testbench creation, synthesis optimization, and physical design with PnR optimization for the counter.

11 References

[1] Sourabhshenoy. (n.d.-b). *GitHub - sourabhshenoy04/synch_mod12_counter: Date of Respository Creation: 05/08/2023. This project was undertaken as a part of 4th semester System Verilog Project.* GitHub. https://github.com/sourabhshenoy04/synch_mod12_counter

[2] GeeksforGeeks. (2024, September 9). *Counter Design using verilog HDL*. GeeksforGeeks. https://www.geeksforgeeks.org/counter-design-using-verilog-hdl/