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

EEE 468 (July 2023)
VLSI Laboratory

Final Project Report

Section: G1 Group: 08

16-bit Ripple Carry Adder

Course Instructors:
Nafis Sadik, Lecturer
Rafid Hassan 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: Tasmin Khan** **Student ID: 1906055**	**Signature:** _____ **Full Name: Tanvir Hossain** **Student ID: 1906040**

1 Contents

1 Abstract

Adders are digital circuits used for binary addition. They are fundamental in arithmetic operations, forming the basis of components like ALUs, multipliers, and processors. A 16- bit adder have two 16-bit inputs and a carry in and a 16-bit output sum and a carryout. The adder needs to be designed, verified, synthesized and we need to design the layout

2 Introduction

The half adder adds two single-bit inputs, A and B. The result is 0, 1, or 2, so two bits are required to represent the value; they are called the sum S and carry-out 'Cout'.
A full adder is a fundamental combinational logic circuit used in digital electronics to perform the addition of three binary inputs: two significant bits and a carry input Cin from a previous stage. It generates two outputs: the sum and the carry-out Cout. The sum represents the result of adding the three input bits, while the carry-out indicates whether there is an overflow (i.e., when the total exceeds the capacity of a single bit).

A	B	C	G	P	K	C_{out}	S
0	0	0	0	0	1	0	0
		1				0	1
0	1	0	0	1	0	0	1
		1				1	0
1	0	0	0	1	0	0	1
		1				1	0
1	1	0	1	0	0	1	0
		1				1	1

Figure 1.1: truth table for a full adder

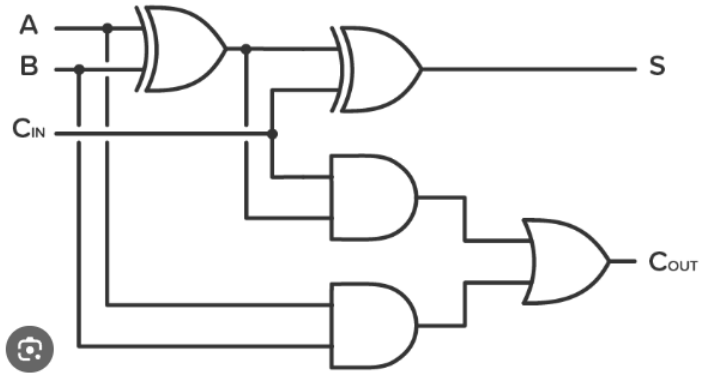


Figure 1.2: full adder circuit diagram

We have to design a 16 bit full adder with the propagate generate logic. For a full adder, it is sometimes useful to define Generate (G), Propagate (P), and Kill (K) signals. The adder generates a carry when Cout is true independent of Cin, so G = A · B. The adder propagates a carry; i.e., it produces a carry-out if and only if it receives a carry-in, when exactly one input is true: P = A xor B.
The full adder logic is:

$$
\begin{aligned}
S &= A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC \\
&= \left(A \oplus B\right) \oplus C = P \oplus C \\
C_{\text{out}} &= AB + AC + BC \\
&= AB + C\left(A + B\right) \\
&= \overline{\bar{A}\bar{B} + \bar{C}\left(\bar{A} + \bar{B}\right)} \\
&= \text{MAJ}(A,\ B,\ C)
\end{aligned}
$$

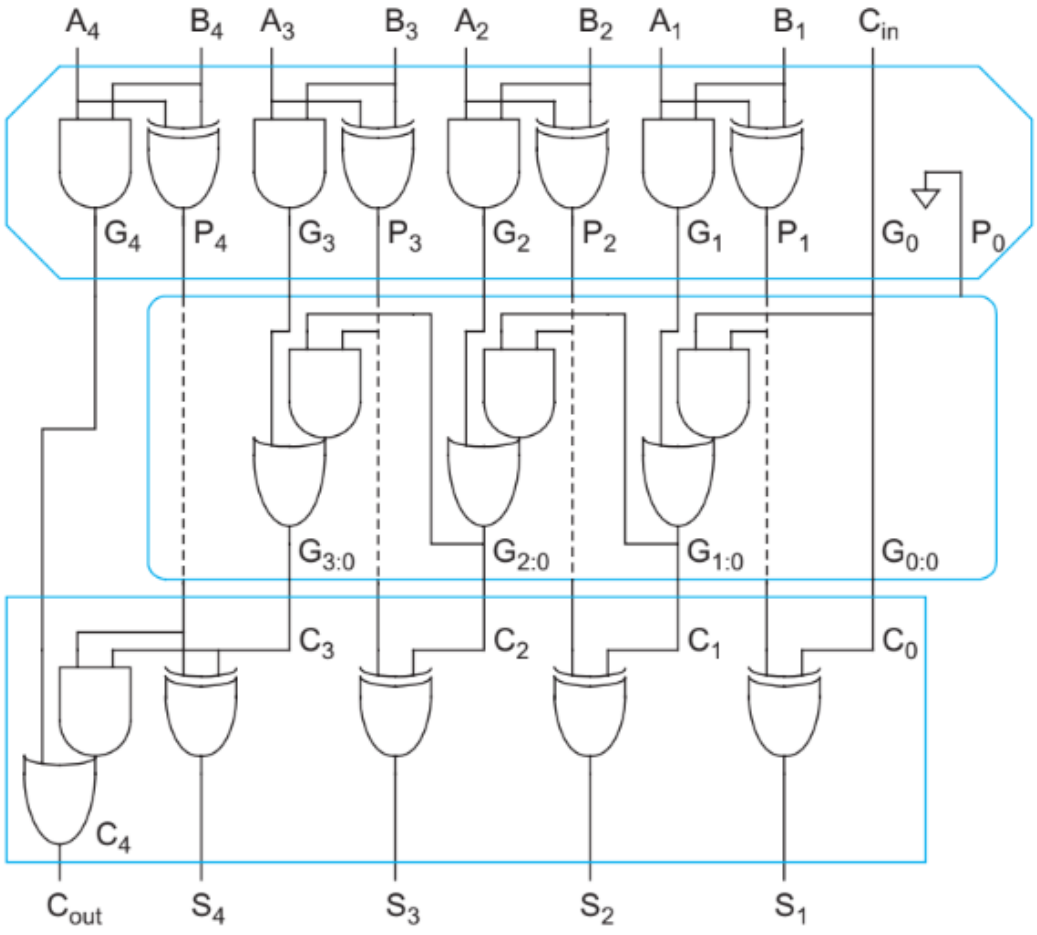
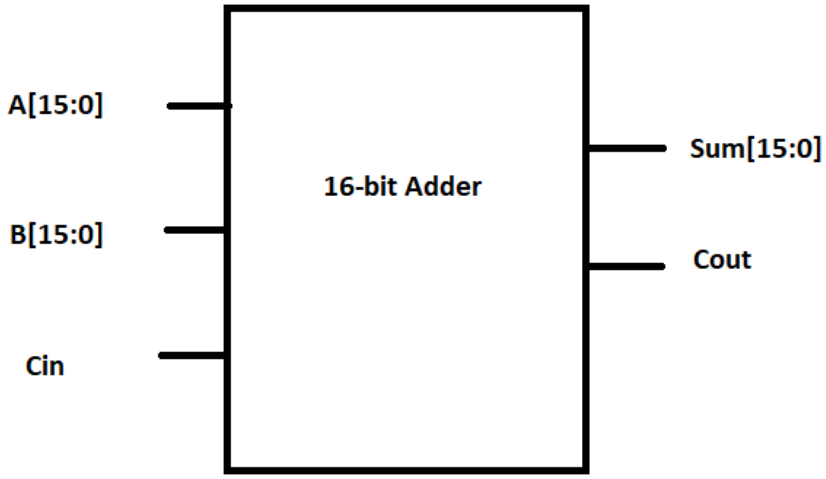


Figure 1.3: 4-bit carry-ripple adder using PG logic

The expressions for the carry in for the stages in terms of PG logic is as follows:

$$
\begin{aligned}
C_i &= A_iB_i + \left(A_i + B_i\right)C_{i-1} \\
&= A_iB_i + \left(A_i \oplus B_i\right)C_{i-1} \\
&= G_i + P_iC_{i-1}
\end{aligned}
$$

We have to design a 16-bit ripple carry adder. So, our Verilog module should look like below



3 Design

3.1 Problem Formulation

EEE 468 January 2024 Lab Projects

Project No.	Project Name	Inputs	Outputs	Additional Description
1	16-bit Ripple Carry Adder	First number **A [15:0]**, Second number **B [15:0]**, Carry in **C-in**	Sum **S [15:0]**, Carry out **Cout**	Encouraged to follow Propagate Generate Logic

3.2 Design Method

1. Design Code
2. Verification using Directed and Layered Testbench
3. RTL Synthesis
4. Power and Area Optimization
5. PnR Analysis
6. PnR Optimization

3.3 Tools Used

Cadence CAD Tools:

1. NCSim for simulation
2. Simvision for visualization
3. Genus for RTL synthesis
4. Innovus for PnR

4 Design

4.1 Verilog Code

Ripple Carry Adder Module without clock:

```
module RippleCarryAdder #(parameter int N = 16) (
    input logic [N-1:0] A,
    input logic [N-1:0] B,
    input logic Cin,
    output logic [N-1:0] S,
    output logic Cout
);
    logic [N-1:0] C;

    genvar i;
    generate
        for (i = 0; i < N; i++) begin
            if (i == 0)
                fulladder f(.a(A[0]), .b(B[0]), .cin(Cin), .s(S[0]), .co(C[0]));
            else
                fulladder f(.a(A[i]), .b(B[i]), .cin(C[i-1]), .s(S[i]), .co(C[i]));
        end
    endgenerate

    assign Cout = C[N-1];

endmodule

module fulladder(
    input logic a, b, cin,
    output logic s, co
);
    logic p, g;

    assign p = a ^ b;
    assign g = a & b;
    assign s = p ^ cin;
    assign co = g | (p & cin);

endmodule

```

Ripple Carry Adder Module with Clock:

```
module RippleCarryAdder (
  input logic [16-1:0] A,
  input logic [16-1:0] B,
  input logic Cin, clk,
  output logic [16-1:0] S,
  output logic Cout
);

  logic [16-1:0] P, G;
  logic [16:0] C;

    always_ff @(posedge clk) begin
      C[0] = Cin;
      for (int i = 0; i < 16; i++) begin
            P[i] = A[i] ^ B[i];
            G[i] = A[i] & B[i];
            C[i+1] = G[i] | (P[i] & C[i]);
            S[i] = P[i] ^ C[i];
        end
        Cout = C[16];
     end


endmodule

```

The module with clock is used to get the results at the positive edge of the clock also following the Propagate Generate logic.

5 Testbench

5.1 Flat Testbench

Code:

```
  initial begin

    Cin=0;
    repeat(j) begin
      A=$random;
      B=$random;
      #10; {A,B} = {A,B} + 1;
    end
    Cin=1;
    repeat(j) begin
      A=$random;
      B=$random;
      #10; {A,B} = {A,B} + 1;
    end
    end

  initial begin
    $dumpfile("RippleCarryAdder.vcd");
    $dumpvars(0,RippleCarryAdder_tb);
    $monitor($time, ": %b + %b + %b = %b, %b", A, B, Cin, Cout, S);
    #320;
    $finish;
    end
```

Output in binary:

```
  0: 0011010100100100 + 0101111010000001 + 0 = 0, 1001001110100101
 10: 1101011000001001 + 0101011001100011 + 0 = 1, 0010110001101100
 20: 0111101100001101 + 1001100110001101 + 0 = 1, 0001010010011010
 30: 1000010001100101 + 0101001000010010 + 0 = 0, 1101011001110111
 40: 1110001100000001 + 1100110100001101 + 0 = 1, 1011000000001110
 50: 1111000101110110 + 1100110100111101 + 0 = 1, 1011111010110011
 60: 0101011111101101 + 1111011110001100 + 0 = 1, 0100111101111001
 70: 1110100111111001 + 0010010011000110 + 0 = 1, 0000111010111111
 80: 1000010011000101 + 1101001010101010 + 0 = 1, 0101011101101111
 90: 1111011111100101 + 0111001001110111 + 0 = 1, 0110101001011100
100: 1101011000010010 + 1101101110001111 + 1 = 1, 1011000110100010
110: 0110100111110010 + 1001011011001110 + 1 = 1, 0000000011000001
120: 0111101011101000 + 0100111011000101 + 1 = 0, 1100100110101110
130: 0100100101011100 + 0010100010111101 + 1 = 0, 0111001000011010
140: 0101100000101101 + 0010011001100101 + 1 = 0, 0111111010010011
150: 0110001001100011 + 1000011100001010 + 1 = 0, 1110100101101110
160: 0010001010000000 + 0010000100100000 + 1 = 0, 0100001110100001
170: 0100010110101010 + 1100110010011101 + 1 = 1, 0001001001001000
180: 0011111010010110 + 1011100000010011 + 1 = 0, 1111011010101010
190: 0011100000001101 + 1101011001010011 + 1 = 1, 0000111001100001
200: 0011100000001101 + 1101011001010100 + 1 = 1, 0000111001100010
```

Flat Testbench Result printed in decimal:

```
# KERNEL: SLP loading done - time: 0.0 [s].
# KERNEL: Warning: You are using the Riviera-PRO EDU Edition. The performance of simulation is reduced.
# KERNEL: Warning: Contact Aldec for available upgrade options - sales@aldec.com.
# KERNEL: SLP simulation initialization done - time: 0.0 [s].
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 586 kB (elbread=428 elab2=21 kernel=135 sdf=0)
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# KERNEL:                   0: 13604 + 24193 + 0 = 0, 37797
# KERNEL:                  10: 54793 + 22115 + 0 = 1, 11372
# KERNEL:                  20: 31501 + 39309 + 0 = 1,  5274
# KERNEL:                  30: 33893 + 21010 + 0 = 0, 54903
# KERNEL:                  40: 58113 + 52493 + 0 = 1, 45070
# KERNEL:                  50: 61814 + 52541 + 1 = 1, 48820
# KERNEL:                  60: 22509 + 63372 + 1 = 1, 20346
# KERNEL:                  70: 59897 +  9414 + 1 = 1,  3776
# KERNEL:                  80: 33989 + 53930 + 1 = 1, 22384
# KERNEL:                  90: 63461 + 29303 + 1 = 1, 27229
# KERNEL:                 100: 63461 + 29304 + 1 = 1, 27230
# RUNTIME: Info: RUNTIME_0068 testbench.sv (39): $finish called.
# KERNEL: Time: 320 ns,  Iteration: 0,  Instance: /RippleCarryAdder_tb,  Process: @INITIAL#34_2@.
# KERNEL: stopped at time: 320 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
```

5.2 Layered Testbench:

Testbench module:

```
`include "testcase01.sv"
`include "interface.sv"

module testbench;

  bit clk;

  initial begin
    forever #10 clk=~clk;
  end

  int count = `test_case_count;
  RCA_interface realRCA(clk);
  test test01(count, realRCA); //passing the interface to the testclass

  RippleCarryAdder #(`N) DUT(//here passing the dut connections to the interface
    .A(realRCA.A),
    .B(realRCA.B),
    .Cin(realRCA.Cin),
    .S(realRCA.S),
    .Cout(realRCA.Cout)
  );


  covergroup RCA_coverage;
    option.per_instance = 1;
    a: coverpoint realRCA.A {
      bins all_values[`bin_count] = {[0:(2**`N)-1]};
    }

    b: coverpoint realRCA.B {
      bins all_values[`bin_count] = {[0:(2**`N)-1]};
    }

    c: coverpoint realRCA.Cin {
      bins bin_0 = {0};
      bins bin_1 = {1};
    }

    total: cross realRCA.A, realRCA.B, realRCA.Cin {
      bins all_values[`bin_count] = {[]};
    }
  endgroup



  RCA_coverage cg;
  initial begin
    cg = new();
  end

  always @(posedge clk) begin;
    cg.sample();
  end

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


  final begin
    $display("Coverage Results:");
    $display("Total Coverage: %f%% with %0d test combinations", cg.get_coverage(), count);

    $display("Coverpoint A Coverage: %0.2f%%", cg.a.get_inst_coverage());
    $display("Coverpoint B Coverage: %0.2f%%", cg.b.get_inst_coverage());
    $display("Coverpoint Cin Coverage: %0.2f%%", cg.c.get_inst_coverage());
    $display("Cross Coverage (A x B x Cin): %0.2f%%", cg.total.get_inst_coverage());

    $finish;
  end

endmodule
```

Here we introduced coverage group with cover point for each input signal as well as a cross coverage for all the signals. We also took sampling for the coverpoints at each posedhe of the clock.

Interface.sv

```
interface RCA_interface (input clk);

  //input
  logic [`N-1:0] A, B;
  logic Cin;
  //output
  logic [`N-1:0] S;
  logic Cout;


  //driver block
  clocking driver_cb @(negedge clk);
    default input #3 output #2;
    output A,B,Cin;
  endclocking

  //monitor block
  clocking monitor_cb @(posedge clk);
    default input #3 output #2;
    input A,B,Cin,S,Cout;
  endclocking

  modport DRIVER(clocking driver_cb, input clk);
  modport MONITOR(clocking monitor_cb, input clk);

endinterface
```

Driver Class:

```
  class driver;

    //this class first recieves the mail from gen, then drives output signal
  based on the mail, puts the mail out for sb and raises a flag.
    mailbox #(transaction) gen2driv, driv2sb;
    virtual RCA_interface.DRIVER vRCA;
    transaction d_trans;
    event driven;


    function new(mailbox #(transaction) gen2driv, driv2sb, virtual
  RCA_interface.DRIVER vRCA, event driven);
      this.gen2driv = gen2driv;
      this.driv2sb = driv2sb;
      this.vRCA = vRCA;
      this.driven = driven;
    endfunction


    task main(input int count);

      repeat(count) begin
        d_trans = new();
        gen2driv.get(d_trans);

        @(vRCA.driver_cb);
        vRCA.driver_cb.A <= d_trans.A;
        vRCA.driver_cb.B <= d_trans.B;
        vRCA.driver_cb.Cin <= d_trans.Cin;

        driv2sb.put(d_trans);
        ->driven;
      end

    endtask
  endclass
```

Scoreboard.sv

```
class scoreboard;

  mailbox #(transaction) driv2sb;
  mailbox #(transaction) mon2sb;
  transaction d_trans;
  transaction m_trans;

  function new(mailbox #(transaction) driv2sb, mailbox #(transaction) mon2sb);
    this.driv2sb = driv2sb;
    this.mon2sb = mon2sb;
  endfunction

  task main(input int count);
    $display("---Scoreboard Test Starts-----------");
    repeat(count) begin
      infer();
      m_trans = new();
      mon2sb.get(m_trans);
      report();
    end
    $display("------Scoreboard Test Ends-----------------");
  endtask: main

  task infer();
    d_trans = new();
    driv2sb.get(d_trans);
    {d_trans.Cout, d_trans.S} = d_trans.A + d_trans.B + d_trans.Cin;
  endtask


  task report();
    if ((m_trans.S != d_trans.S) || (m_trans.Cout != d_trans.Cout)) begin
      $display("Failed: A=%d + B=%d + Cin=%d = Cout=%d + S=%d  Expected out = Cout=%d +
S=%d ", d_trans.A, d_trans.B, d_trans.Cin, m_trans.Cout, m_trans.S, d_trans.Cout,
d_trans.S);
    end

  endtask: report
endclass: scoreboard
```

Monitor Class

```
class monitor;

  mailbox #(transaction) mon2sb;
  virtual RCA_interface.MONITOR vRCA;
  transaction m_trans;
  event driven;

  function new(mailbox #(transaction) mon2sb, virtual RCA_interface.MONITOR
vRCA, event driven);
    this.mon2sb = mon2sb;
    this.vRCA = vRCA;
    this.driven = driven;
  endfunction

  task main(input int count);
    //@(driven);
    //@(vRCA.monitor_cb);
    repeat(count) begin
      m_trans = new();
      //@(posedge vRCA.clk);
      @(driven);
      @(vRCA.monitor_cb);
      m_trans.Cout = vRCA.monitor_cb.Cout;
      m_trans.S = vRCA.monitor_cb.S;
      mon2sb.put(m_trans);
    end
  endtask
 endclass
```

Generator Class:

```
`include "transaction.sv"

class generator;
  mailbox #(transaction) gen2driv;
  transaction g_trans, custom_trans;
  transaction copy_trans;

  function new(mailbox #(transaction) gen2driv);
    this.gen2driv = gen2driv;
  endfunction

  task main(input int count);
    g_trans = new();
    g_trans = new custom_trans;

    repeat(count) begin
      void'(g_trans.randomize());

      copy_trans = new();
      copy_trans = new g_trans;

      gen2driv.put(copy_trans);

    end

  endtask

endclass
```

Enviroment Class:

```
`include "environment.sv"

//program block is used to avoid race conditions by being run after module
block. Ensures the testbench (program) logic always runs after the DUT logic
(module) in the same simulation timestep.

program test (input int count, RCA_interface realRCA);
  environment env;

  class testcase01 extends transaction;
  endclass

  initial begin
    testcase01 testcase01handle;
    testcase01handle = new();

    env = new(realRCA);
    env.gen.custom_trans = testcase01handle;
    //here we are passing the constraints through the custom_trans class as
said in the generator block
    env.main(count);
  end

endprogram
```

Transaction.sv

```
`include "paramsmacro.svh"

class transaction;

  rand bit [`N-1:0] A;
  rand bit [`N-1:0] B;
  rand bit Cin;
  bit [`N-1:0] S;
  bit Cout;


  int unsigned A_weights[0:1023];
  int unsigned B_weights[0:1023];

  function new();
    A_weights = '{default: `DEFAULT_WEIGHT};
    B_weights = '{default: `DEFAULT_WEIGHT};
  endfunction

  `include "consss.svh"
  `include "randomization.svh"

endclass
```

Constraint Header File

```
constraint weight {
    A dist {
      [0    : 63]      :/ A_weights[0],
      [64   : 127]     :/ A_weights[1],
      [128  : 191]     :/ A_weights[2],
      [192  : 255]     :/ A_weights[3],
      [256  : 319]     :/ A_weights[4],
      [320  : 383]     :/ A_weights[5],
      [384  : 447]     :/ A_weights[6],
      [448  : 511]     :/ A_weights[7],
      [512  : 575]     :/ A_weights[8],
      [576  : 639]     :/ A_weights[9],
      [640  : 703]     :/ A_weights[10],
      [704  : 767]     :/ A_weights[11],
      [768  : 831]     :/ A_weights[12],
      [832  : 895]     :/ A_weights[13],
      [896  : 959]     :/ A_weights[14],
      [960  : 1023]    :/ A_weights[15],
      [1024 : 1087]    :/ A_weights[16],
      [1088 : 1151]    :/ A_weights[17],
      [1152 : 1215]    :/ A_weights[18],
      [1216 : 1279]    :/ A_weights[19],
      [1280 : 1343]    :/ A_weights[20],
      [1344 : 1407]    :/ A_weights[21],
      [1408 : 1471]    :/ A_weights[22],
      [1472 : 1535]    :/ A_weights[23],
      [1536 : 1599]    :/ A_weights[24],
```

```
1025        [65408 : 65471]   :/ A_weights[1022],
1026        [65472 : 65535]   :/ A_weights[1023]
1027      };
1028
1029        B dist {
1030        [0    : 63]       :/ B_weights[0],
1031        [64   : 127]      :/ B_weights[1],
1032        [128  : 191]      :/ B_weights[2],
1033        [192  : 255]      :/ B_weights[3],
1034        [256  : 319]      :/ B_weights[4],
1035        [320  : 383]      :/ B_weights[5],
1036        [384  : 447]      :/ B_weights[6],
1037        [448  : 511]      :/ B_weights[7],
1038        [512  : 575]      :/ B_weights[8],
1039        [576  : 639]      :/ B_weights[9],
1040        [640  : 703]      :/ B_weights[10],
1041        [704  : 767]      :/ B_weights[11],
1042        [768  : 831]      :/ B_weights[12],
1043        [832  : 895]      :/ B_weights[13],
1044        [896  : 959]      :/ B_weights[14],
1045        [960  : 1023]     :/ B_weights[15],
1046        [1024 : 1087]     :/ B_weights[16],
1047        [1088 : 1151]     :/ B_weights[17],
1048        [1152 : 1215]     :/ B_weights[18],
1049        [1216 : 1279]     :/ B_weights[19],
1050        [1280 : 1343]     :/ B_weights[20],
```

We declared the transaction class with randomization function for each input signal and a distribution weight for each bin. We took 1024 bins and so there were 1024 weights declared for them, We initially started all the bins with the same weight and then reduced the weight by a defined penalty number each time a number was produced from that bin. A conditional statement was checked beforehand to ensure no negative weight is not introduced. Another conditional statement is implemented to reset the weights when all the weights hit the minimum.
This way the distribution is skewed making the probability of the bin that is not hit, higher after every iteration. As a result, achieving coverage is now possible with fewer epochs,
This distribution skewing is done by declaring a pre randomization function that is invoked automatically every time before randomization function is called. We also introduced a conditional statement to avoid cases where no value has been assigned to A or B yet to skip unnecessary distribution skewing.
The code for the pre randomization function and the result of skewing after each iteration is shown in the next page.

Pre-Randomization Header File

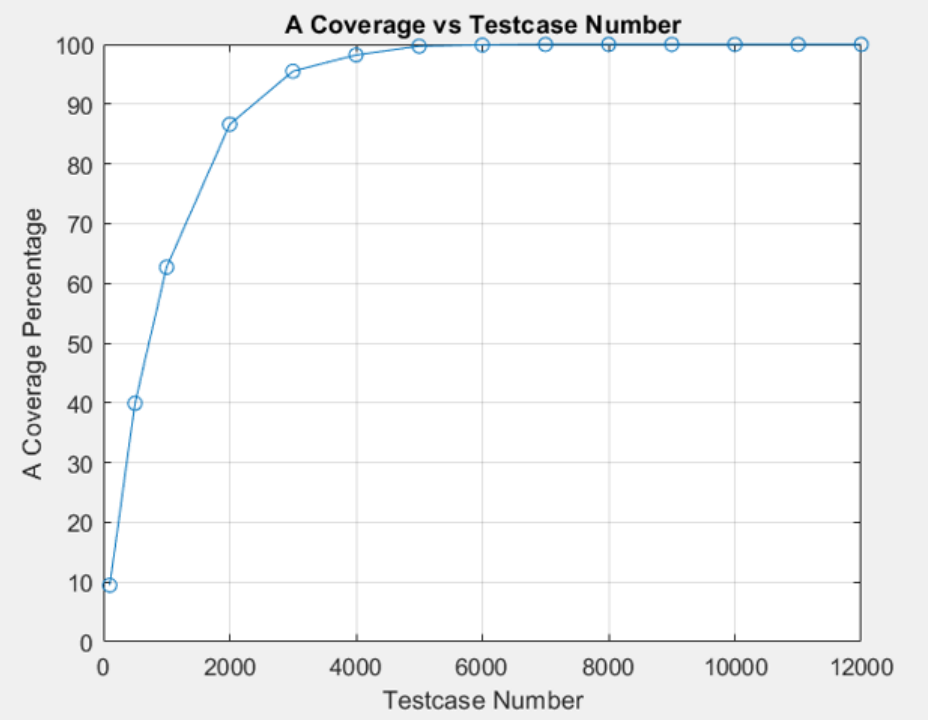
```
function bit check_all_weights_min();
    foreach (A_weights[i]) begin
      if (A_weights[i] != 0)
          return 0;
    end
    foreach (B_weights[i]) begin
      if (B_weights[i] != 0)
          return 0;
    end
    return 1;
endfunction


function void reset_weights();
    A_weights = '{default: `DEFAULT_WEIGHT};
    B_weights = '{default: `DEFAULT_WEIGHT};
endfunction

function void pre_randomize();
    int index_A;
    int index_B;

    // Skip adjustment if A or B is uninitialized
    if ((A === 'x || A === 'z) || (B === 'x || B === 'z)) begin
        return;
    end

    // Determine which range the current value falls into
    for (int i = 0; i < 1024; i++) begin
        if (A >= i * 64 && A < (i + 1) * 64) begin
            index_A = i;
            break;
        end
    end

    for (int i = 0; i < 1024; i++) begin
        if (B >= i * 64 && B < (i + 1) * 64) begin
            index_B = i;
            break;
        end
    end

  if (A_weights[index_A] >= `penalty) begin
      A_weights[index_A] -=`penalty; end
  else if (A_weights[index_A] !=0) begin
        A_weights[index_A] =0; end
   if (B_weights[index_B] >= `penalty) begin
        B_weights[index_B] -= `penalty; end
   else if (B_weights[index_B] !=0) begin
        B_weights[index_B] =0; end

    // Check if all weights are 1 or less, reset if true
    if (check_all_weights_min()) begin
        reset_weights();
    end

 endfunction
```

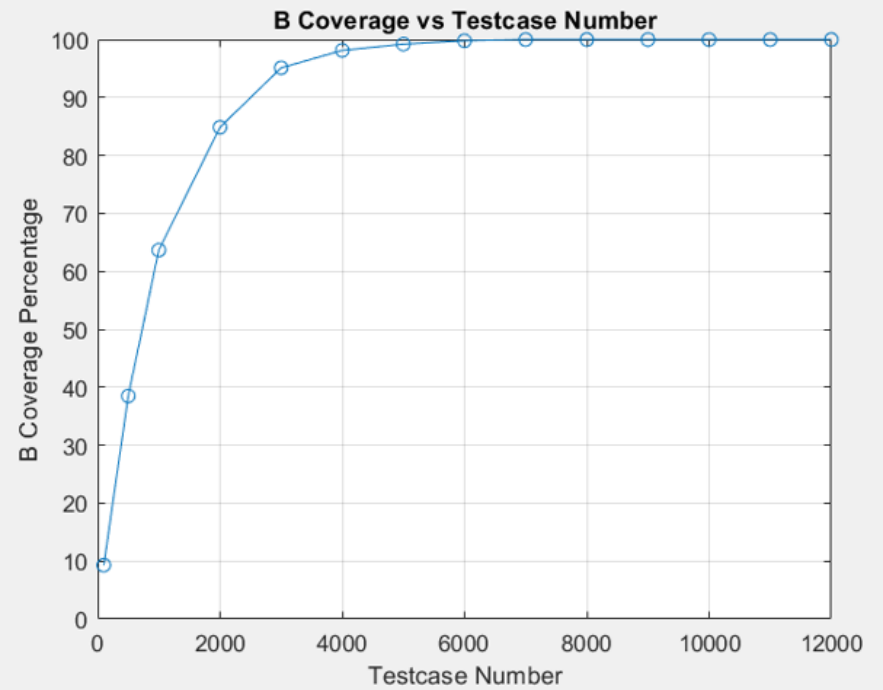
Pre-Randomization

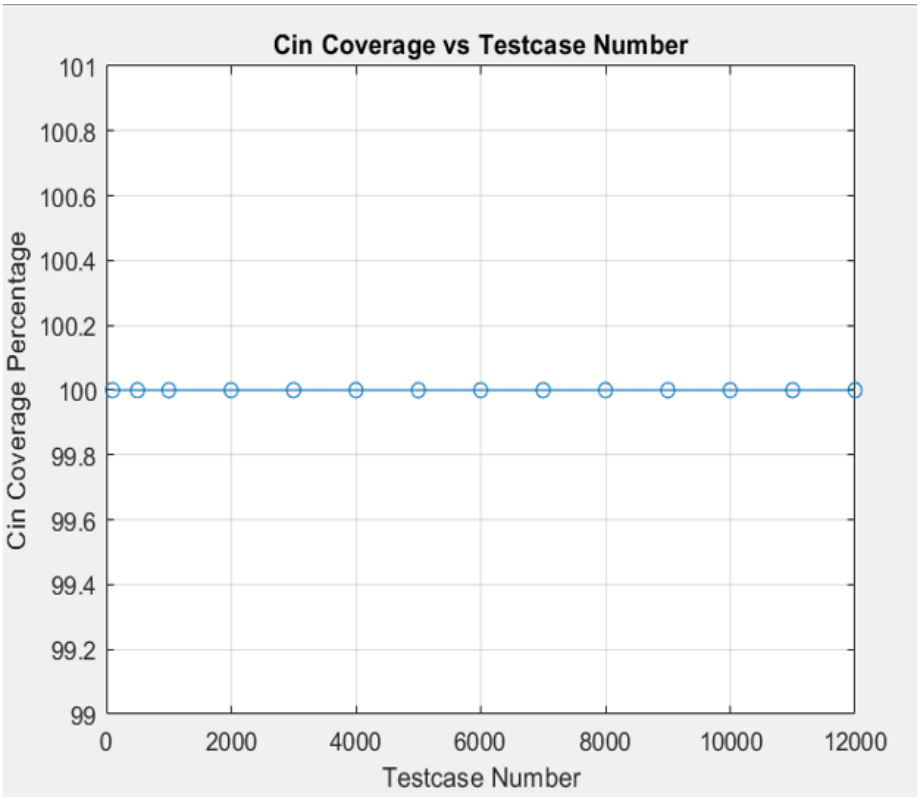
```
# KERNEL: Pre-Randomize: Current A_weights:
# KERNEL:    Range 0: Weight = 100
# KERNEL:    Range 1: Weight = 100
# KERNEL:    Range 2: Weight = 100
# KERNEL:    Range 3: Weight = 100
# KERNEL: Pre-Randomize: Current A_weights:
# KERNEL:    Range 0: Weight = 99
# KERNEL:    Range 1: Weight = 100
# KERNEL:    Range 2: Weight = 100
# KERNEL:    Range 3: Weight = 100
# KERNEL: Pre-Randomize: Current A_weights:
# KERNEL:    Range 0: Weight = 99
# KERNEL:    Range 1: Weight = 100
# KERNEL:    Range 2: Weight = 100
# KERNEL:    Range 3: Weight = 99
# KERNEL: Pre-Randomize: Current A_weights:
# KERNEL:    Range 0: Weight = 99
# KERNEL:    Range 1: Weight = 100
# KERNEL:    Range 2: Weight = 99
# KERNEL:    Range 3: Weight = 99
# KERNEL: Pre-Randomize: Current A_weights:
# KERNEL:    Range 0: Weight = 98
# KERNEL:    Range 1: Weight = 100
# KERNEL:    Range 2: Weight = 99
# KERNEL:    Range 3: Weight = 99
```

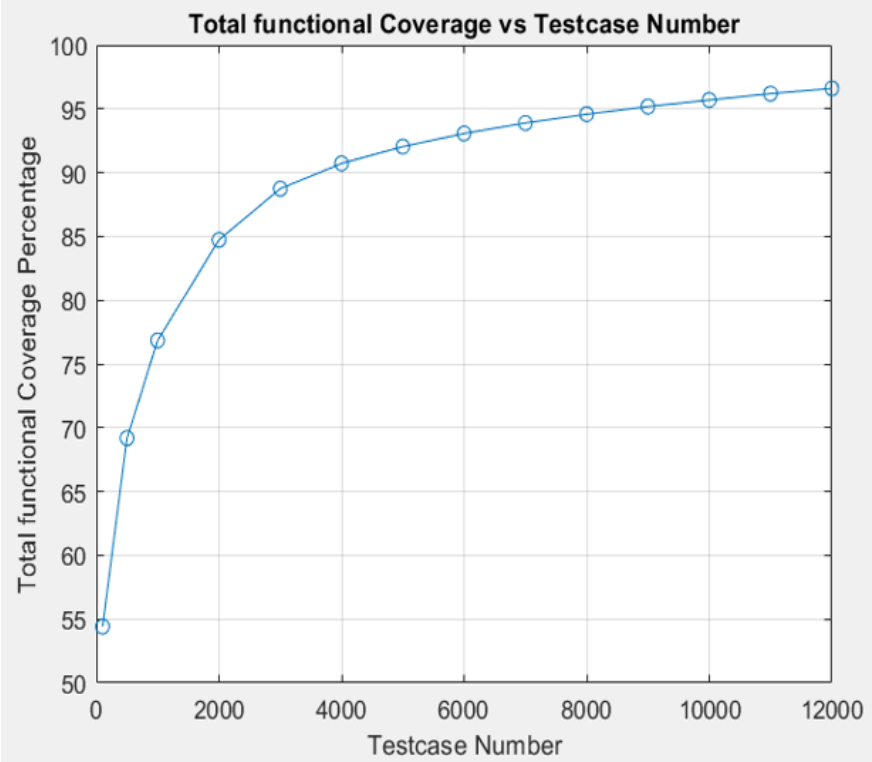
```
Pre-Randomize: Current A_weights:
  Range 0: Weight = 97
  Range 1: Weight = 100
  Range 2: Weight = 99
  Range 3: Weight = 99
Pre-Randomize: Current A_weights:
  Range 0: Weight = 97
  Range 1: Weight = 100
  Range 2: Weight = 98
  Range 3: Weight = 99
Pre-Randomize: Current A_weights:
  Range 0: Weight = 96
  Range 1: Weight = 100
  Range 2: Weight = 98
  Range 3: Weight = 99
Pre-Randomize: Current A_weights:
  Range 0: Weight = 96
  Range 1: Weight = 99
  Range 2: Weight = 98
  Range 3: Weight = 99
Pre-Randomize: Current A_weights:
  Range 0: Weight = 96
  Range 1: Weight = 98
  Range 2: Weight = 98
  Range 3: Weight = 99
```

Testbench Coverage plots:

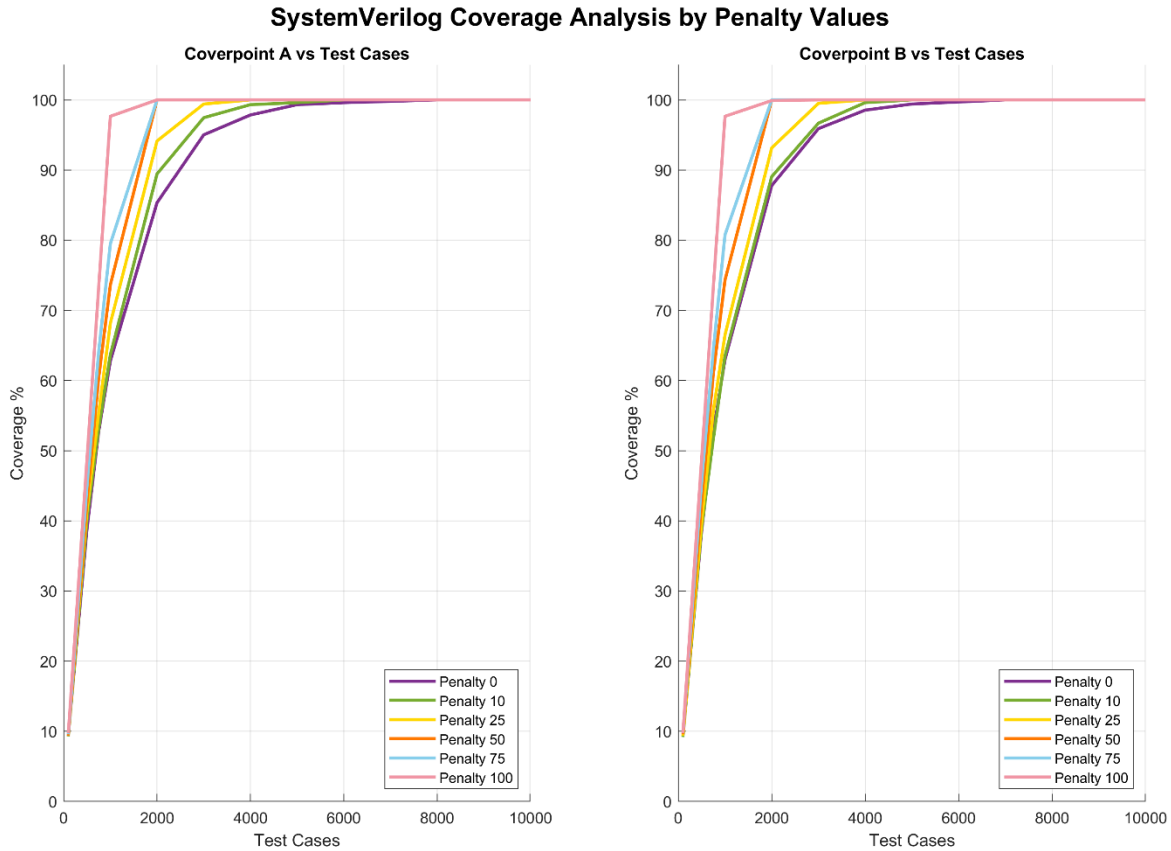


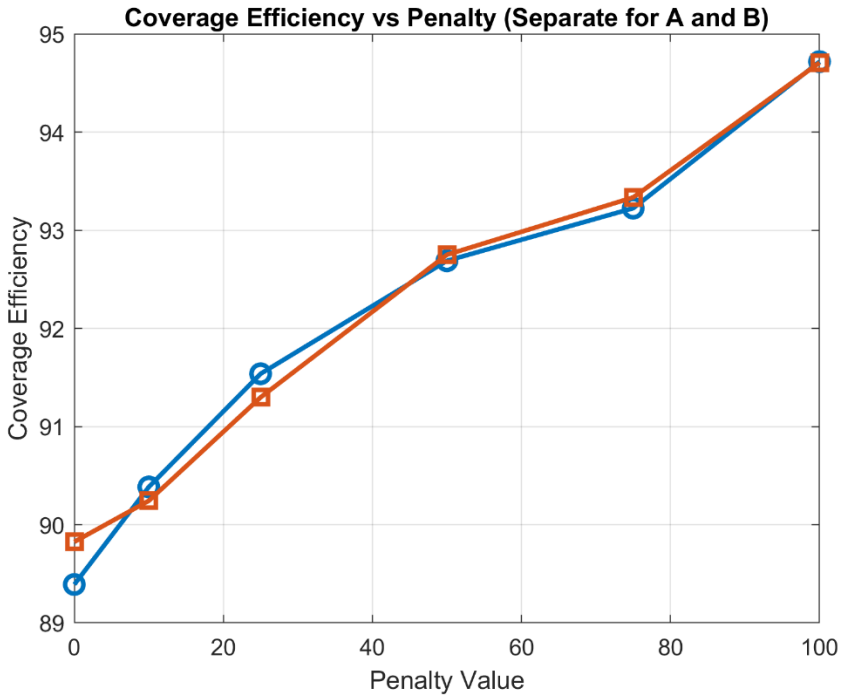

B Coverage vs Testcase Number
B Coverage Percentage
Testcase Number
0
10
20
30
40
50
60
70
80
90
100
0
2000
4000
6000
8000
10000
12000

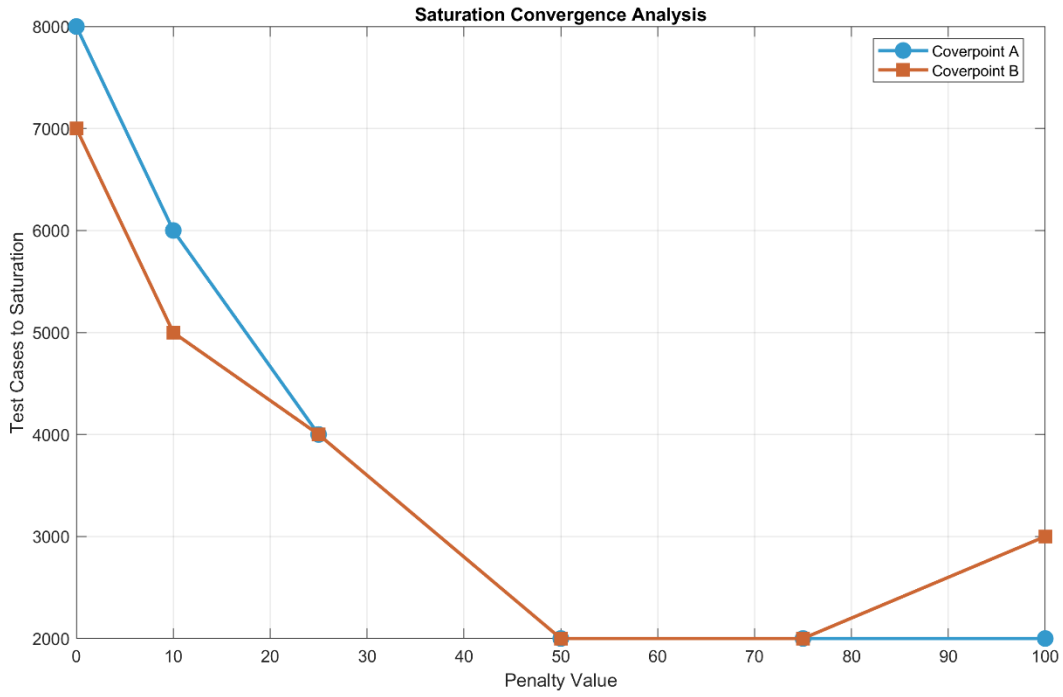

Cin Coverage vs Testcase Number
Cin Coverage Percentage
Testcase Number
99
99.2
99.4
99.6
99.8
100
100.2
100.4
100.6
100.8
101
0
2000
4000
6000
8000
10000
12000



Relationship of Coverage with penalty plots:




Coverage Efficiency vs Penalty (Separate for A and B)
Coverage Efficiency
Penalty Value
95
94
93
92
91
90
89
0
20
40
60
80
100


Saturation Convergence Analysis
Test Cases to Saturation
Penalty Value
Coverpoint A
Coverpoint B
8000
7000
6000
5000
4000
3000
2000
0
10
20
30
40
50
60
70
80
90
100

6 Synthesis and Optimization

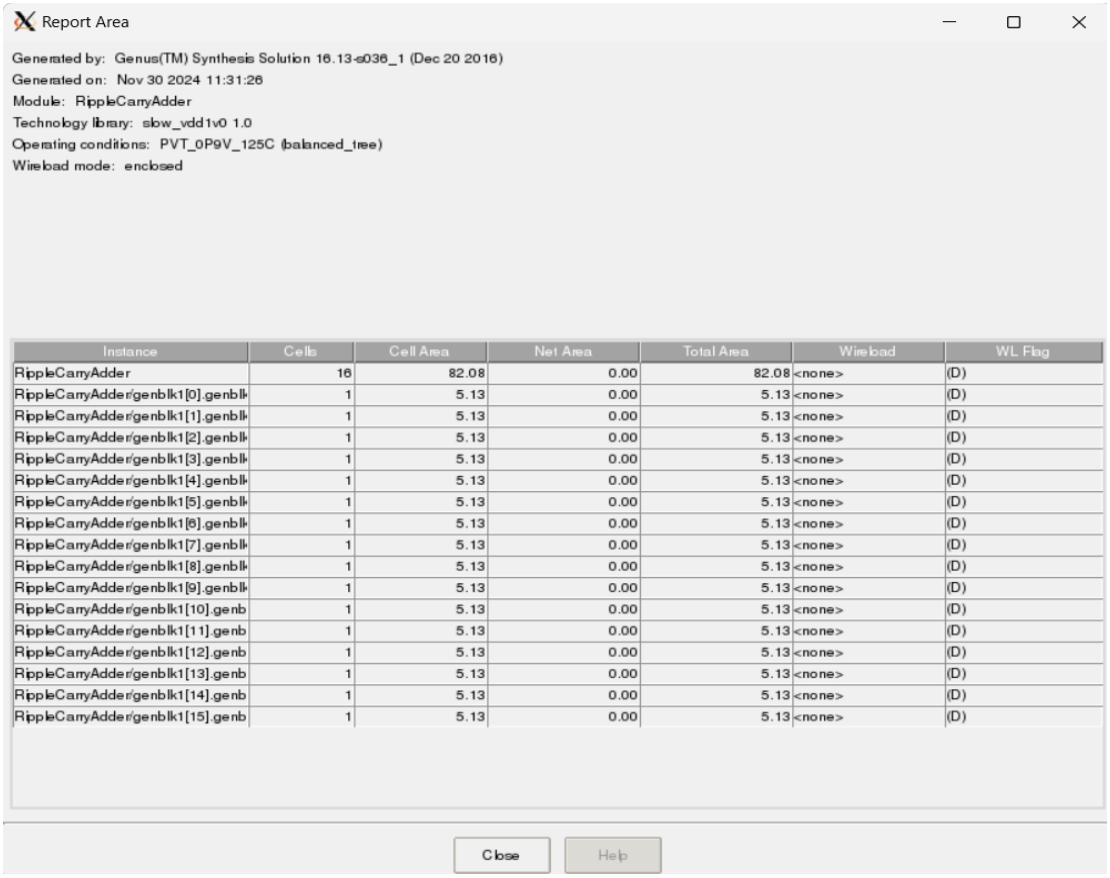
We need to tune these parameters:

1. Input Delay
2. Output Delay
3. Frequency

Then our optimization targets are:

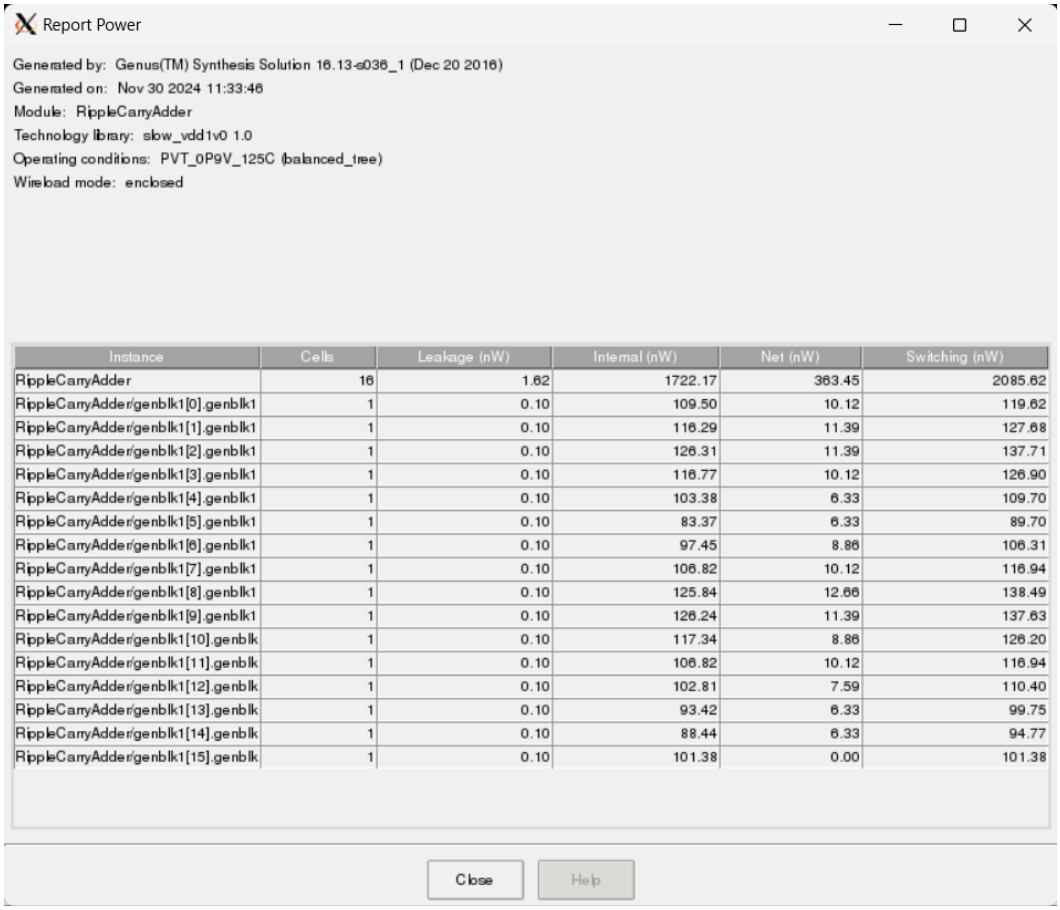
1. Power
2. Area
3. Delay
4.

Synthesis of adder module without clock in Genus:



Instance	Cells	Cell Area	Net Area	Total Area	Wireload	WL Flag
RippleCarryAdder	16	82.08	0.00	82.08	<none>	(D)
RippleCarryAdder/genblk1[0].genblk	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[1].genblk	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[2].genblk	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[3].genblk	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[4].genblk	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[5].genblk	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[6].genblk	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[7].genblk	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[8].genblk	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[9].genblk	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[10].genb	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[11].genb	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[12].genb	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[13].genb	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[14].genb	1	5.13	0.00	5.13	<none>	(D)
RippleCarryAdder/genblk1[15].genb	1	5.13	0.00	5.13	<none>	(D)

Fig: Area Report

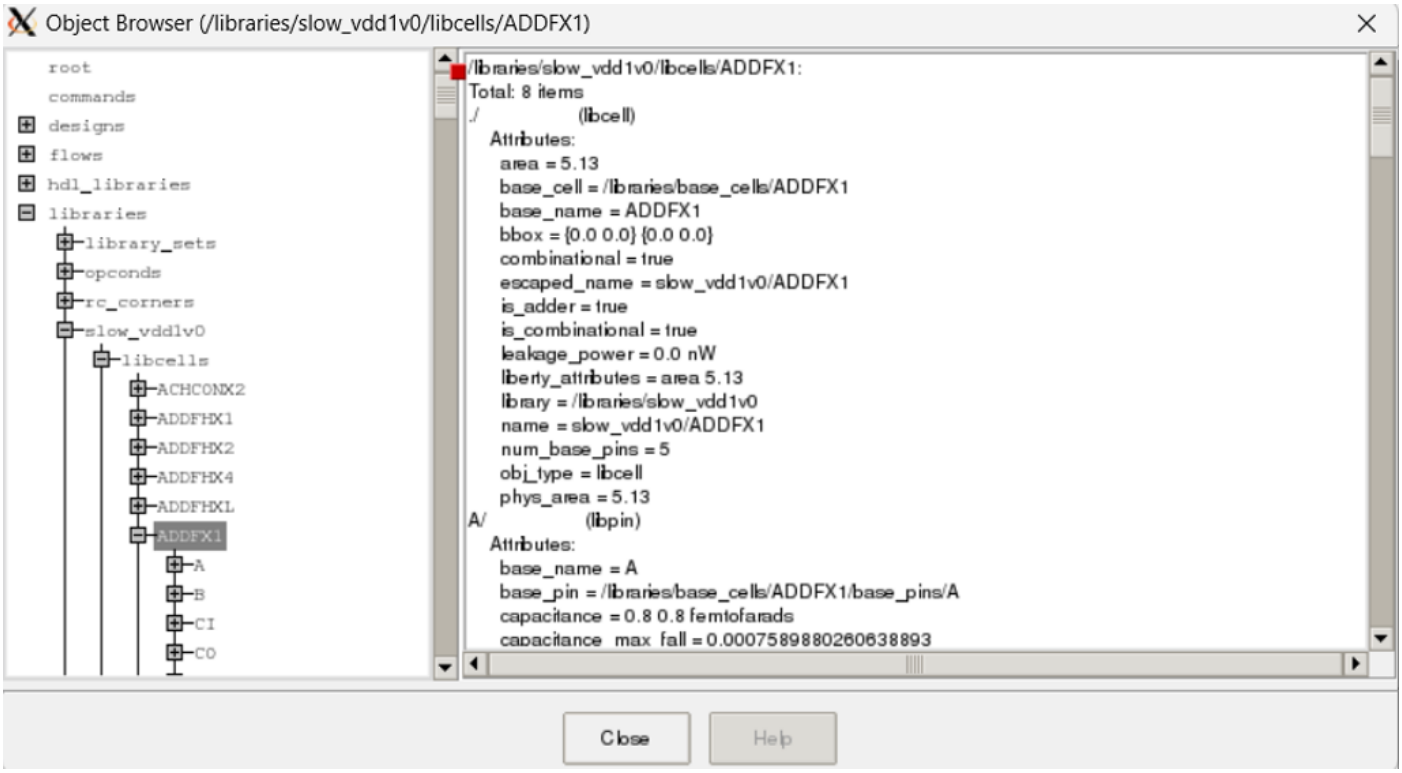


Instance	Cells	Leakage (nW)	Internal (nW)	Net (nW)	Switching (nW)
RippleCarryAdder	16	1.62	1722.17	363.45	2085.62
RippleCarryAdder/genblk1[0].genblk1	1	0.10	109.50	10.12	119.62
RippleCarryAdder/genblk1[1].genblk1	1	0.10	116.29	11.39	127.68
RippleCarryAdder/genblk1[2].genblk1	1	0.10	126.31	11.39	137.71
RippleCarryAdder/genblk1[3].genblk1	1	0.10	116.77	10.12	126.90
RippleCarryAdder/genblk1[4].genblk1	1	0.10	103.38	6.33	109.70
RippleCarryAdder/genblk1[5].genblk1	1	0.10	83.37	6.33	89.70
RippleCarryAdder/genblk1[6].genblk1	1	0.10	97.45	8.86	106.31
RippleCarryAdder/genblk1[7].genblk1	1	0.10	106.82	10.12	116.94
RippleCarryAdder/genblk1[8].genblk1	1	0.10	125.84	12.66	138.49
RippleCarryAdder/genblk1[9].genblk1	1	0.10	126.24	11.39	137.63
RippleCarryAdder/genblk1[10].genblk	1	0.10	117.34	8.86	126.20
RippleCarryAdder/genblk1[11].genblk	1	0.10	106.82	10.12	116.94
RippleCarryAdder/genblk1[12].genblk	1	0.10	102.81	7.59	110.40
RippleCarryAdder/genblk1[13].genblk	1	0.10	93.42	6.33	99.75
RippleCarryAdder/genblk1[14].genblk	1	0.10	88.44	6.33	94.77
RippleCarryAdder/genblk1[15].genblk	1	0.10	101.38	0.00	101.38

Fig: Power report

This is the standard cell in synthesized circuit:

Details of the standard cell:

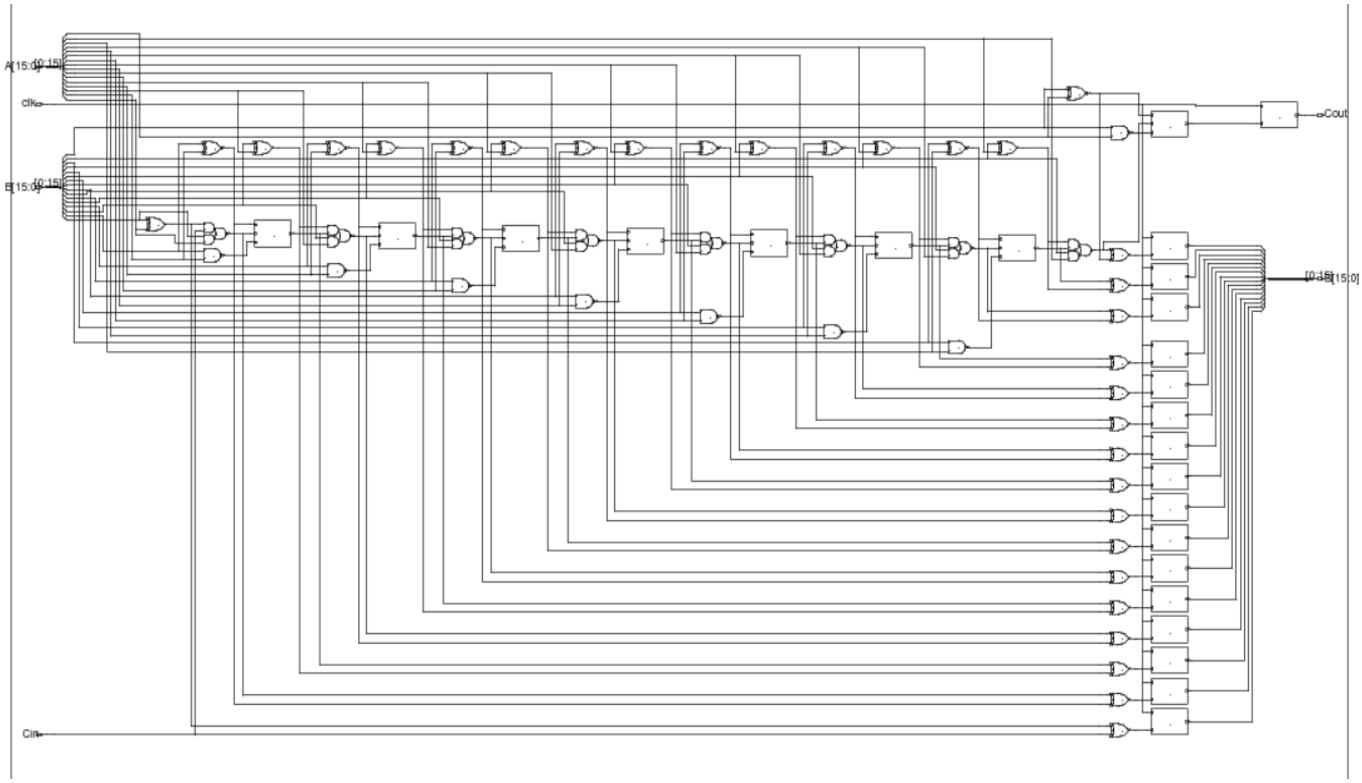


Observation:

No relation with clock frequency and input output delay

Optimization needs change in fundamental architecture of the adder

Synthesis of adder module with clock in Genus:



6.1 Optimization

Tuning Parameter range:

clk 1 5 10 50 100 500 1000 ns

Frequencies 1000,200,100,20,10,2,1 MHz

input_delays 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

output delays 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

Formula used for weighted sum:

```
% Calculate the weighted sum (60% Time, 20% Power, 20% Area)
data.Weighted_Sum = 0.6 * data.Normalized_Time + 0.2 * data.Normalized_Power + 0.2 * data.Normalized_Area;
```

Optimized point for minimum weighted sum:

```
Minimum Weighted Sum: 0.227426
Corresponding Frequency (MHz): 100.000000
Corresponding Input Delay (ns): 0.100000
Corresponding Output Delay (ns): 0.100000
```

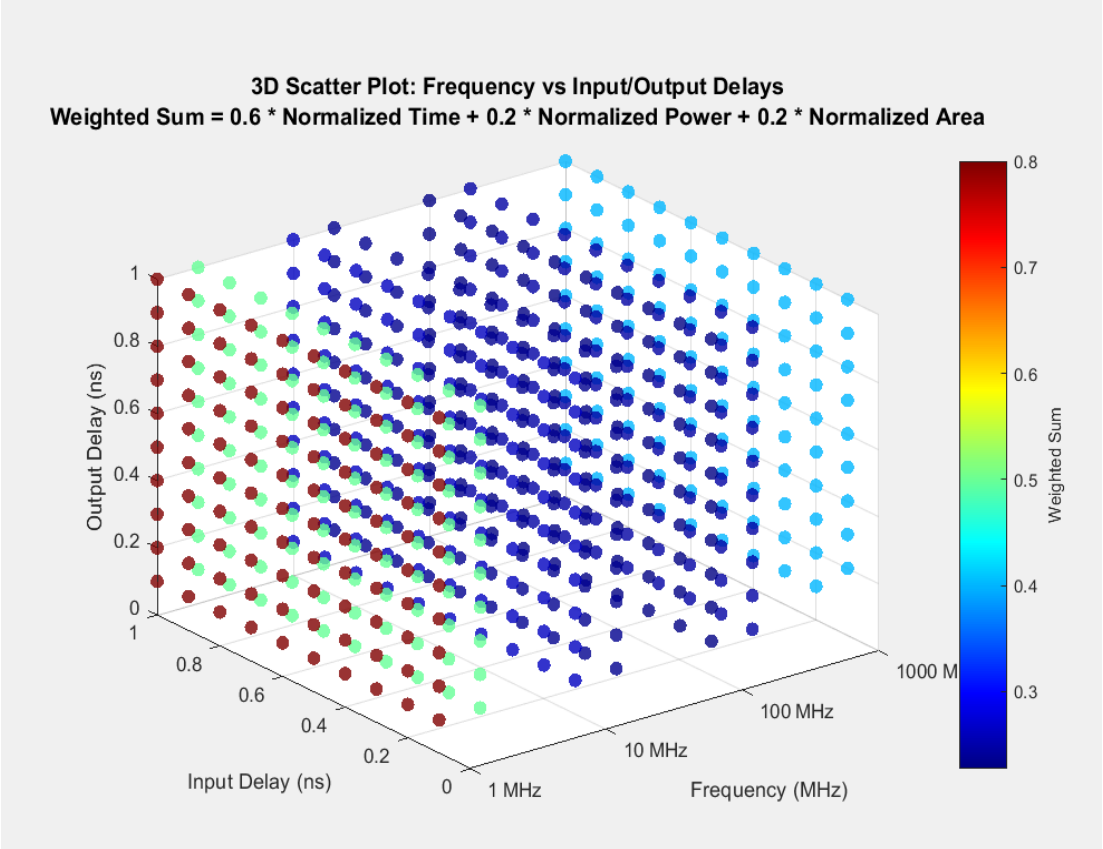
TCL file to generate SDC files according to parameters:

```
::legacy::set_attribute common_ui false / ;
if {[file exists /proc/cpuinfo]} {
sh grep "model name" /proc/cpuinfo
sh grep "cpu MHz" /proc/cpuinfo
}
puts "Hostname : [info hostname]"


set clock_periods {1 5 10 50 100 500 1000}
set input_delays {0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1}
set output_delays {0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1}

set DESIGN RippleCarryAdder
set SYN_EFF high
set MAP_EFF high
set OPT_EFF high

set pdk_dir /home/cad/VLSI2Lab/Digital/library/
set_attribute init_lib_search_path $pdk_dir


set_attribute syn_generic_effort $SYN_EFF
set_attribute syn_map_effort $MAP_EFF
set_attribute syn_opt_effort $OPT_EFF
set_attribute library "slow_vdd1v0_basicCells.lib"
set_dont_use [get_lib_cells CLK*]
set_dont_use [get_lib_cells SDFF*]
set_dont_use [get_lib_cells DLY*]
set_dont_use [get_lib_cells HOLD*]


read_hdl -sv "\
${DESIGN}.sv"
elaborate $DESIGN
puts "Runtime & Memory after 'read_hdl"
time_info Elaboration
check_design -unresolved



foreach clock_period $clock_periods {
foreach input_delay $input_delays {
foreach output_delay $output_delays {




# Create a new SDC file for each setup time
    set sdc_file "${DESIGN}_clk_${clock_period}_ipd_${input_delay}_opd_$
{output_delay}.sdc"

    # Write the new SDC file
    set output [open $sdc_file "w"]

    puts $output "create_clock -period ${clock_period}  -waveform {0 6} -name
func_clk [get_ports clk]"
    puts $output "set_input_delay ${input_delay} -clock [get_clocks func_clk] {A,
B, Cin}"AS
```

```
    puts $output "set_output_delay ${output_delay} -clock [get_clocks func_clk] {S,
Cout}"

    close $output

    # Read the newly created SDC file
    read_sdc $sdc_file




syn_generic
puts "Runtime & Memory after 'syn_generic'"
time_info GENERIC

    report_power -verbose -detail >> reports/power_clk_${clock_period}_ipd_$
{input_delay}_opd_${output_delay}.txt
    report_area  >> reports/area_clk_${clock_period}_ipd_${input_delay}_opd_$
{output_delay}.txt
    report timing -encounter >> reports/time_clk_${clock_period}_ipd_$
{input_delay}_opd_${output_delay}.txt

}}}

##This synthesizes your code
synthesize


## THESE FILES ARE NOT REQUIRED, THE SDC FILE IS A TIMING FILE
write_script > script
```

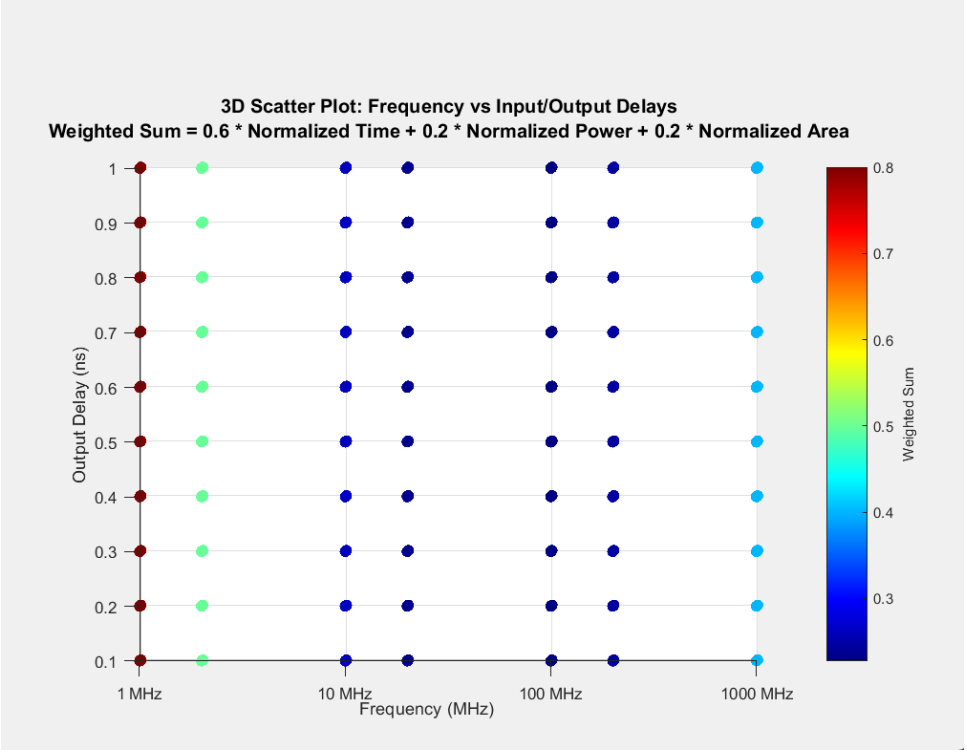
Here the TCL file was coded to write SDC file for each iteration of the input variables and then read the SDC file to generate reports. We ran nested loops to be able to generate reports for variation of each variable. We generated about 2100 reports and then plotted the results using MATLAB for visualization and to understand the trends.

Variation of Weighted Sum with Tuning Parameters:

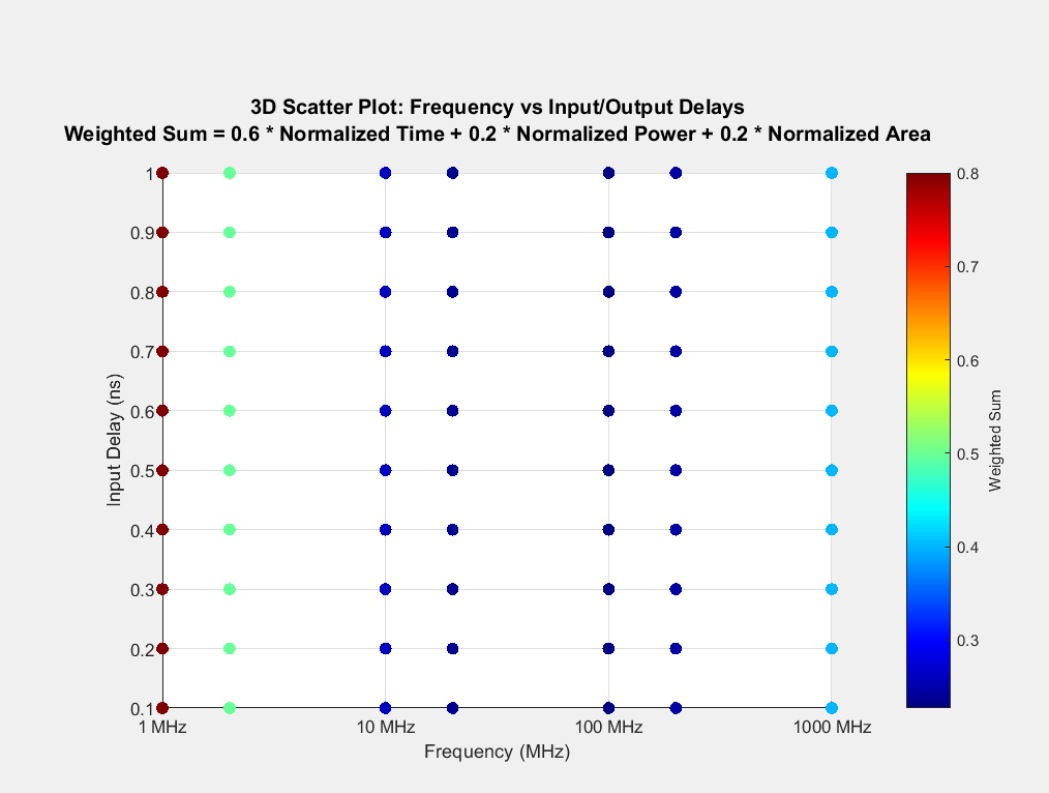


Here the weighted sum is shown as a color gradient where the lower values have a bluer color and higher ones have red color.

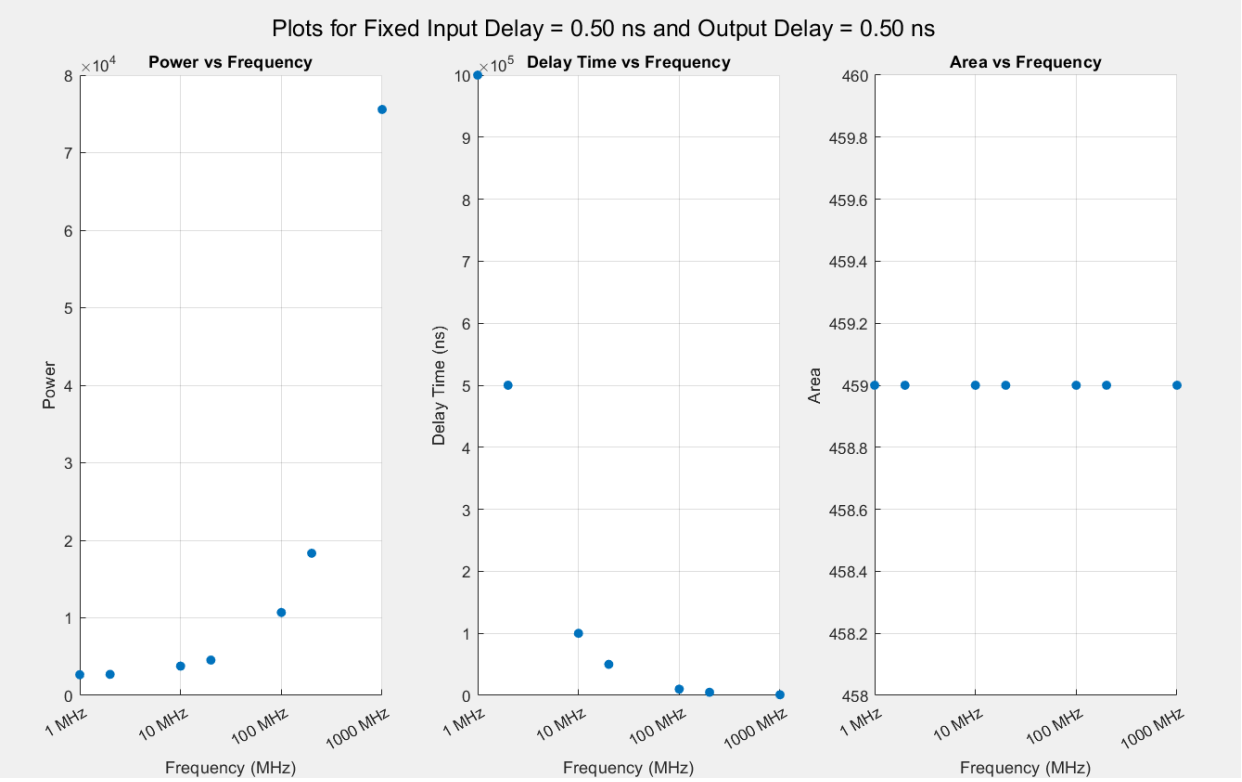
Below is only the output delay vs Frequency plot



Below is only the input delay vs Frequency plot

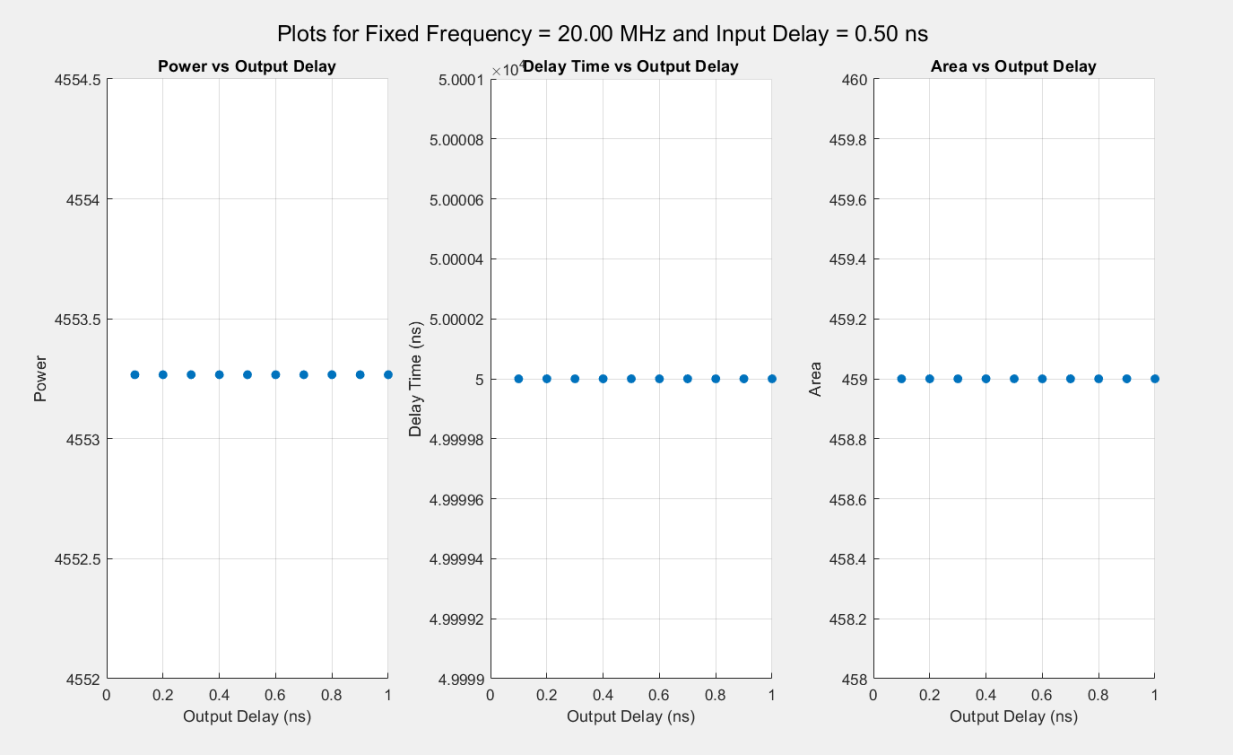


Plots for Fixed Delay:



It can be seen that as frequency increases power increases because there are more switching losses, Delay time decreases as clock cycle gets lower and the results can come faster. Area don't change with frequency

Plots for Fixed frequency:



Here the values don't change with respect to the delay parameter.

Reports:

```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 16.13-s036_1
  Generated on:           Dec 15 2024  03:30:53 pm
  Module:                 RippleCarryAdder
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

Path 1: MET Setup Check with Pin Cout_reg/clk
Endpoint:   Cout_reg/d (^) checked with leading edge of 'func_clk'
Beginpoint: B[0]       (^) triggered by leading edge of 'func_clk'
Other End Arrival Time         100000
- Setup                           119
= Required Time                 99881
- Arrival Time                   3053
= Slack Time                    96828
    Clock Rise Edge                 0
    + Input Delay                 200
    = Beginpoint Arrival Time     200
```

Figure: Timing Report

Here the Other end Arrival time is 100000. This is the Critical path delay time for design with clock.

```
====================================================================
  Generated by:           Genus(TM) Synthesis Solution 16.13-s036_1
  Generated on:           Dec 15 2024  03:34:51 pm
  Module:                 RippleCarryAdder
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
====================================================================

                       Leakage   Internal    Net        Dynamic          Total
                                                     (Internal+Net) (Leakage+Dynamic)
    Instance     Cells Power(nW) Power(nW) Power(nW)    Power(nW)       Power(nW)
--------------------------------------------------------------------------------------
RippleCarryAdder   193     4.936  1472.185  1235.660       2707.845          2712.781
```

Figure: Power Report

We can see that the total leakage plus dynamic power is 2712.78 nW

```
====================================================================
  Generated by:           Genus(TM) Synthesis Solution 16.13-s036_1
  Generated on:           Dec 15 2024  03:21:39 pm
  Module:                 RippleCarryAdder
  Technology library:     slow_vdd1v0 1.0
  Operating conditions:   PVT_0P9V_125C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
====================================================================

    Instance     Module  Cells  Cell Area  Net Area   Total Area  Wireload
-------------------------------------------------------------------------------
RippleCarryAdder           193        459         0          459    <none> (D)

 (D) = wireload is default in technology library
```

Figure: Area Report

Observations:

1. The delay time through the critical path depends only on clock frequency
2. The power requirement increases with high frequency
3. Input and output delay has no effect on power, area or delay time
4. Making the addition synchronized with clock requires more power and area and introduces even more delay in the critical path

7 Physical Design (PnR)

Die Size: 20 µm × 20 µm

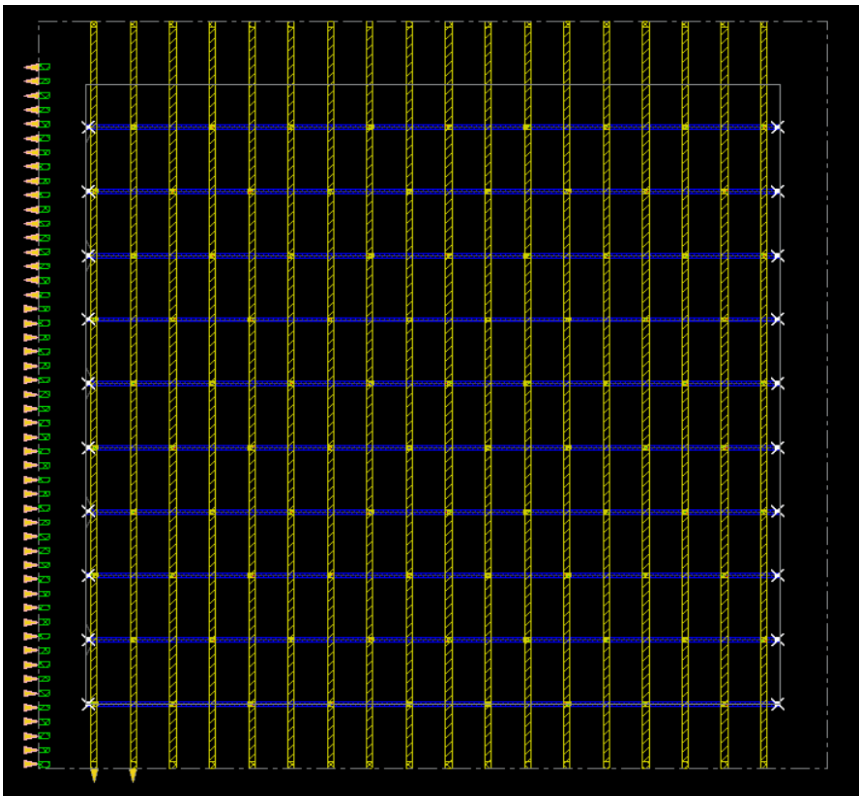


Fig: initial die declaration

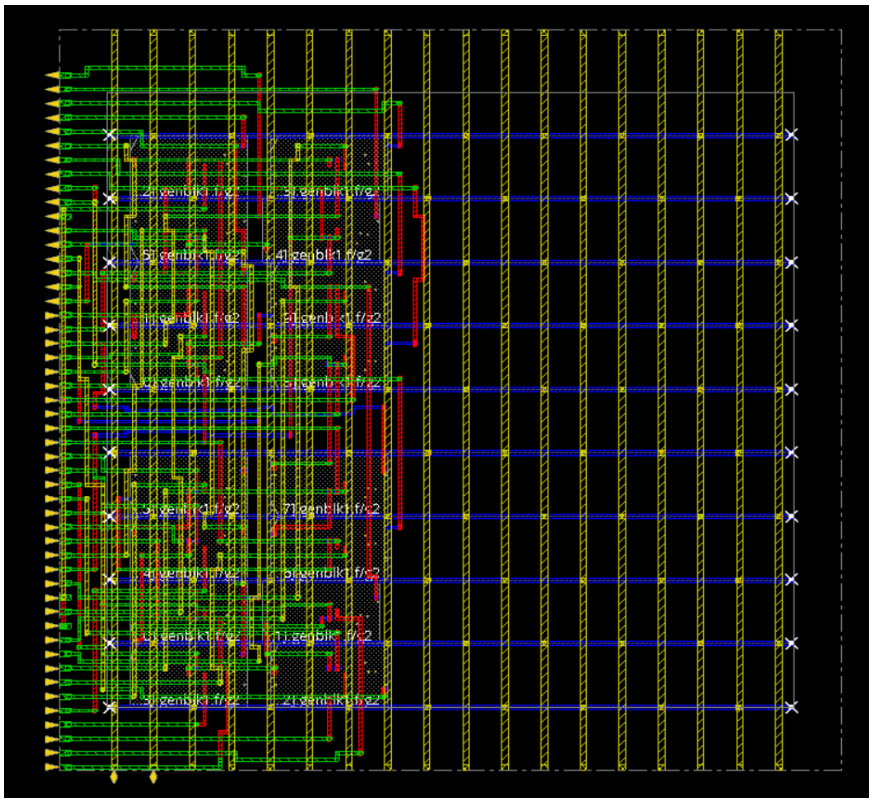


Fig: after cell placement with nano routing

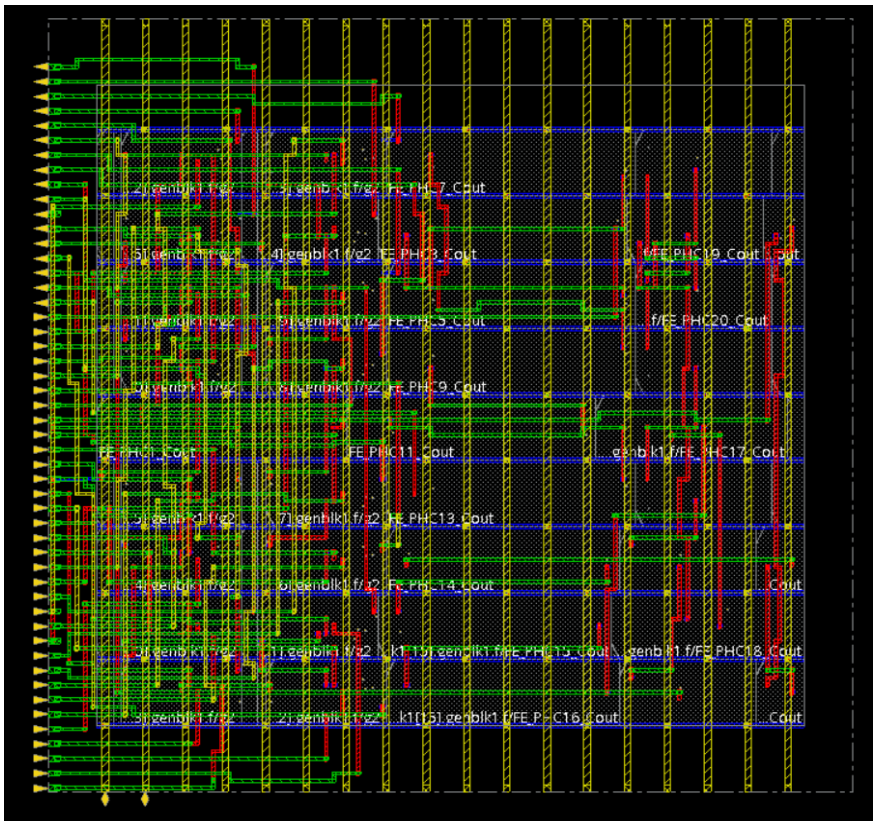


Fig: after filler cell

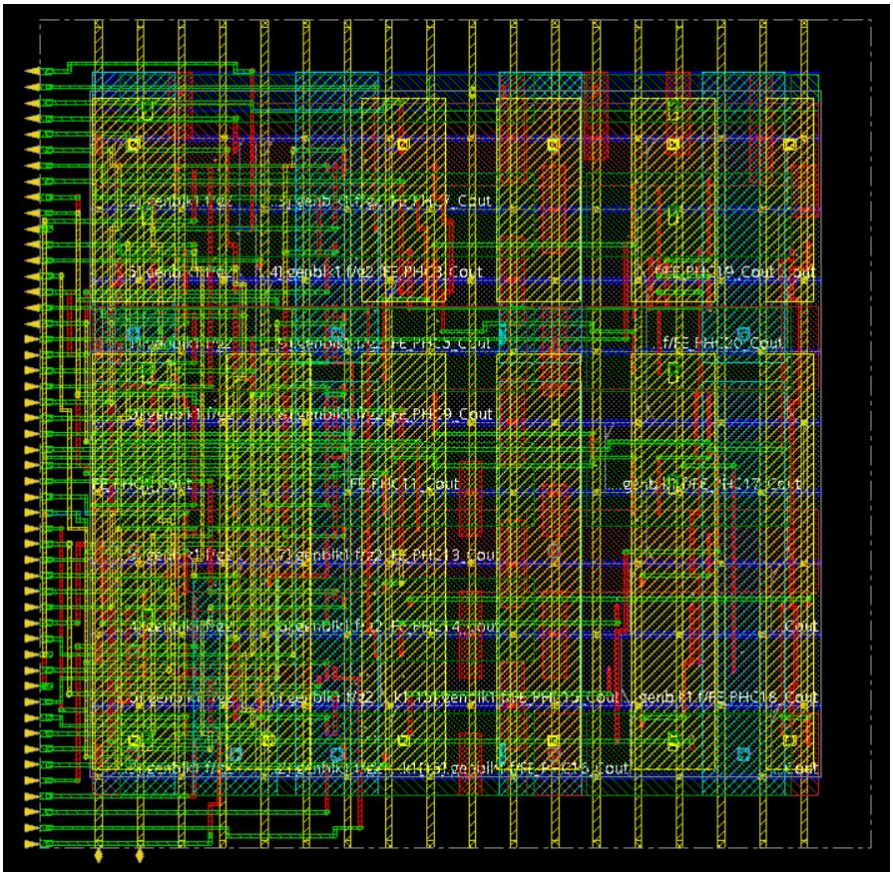


Fig: Layout with metal fill

```
innovus 27> innovus 27>  *** Starting Verify Geometry (MEM: 1367.6) ***

**WARN: (IMPVFG-257):   verifyGeometry command is replaced by verify_drc command
. It still works in this release but will be removed in future release. Please u
pdate your script to use the new command.
  VERIFY GEOMETRY ..... Starting Verification
  VERIFY GEOMETRY ..... Initializing
  VERIFY GEOMETRY ..... Deleting Existing Violations
  VERIFY GEOMETRY ..... Creating Sub-Areas
                  ..... bin size: 1920
  VERIFY GEOMETRY ..... SubArea : 1 of 1
  VERIFY GEOMETRY ..... Cells          :  0 Viols.
  VERIFY GEOMETRY ..... SameNet        :  0 Viols.
  VERIFY GEOMETRY ..... Wiring         :  0 Viols.
  VERIFY GEOMETRY ..... Antenna        :  0 Viols.
  VERIFY GEOMETRY ..... Sub-Area : 1 complete 0 Viols. 0 Wrngs.
VG: elapsed time: 1.00
Begin Summary ...
  Cells       : 0
  SameNet     : 0
  Wiring      : 0
  Antenna     : 0
  Short       : 0
  Overlap     : 0
End Summary

  Verification Complete : 0 Viols.  0 Wrngs.

**********End: VERIFY GEOMETRY**********
 *** verify geometry (CPU: 0:00:00.1  MEM: 168.1M)
```

Fig: Verification report

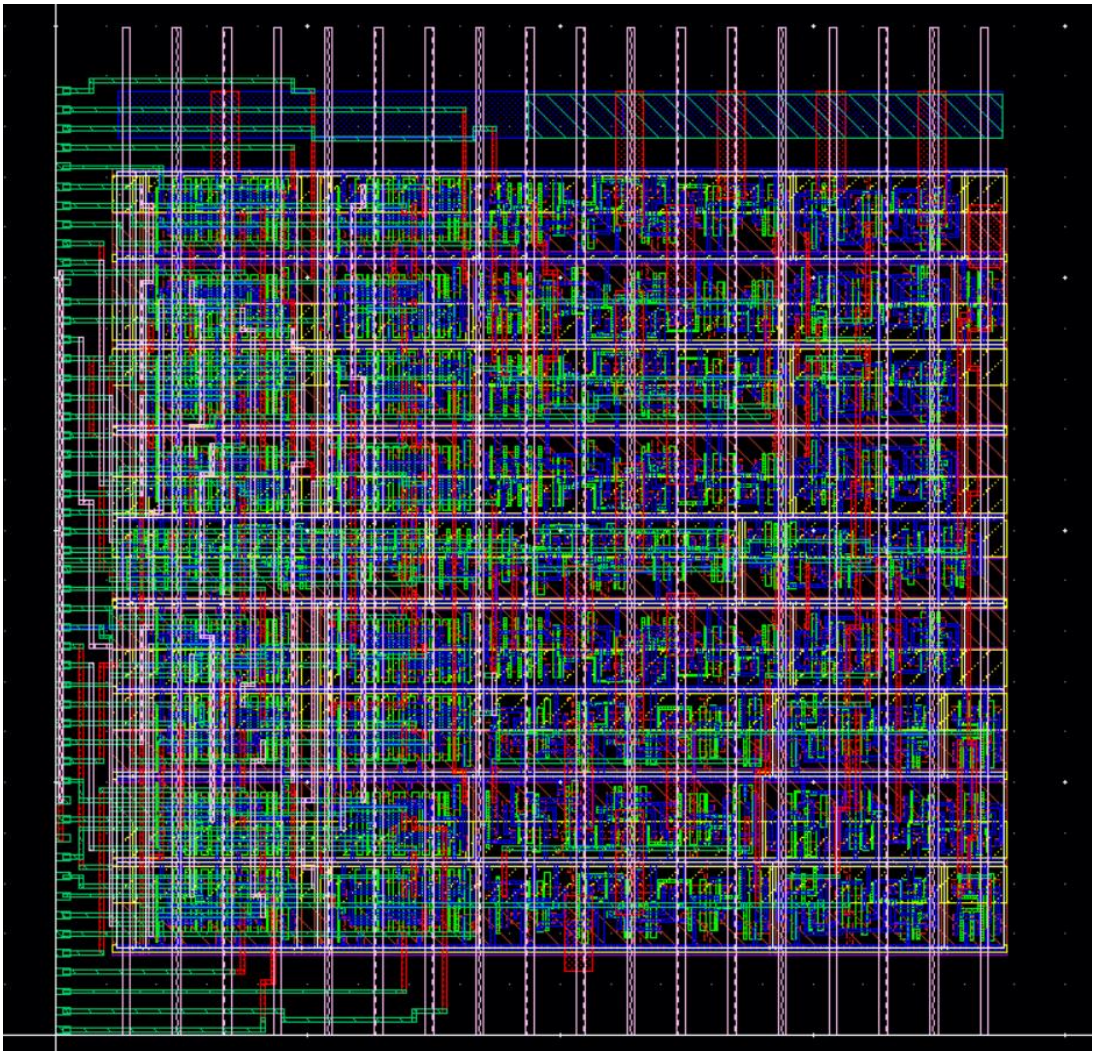


Fig: Final layout

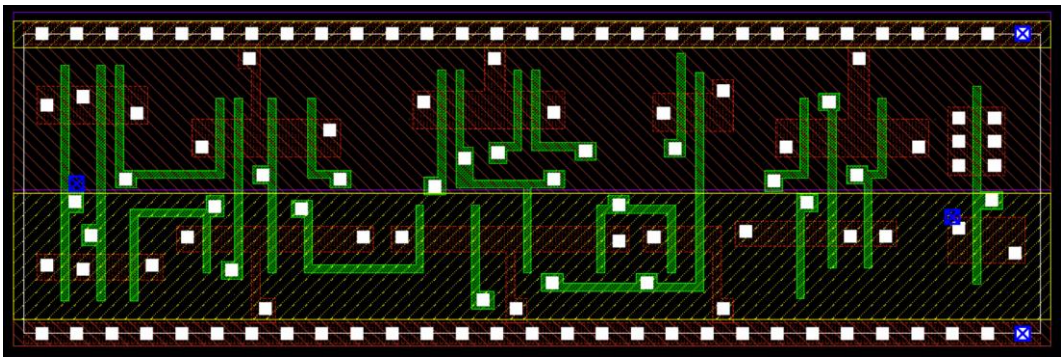


Fig: Zoomed in final layout

DRC Report:

```
Total CPU Time                  : 1(s)
Total Real Time                 : 1(s)
Peak Memory Used                : 22(M)
Total Original Geometry         : 1615(8352)
Total DRC RuleChecks            : 562
Total DRC Results               : 0 (0)
Summary can be found in file RippleCarryAdder.sum
ASCII report database is /home/vlsi25/cds_digital_labtest/synthesis/RippleCarryAdder.drc_errors.ascii
Checking in all SoftShare licenses.

Design Rule Check Finished Normally. Sat Nov 30 15:08:15 2024
```

Cell Name
⊞ RippleCarryAdder (E0)

Now we will try to reduce the area for area optimization. We will also distribute the pins on both sides.

Die Size: 12 µm × 15 µm

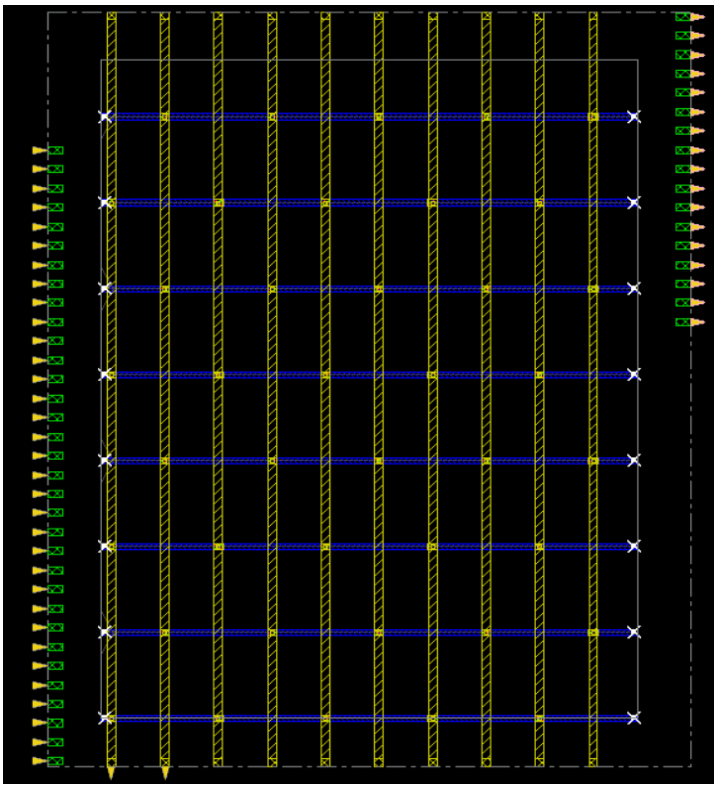


Fig: initial die declaration

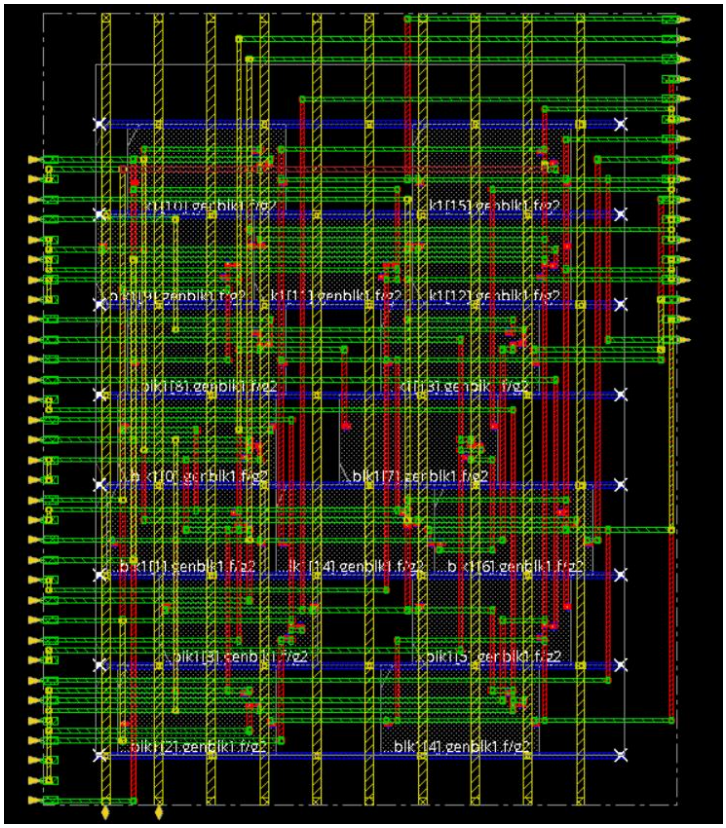


Fig: after cell placement with nano routing

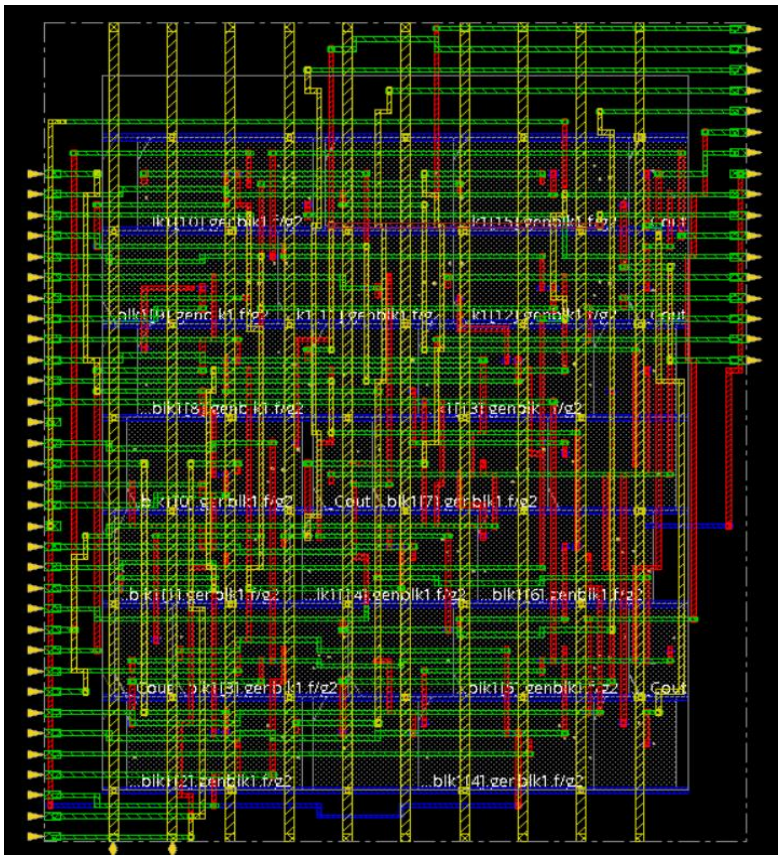


Fig: after Filler Cell

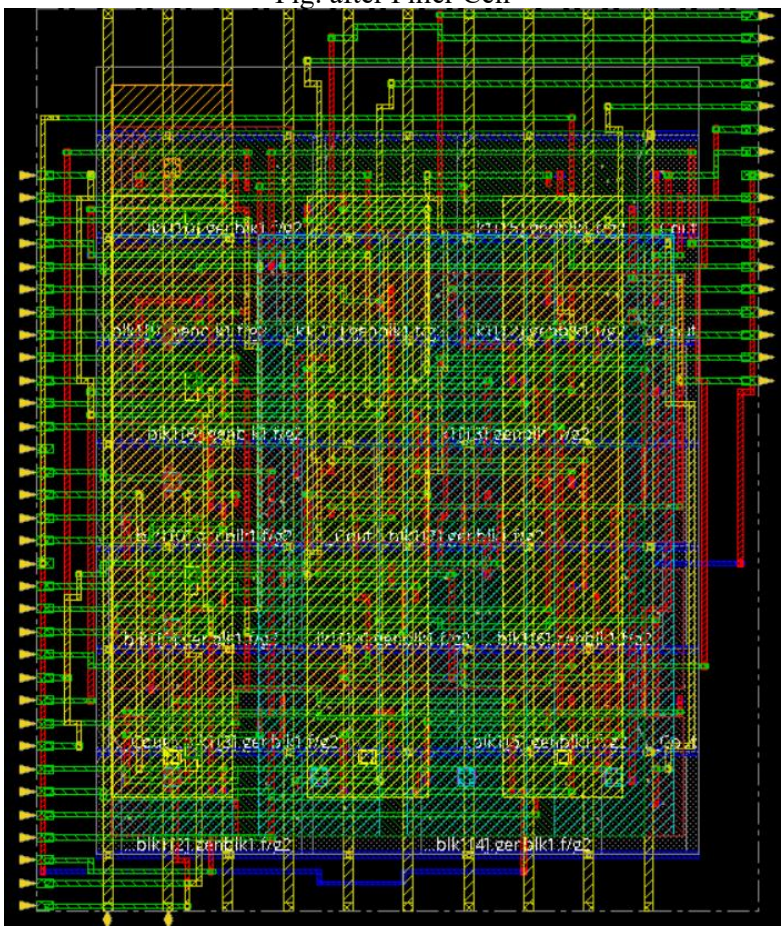


Fig: metal fill

```
------------------------------------------------------------
     optDesign Final SI Timing Summary
------------------------------------------------------------

Setup views included:
 func@BC_rcbest0.hold
Hold  views included:
 func@BC_rcbest0.hold

+--------------------+---------+---------+
|     Setup mode     |   all   | default |
+--------------------+---------+---------+
|           WNS (ns):| 27.889  | 27.889  |
|           TNS (ns):|  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |
|          All Paths:|    1    |    1    |
+--------------------+---------+---------+

+--------------------+---------+---------+
|     Hold mode      |   all   | default |
+--------------------+---------+---------+
|           WNS (ns):| -18.986 | -18.986 |
|           TNS (ns):| -18.986 | -18.986 |
|    Violating Paths:|    1    |    1    |
|          All Paths:|    1    |    1    |
+--------------------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------|
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |      0 (0)       |   0.000    |      0 (0)       |
|   max_tran     |      0 (0)       |   0.000    |      0 (0)       |
|   max_fanout   |      0 (0)       |     0      |      0 (0)       |
|   max_length   |      0 (0)       |     0      |      0 (0)       |
+----------------+------------------+------------+------------------+

Density: 93.714%
Total number of glitch violations: 0
------------------------------------------------------------
```

Fig: optimization summary

We can see that the Density is 93.714% This is our most area optimized design.

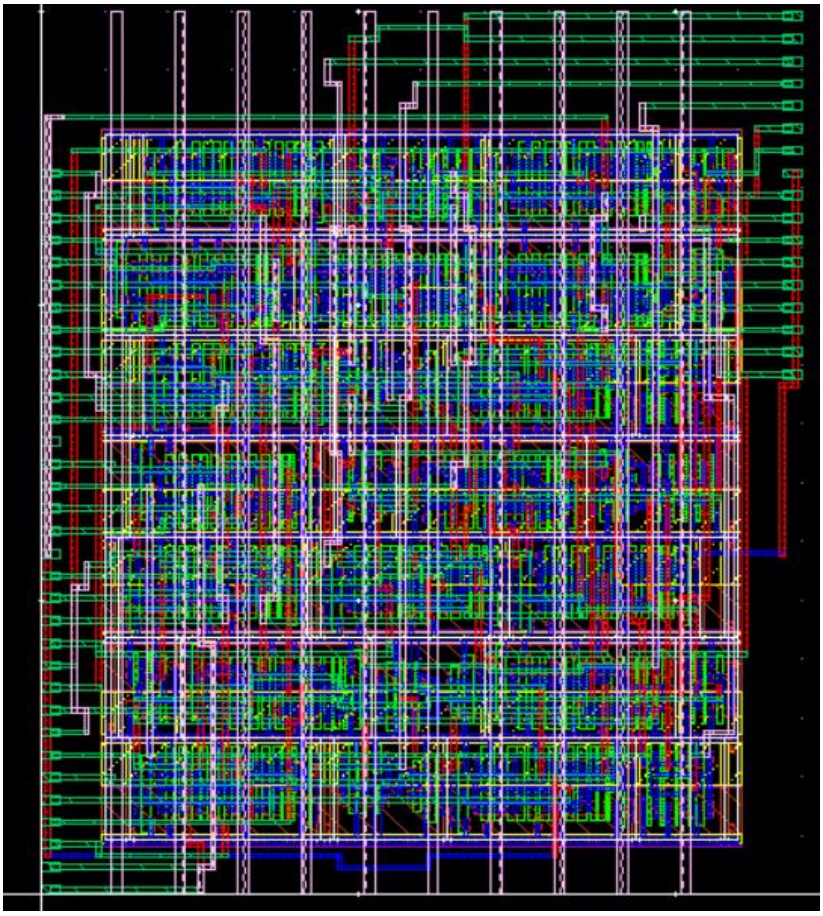


Fig: Final Layout of the die size 12 µm × 15 µm

DRC report shows no error

```
Total CPU Time                    : 1(s)
Total Real Time                   : 1(s)
Peak Memory Used                  : 20(M)
Total Original Geometry           : 998(4784)
Total DRC RuleChecks              : 562
Total DRC Results                 : 0 (0)
Summary can be found in file RippleCarryAdder.sum
ASCII report database is /home/vlsi25/cds_digital_labtest/synthesis/optimize_innovus/RippleCarryAdder.drc
Checking in all SoftShare licenses.


Design Rule Check Finished Normally. Sun Dec 15 19:33:50 2024
```

Now if we try to further reduce the size further, that results in errors.

Size: 10 µm × 10 µm

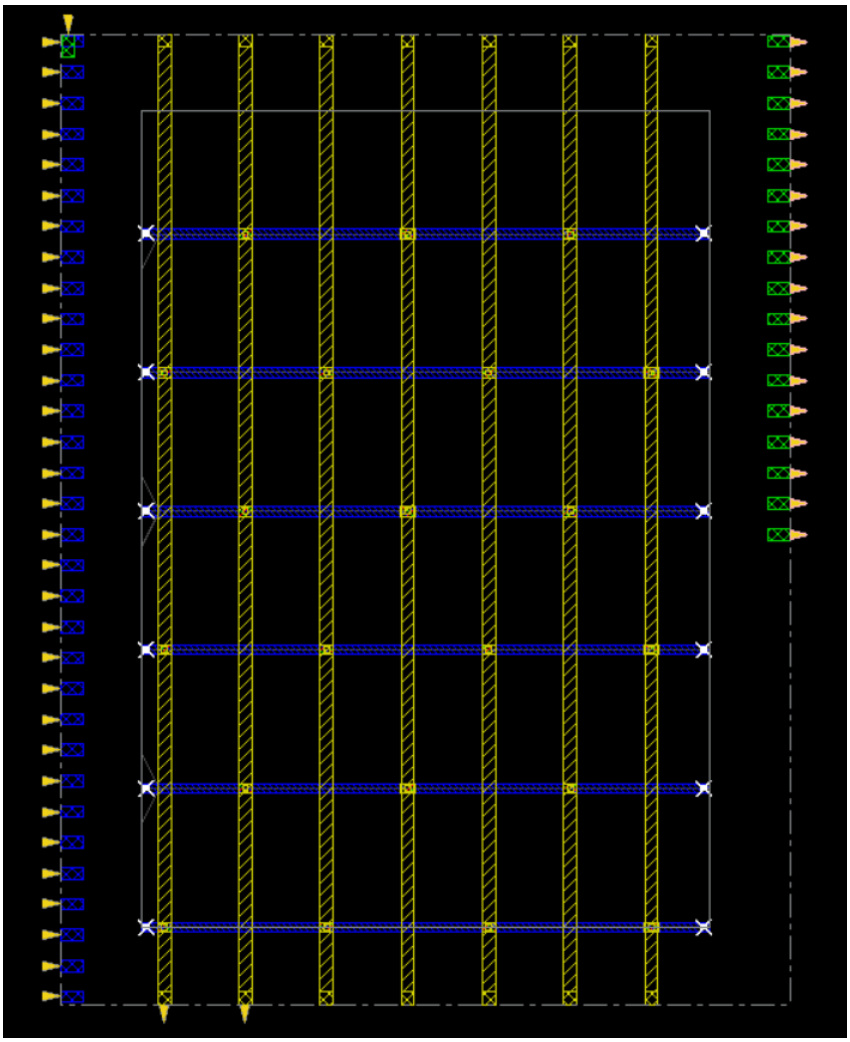


Fig: initiating die for 10 µm × 10 µm

Shows error during cell placement:

```
**ERROR: (IMPSP-190):   design has util 137.1% > 100%. Placer cannot proceed. Pl
ease correct this problem first.
**placeDesign ... cpu = 0: 0: 0, real = 0: 0: 0, mem = 813.4M **
```

Ultimately Our is design is best optimized at die size 12 µm × 15 µm. as reducing from there gives density greater than 100%

Here's the summary:

Die size (um)	Density Utilization
20 by 20	66.50 %
12 by 15	93.714%
10 by 10	137%

Hence we conclude our design with a die size of 12 by 15 for maximum utilization with error free simulation.

8 Future Works

We can use advanced layered testbench to reduce testing times even further. The standard cell of adder can be modified manually to give better delay and other results. The core architecture of the adders need to be improved to tackle the issue like introducing carry look ahead or carry skip adder architecture, PnR can be micro-improved although it may be time consuming.

9 References

i. CMOS VLSI Design
A Circuits and Systems Perspective
Neil H. E. Weste
David Money Harris

ii. A practical look at system Verilog
Doug Smith
Dough.smith@doulos.com

iii. System Verilog for Verification
3rd editiom
Chris Spear and Greg Tumbush