SYNOPSYS RTL-TO-GDSII WORKSHOP REPORT

Organized by

VLSI EXPERT

Submitted by:
Name: Ira Tyagi
Branch: Electronics Engineering (VLSI & Design Technology)
Department: Department of Electronics Engineering
College: Vivekananda Institute of Professional studies - Technical Campus (VIPS-TC)
Semester: 3^{rd} Semester
Date of Submission: 24 October 2025

Submitted to:

Workshop Mentor / Trainer: Bala Krishnan and Puneet Mittal
Organization: VLSI Expert

Workshop Objective:

To understand the **complete RTL to GDSII design flow** using **Synopsys EDA tools**, starting from RTL design, simulation (VCS, Verdi), synthesis (Design Compiler), and physical design (IC Compiler II), leading to signoff (PrimeTime).

योगः कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

"A Technical Workshop Report under Synopsys EDA Tools"

Acknowledgment

I would like to express my sincere gratitude to VLSI Expert for organizing this workshop on RTL-to-GDSII flow. I thank my mentor Bala Krishna and Puneet Mittal sir for their guidance and support throughout the sessions. I am also thankful to my Department of Electronics Engineering and college VIPS-TC for providing resources and encouragement to complete this report.

Table of Contents

Introduction

The **RTL-to-GDSII flow** is a step-by-step digital design process that converts a hardware design written in **Verilog RTL** into a physical layout (GDSII) ready for fabrication.

In this workshop, we focus on designing and verifying a **Full Adder** at the RTL level, which is a fundamental building block in digital circuits used for **binary addition**. The **Full Adder** takes three inputs (A, B, Carry-in) and produces two outputs (Sum and Carry-out).

Simulation and verification using **Synopsys VCS** and waveform analysis with **Verdi** ensure that the Full Adder design works correctly before moving to synthesis and physical design. This workshop helps students understand **functional verification, simulation flow, and debugging techniques** for small but essential digital modules like a Full Adder.

Objective

- Understand and design a **Full Adder** at RTL level.
- Simulate and verify functionality using **VCS**.
- Debug and analyse outputs and waveforms with **Verdi**.
- Prepare the verified design for **synthesis and physical implementation**.

By the end of the workshop, students were expected to understand not only the individual design steps but also the interdependencies between them-forming a solid foundation for further exploration into physical design, timing optimization, and VLSI layout methodologies.

Tools Used

Tool	Purpose
VCS	Verilog simulation & compilation
Verdi	Waveform visualization & debugging
Design Compiler	RTL synthesis
IC Compiler II	Placement & routing
PrimeTime	Static timing analysis

RTL Design & Simulation

1. Design Description

The design consists of two files:

- full_adder.v → RTL design file describing the logic of a full adder.
- full_adder_tb.v → Testbench file that applies input combinations and checks outputs.

2. Creating Verilog Files

```
[ws_263@wsv3 ~]$ cd RTL2GDSII_TESTCASE
[ws_263@wsv3 RTL2GDSII_TESTCASE]$ cd rtl
[ws_263@wsv3 rtl]$ ls
csrc             novas.conf      novas.rc     ucli.key
full_adder_tb.v  novas_dump.log  simv         verdi_config_file
full_adder.v     novas.fsdb      simv.daidir  verdiLog
[ws_263@wsv3 rtl]$
```

Full Adder:

```
module full_adder (A, B, C_in, C_out, Clock, SUM);
input [3:0] A, B;
input Clock, C_in;
output reg [3:0] SUM;
output reg C_out;

//Internal registers
reg [3:0] reg1, reg2, sum_i;
reg c_in, c_out;

//Inputs to Internal Registers (synchronized to clock)
always @ (posedge Clock)
    begin
        reg1 <= A;
        reg2 <= B;
            c_in <= C_in;
    end

//Output Registering Synchronized to clock
always @ (posedge Clock)
    begin
        SUM <= sum_i;
        C_out <= c_out;
    end

//combinational 4-bit full addition logic
always @ *
    begin
        {c_out, sum_i} = reg1 + reg2 + c_in;
    end
endmodule
```

Full Adder Testbench:

```
`timescale 1ns/1ns

`include "full_adder.v" // includes the module definition for the full adder

module testbench;
        reg [3:0] A, B;
        reg Clock, C_in;
        wire [3:0] SUM;
        wire C_out;

// Instantiate the module under test
full_adder dut  (.A(A),.B(B),.Clock(Clock),.C_in(C_in),.SUM(SUM),.C_out(C_out));

// Clock generation
always #5 Clock = ~Clock; // Clock signal with a period of 10 ns

// Stimulus
initial begin
         $fsdbDumpvars();

//Tool specific command. Creates novas.fsdb file. Used for waveform generation
//// Reset inputs
//
        A <= 0; B <= 0; C_in <= 0; Clock <= 0;
// Apply test cases
        #20 A <= 4'b0001; B <= 4'b0001; C_in <= 0; // 1 + 1 = 2 (binary: 10)
        $display("A = %b, B = %b, C_in = %b, SUM = %b, C_out = %b", A, B, C_in, SUM,
C_out);

        #20 A <= 4'b0110; B <= 4'b1010; C_in <= 1; // 6 + 10 + 1 = 17 (binary: 10001)
        $display("A = %b, B = %b, C_in = %b, SUM = %b, C_out = %b", A, B, C_in, SUM,
C_out);

        #20 A <= 4'b1000; B <= 4'b1111; C_in <= 1; // 8 + 15 + 1 = 24 (binary: 11000)
        $display("A = %b, B = %b, C_in = %b, SUM = %b, C_out = %b", A, B, C_in, SUM,
C_out);

        #100 $finish;

        end
endmodule
```

~/RTL2GDSII_TESTCASE/rtl/full_adder.v - Mousepad

File Edit Search View Document Help

```
module full_adder (A, B, C_in, C_out, Clock, SUM);
input [3:0] A, B;
input Clock, C_in;
output reg [3:0] SUM;
output reg C_out;

//Internal registers

reg [3:0] reg1, reg2, sum_i;
reg c_in, c_out;

//Inputs to Internal Registers (synchronized to clock)

always @ (posedge Clock)
        begin
                reg1 <= A;
                reg2 <= B;
                c_in <= C_in;
        end

//Output Registering Synchronized to clock

always @ (posedge Clock)
        begin
                SUM <= sum i;
```

~/RTL2GDSII_TESTCASE/rtl/full_adder_tb.v - Mousepad

File Edit Search View Document Help

```
`timescale 1ns/1ns

`include "full_adder.v" // includes the module definition for the full

module testbench;
        reg [3:0] A, B;
        reg Clock, C_in;
        wire [3:0] SUM;
        wire C_out;

// Instantiate the module under test
full_adder dut  (.A(A),.B(B),.Clock(Clock),.C_in(C_in),.SUM(SUM),.C_out

// Clock generation
always #5 Clock = ~Clock; // Clock signal with a period of 10 ns

// Stimulus
initial begin
         $fsdbDumpvars();

//Tool specific command. Creates novas.fsdb file. Used for waveform ger
//// Reset inputs
//
        A <= 0; B <= 0; C_in <= 0; Clock <= 0;
// Apply test cases
```

2. Compilation & Simulation Commands

These steps will compile the RTL and testbench, run the simulation and open the waveform viewer.

Compile the RTL design file: vcs -full64 full_adder.v -debug_access+all -lca -kdb

```
1 module and 0 UDP read.
recompiling module full_adder
rm -f _cuarc*.so _csrc*.so pre_vcsobj_*.so share_vcsobj_*.so
if [ -x ../simv ]; then chmod a-x ../simv; fi
g++  -o ../simv      -rdynamic  -Wl,-rpath='$ORIGIN'/simv.daidir -Wl,-rpath=
./simv.daidir -Wl,-rpath=/usr/synopsys/vcs/U-2023.03-SP2-7/linux64/lib -L/us
r/synopsys/vcs/U-2023.03-SP2-7/linux64/lib  -Wl,-rpath-link=./  /usr/lib64/l
ibnuma.so.1   objs/amcQw_d.o   _533824_archive_1.so   SIM_l.o      rmapats_m
op.o rmapats.o rmar.o rmar_nd.o  rmar_llvm_0_1.o rmar_llvm_0_0.o
-lvirsim -lerrorinf -lsnpsmalloc -lvfs    -lvcsnew -lsimprofile -luclinative
 /usr/synopsys/vcs/U-2023.03-SP2-7/linux64/lib/vcs_tls.o   -Wl,-whole-archiv
e  -lvcsucli    -Wl,-no-whole-archive       _vcs_pli_stub_.o   /usr/synopsy
s/vcs/U-2023.03-SP2-7/linux64/lib/vcs_save_restore_new.o /usr/synopsys/verdi
/U-2023.03-SP2-7/share/PLI/VCS/LINUX64/pli.a -ldl  -lc -lm -lpthread -ldl
../simv up to date
CPU time: .157 seconds to compile + .128 seconds to elab + .118 seconds to l
ink
Verdi KDB elaboration done and the database successfully generated: 0 error(
s), 0 warning(s)
```

Compile Testbench file : vcs -full64 full_adder_tb.v -debug_access+all -lca -kdb

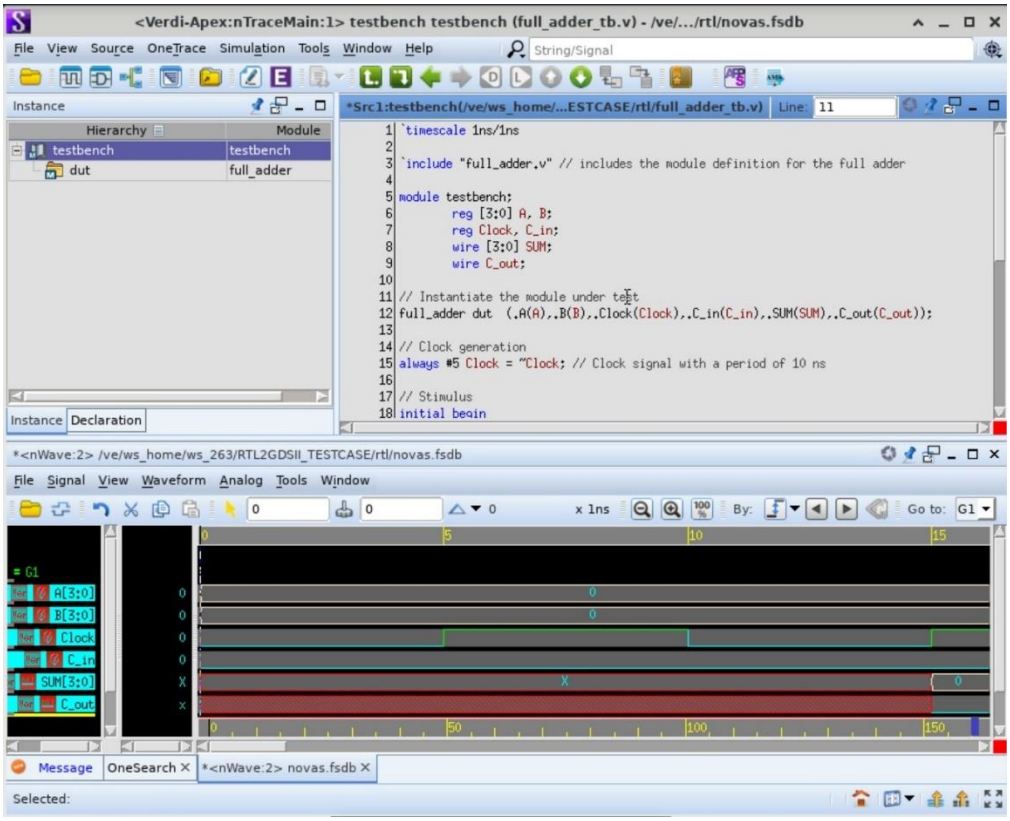
```
1 module and 0 UDP read.
recompiling module testbench
rm -f _cuarc*.so _csrc*.so pre_vcsobj_*.so share_vcsobj_*.so
if [ -x ../simv ]; then chmod a-x ../simv; fi
g++  -o ../simv      -rdynamic  -Wl,-rpath='$ORIGIN'/simv.daidir -Wl,-rpath=
./simv.daidir -Wl,-rpath=/usr/synopsys/vcs/U-2023.03-SP2-7/linux64/lib -L/us
r/synopsys/vcs/U-2023.03-SP2-7/linux64/lib  -Wl,-rpath-link=./  /usr/lib64/l
ibnuma.so.1   objs/amcQw_d.o   _534539_archive_1.so   SIM_l.o      rmapats_m
op.o rmapats.o rmar.o rmar_nd.o  rmar_llvm_0_1.o rmar_llvm_0_0.o
-lvirsim -lerrorinf -lsnpsmalloc -lvfs    -lvcsnew -lsimprofile -luclinative
 /usr/synopsys/vcs/U-2023.03-SP2-7/linux64/lib/vcs_tls.o   -Wl,-whole-archiv
e  -lvcsucli    -Wl,-no-whole-archive       _vcs_pli_stub_.o   /usr/synopsy
s/vcs/U-2023.03-SP2-7/linux64/lib/vcs_save_restore_new.o /usr/synopsys/verdi
/U-2023.03-SP2-7/share/PLI/VCS/LINUX64/pli.a -ldl  -lc -lm -lpthread -ldl
../simv up to date
CPU time: .151 seconds to compile + .125 seconds to elab + .111 seconds to l
ink
Verdi KDB elaboration done and the database successfully generated: 0 error(
s), 0 warning(s)
```

Run simulation : ./simv verdi

```
l64;  Oct 15 21:19 2025
*Verdi* Loading libsscore_vcs202303.so
FSDB Dumper for VCS, Release Verdi_U-2023.03-SP2-7, Linux x86_64/64bit, 06/2
3/2024
(C) 1996 - 2024 by Synopsys, Inc.
*Verdi* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB fil
e may crash the programs that are using this file.
*Verdi* : Create FSDB file 'novas.fsdb'
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : End of traversing.
A = 0000, B = 0000, C_in = 0, SUM = 0000, C_out = 0
A = 0001, B = 0001, C_in = 0, SUM = 0010, C_out = 0
A = 0110, B = 1010, C_in = 1, SUM = 0001, C_out = 1
$finish called from file "full_adder_tb.v", line 35.
$finish at simulation time                  160
          V C S   S i m u l a t i o n   R e p o r t
Time: 160 ns
CPU Time:      0.120 seconds;       Data structure size:   0.0Mb
Wed Oct 15 21:19:20 2025
```

3. Waveform Analysis Using Verdi

View waveform in Verdi: verdi -ssf novas.fsdb -nologo



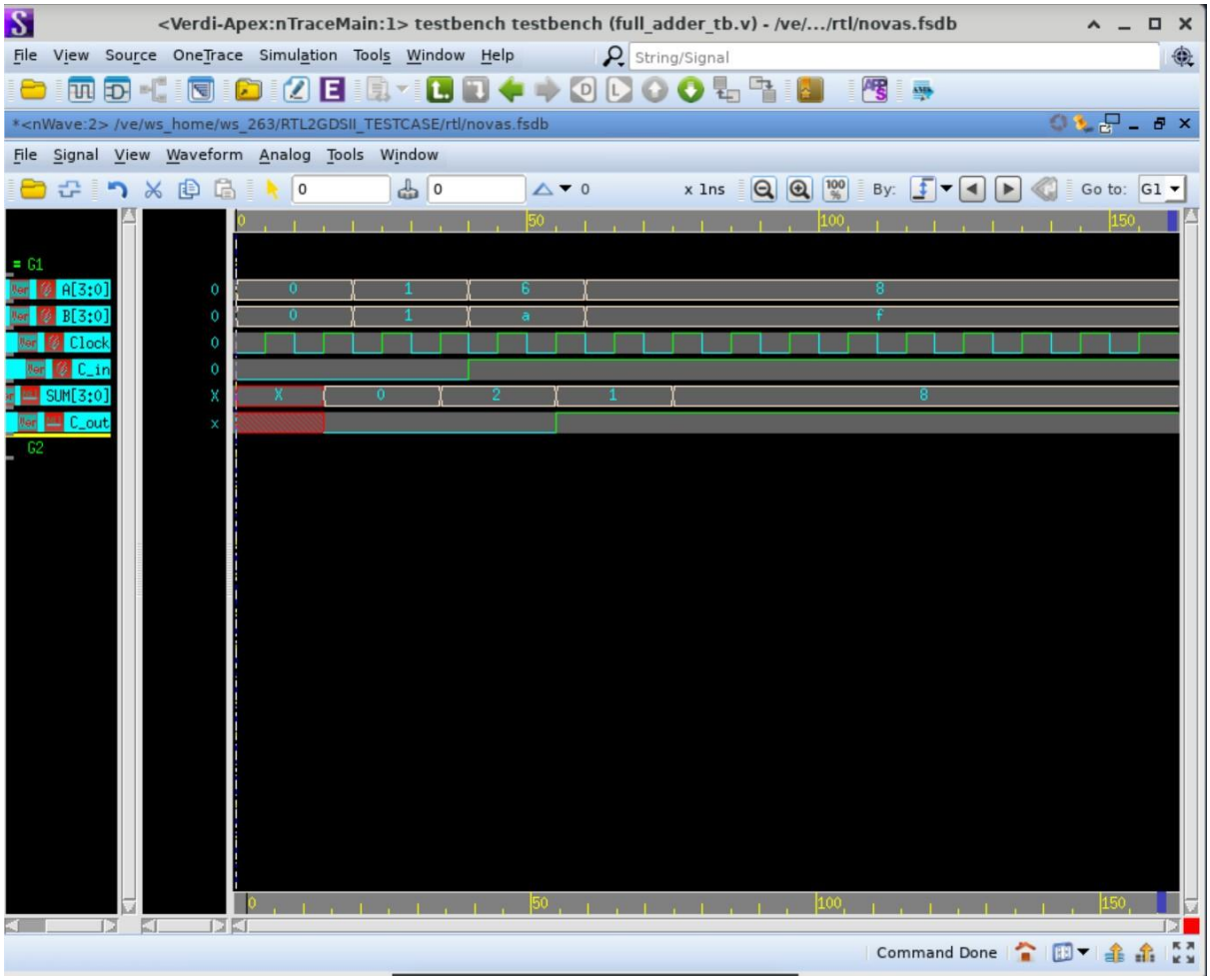


Figure 1: Verdi waveform output of 4-bit Full Adder showing input, output, and carry signals.

1. Input Signals:

- **A[3:0], B[3:0] = 4-bit input operands.**
- **C_in** = Carry input
- **Clock** = Synchronization signal for the testbench.

2. Output Signals:

- **SUM[3:0] = 4-bit sum output.**
- **C_out = Carry-out bit**

3. Observation from Waveform:

- Different combinations of A and B are applied at every clock cycle.
- The **SUM** and **Cout** outputs correctly follow 4-bit addition logic.
- Example:
  - A=0000, B=0000 → SUM=0000, Cout=0
  - A=0110, B=0010 → SUM=1000, Cout=0
  - A=1111, B=0001 → SUM=0000, Cout=1

View Schematic on Verdi :

Schematic -1: Shows **testbench connections** with INIT and inverter.

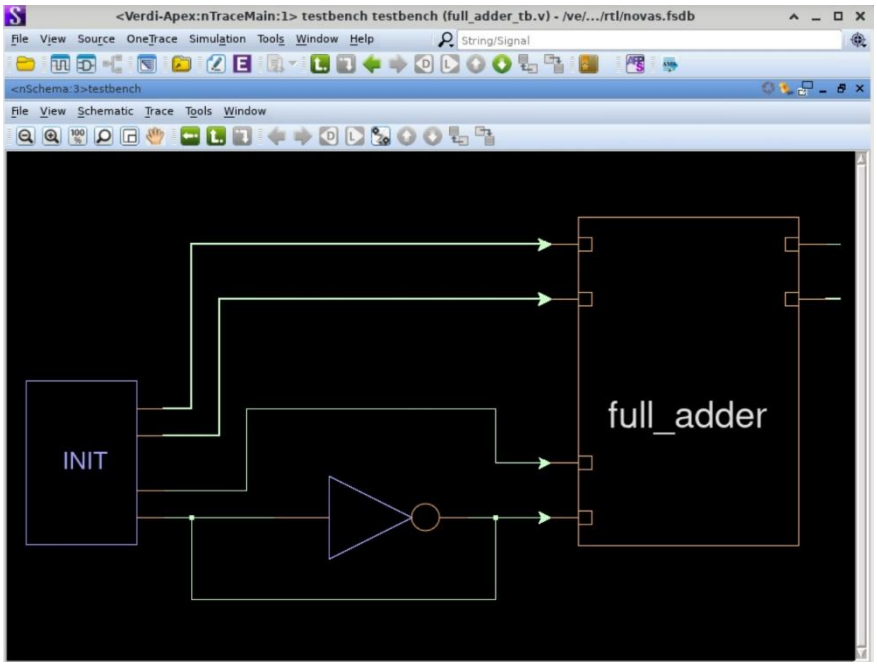


Figure 2: Verdi schematic of testbench connections to full adder

INIT Block (Left)

- Generates testbench stimulus signals for the **full_adder** module.
- It has 3 outputs (A, B, Cin).

Inverter (Middle)

- One of the INIT outputs is inverted before being fed to the full_adder.
- (Likely inverting Cin or B to create some variation in the test pattern).

Full Adder Block (Right)

- The three input signals (two direct + one inverted) go into the **full_adder** module.
- It performs the addition logic (A + B + Cin).

Connections

- 2 INIT outputs → directly to full_adder inputs.
- 1 INIT output → inverter → then to full_adder input.

Schematic-2: Shows **actual DUT (full adder)** structure.

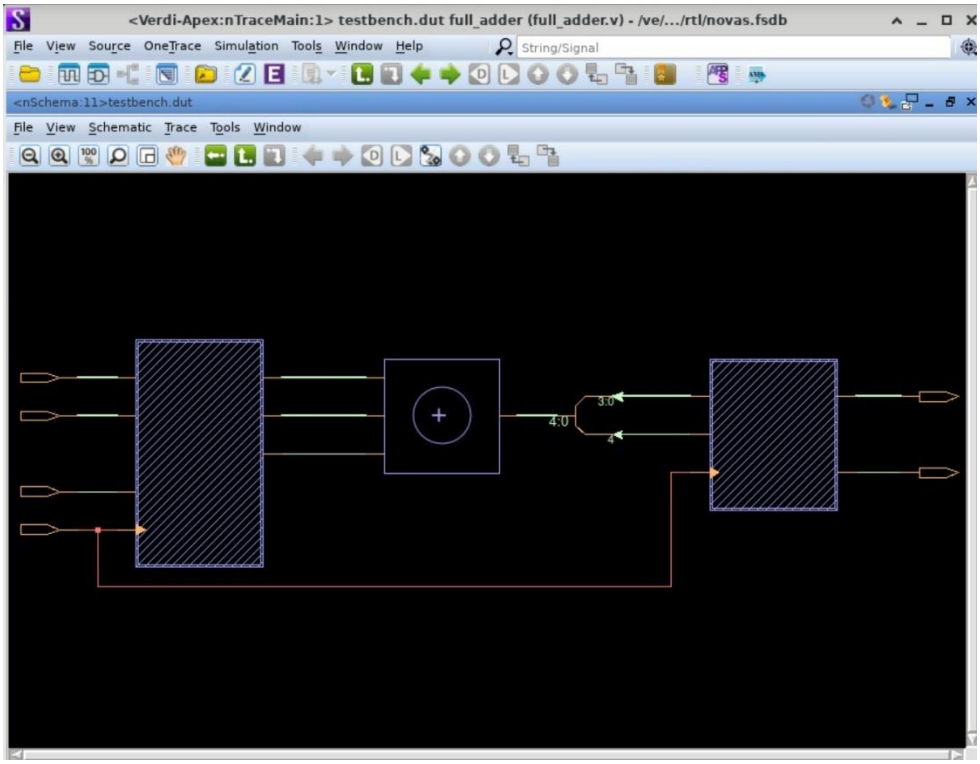


Figure 3: Verdi schematic of full adder DUT internal connections

Left Block (Inputs)

- Three input signals (A, B, Cin) enter from the left into the **full adder** block.
- This block likely represents the **DUT input ports**.

Middle Block (Adder)

- The circle with a "+" indicates the **core full adder logic**, summing the three inputs.

Right Block (Outputs)

- The outputs (Sum and Cout) are sent to the **output block** on the right.
- These are the DUT outputs.

Feedback Path (Bottom Red Line)

- One of the outputs is looped back to an input - likely used for additional testing or sequential check (e.g., carry feedback).

Logic Synthesis using DC_Shell

Process Design Kit (PDK)

A Process Design Kit (PDK) is a collection of files and data provided by a semiconductor foundry that enables designers to create circuits compatible with a specific manufacturing technology node (e.g., 32nm, 45nm, etc.).

PDK Used

- **Technology:** SAED32nm
- **Metal Stack:** 1P9M (1 polysilicon layer, 9 metal layers)
- **Operating Voltages:** 1.05 V / 1.8 V / 2.5 V

```
[ws_262@wsv3 RTL2GDSII_TESTCASE]$ cd ref
[ws_262@wsv3 ref]$ tree
.
├── lib
│   ├── ndm
│   │   └── saed32rvt_c.ndm
│   └── stdcell_rvt
│       ├── saed32rvt_ff1p16v125c.db
│       ├── saed32rvt_ff1p16v125c.lib
│       ├── saed32rvt_ss0p7vn40c.db
│       ├── saed32rvt_ss0p7vn40c.lib
│       ├── saed32rvt_tt0p78vn40c.db
│       └── saed32rvt_tt0p78vn40c.lib
└── tech
    ├── milkyway
    │   └── saed32nm_1p9m_mw.tf
    └── star_rcxt
        ├── saed32nm_1p9m_Cmax.tluplus
        ├── saed32nm_1p9m_Cmin.tluplus
        └── saed32nm_tf_itf_tluplus.map

6 directories, 11 files
```

Logic Synthesis

Logic synthesis converts high-level RTL (Register Transfer Level) code (written in Verilog/VHDL) into a gate-level netlist composed of standard cells from the library.
It bridges the gap between design intent (RTL) and implementation (gates).

Inputs to Logic Synthesis

Input File	Description
RTL Code	The functional hardware description in Verilog/VHDL.
Library Files (.lib, .db)	Contain timing, power, and area data for each standard cell.
Link Library (link_lib)	Provides references for resolving module instances.
Target Library (target_lib)	Defines the available cells for synthesis mapping.
Symbol Library	Used for schematic visualization.
SDC File (Synopsys Design Constraints)	Defines timing, clock, input/output delays, and design constraints.
UPF File (Unified Power Format)	Defines power domains and power intent for multi-voltage designs.
RLC File	Describes resistance, inductance, and capacitance information for interconnects.
LEF File	Used in physical design; contains cell dimensions and pin locations.

Steps Performed:

1. We first go to the directory: cd RTL2GDSII_TESTCASE/DC

dc_shell

```
[ws_262@wsv3 ~]$ cp -r /data/labs/ws/RTL2GDSII_TESTCASE/ .
[ws_262@wsv3 ~]$ cd DC
bash: cd: DC: No such file or directory
[ws_262@wsv3 ~]$ cd RTL2GDSII_TESTCASE
[ws_262@wsv3 RTL2GDSII_TESTCASE]$ cd DC
[ws_262@wsv3 DC]$ dc_shell
Information: License queuing is enabled. (DCSH-18)

                           Design Compiler Graphical
```

2. Here we go to the **rm_setup** directory and open the **common_setup.tcl** in the nano/vi editor.

```
[ws_262@wsv3 DC]$ ls
command.log  filenames.log  reports  results  rm_setup  run_dc.tcl  WORK
[ws_262@wsv3 DC]$ cd rm_setup
[ws_262@wsv3 rm_setup]$ ls
common_setup.tcl  dc_setup_filenames.tcl  dc_setup.tcl
[ws_262@wsv3 rm_setup]$ nano common_setup.tcl
```

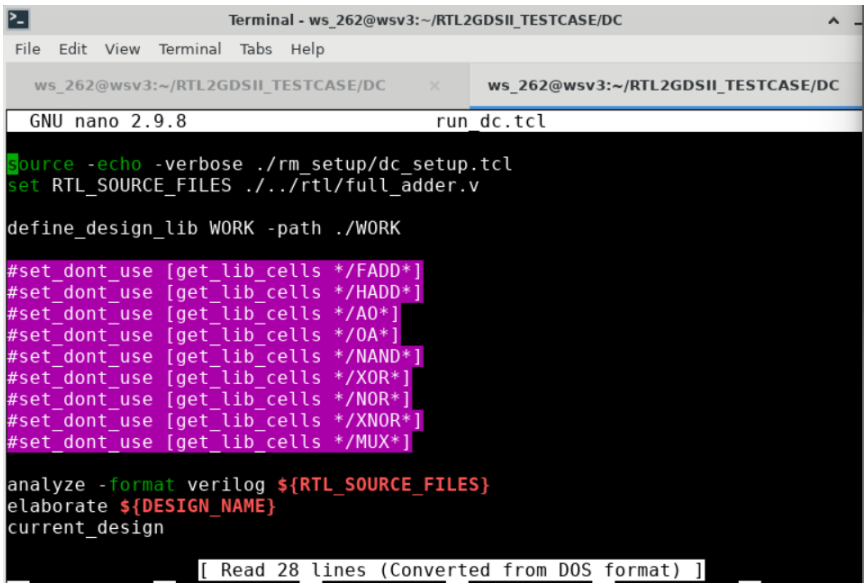
Here we :

i) set DESIGN_NAME “full_adder”

ii) set Target Library Files: set PDK_PATH “./../ref/”

```
  GNU nano 2.9.8                    common_setup.tcl

puts "RM-Info: Running script [info script]\n"

################################################################################
# Variables common to all reference methodology scripts
# Script: common_setup.tcl
# Version: N-2017.09-SP4 (April 23, 2018)
# Copyright (C) 2007-2017 Synopsys, Inc. All rights reserved.
################################################################################

set DESIGN_NAME                 "full_adder"  ;#  The name of the top-leve
set PDK_PATH                  "./../ref/" ; # to set the PDK path for the
set DESIGN_REF_DATA_PATH          ""  ;#  Absolute path prefix variable for
                                      #  Use this variable to prefix the co
                                      #  to the common variables defined be
```

3. We then go to **run_dc.tcl** and open it in vi editor:



Here we:

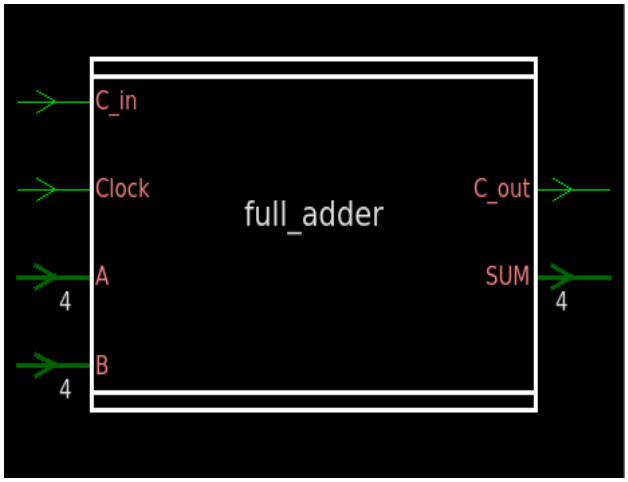
i) set RTL_SOURCE_FILES ./../rtl/full_adder.v
ii) Didn't use cells OI,AO,NAND,MUX
iii) read_sdc ./../CONSTRAINTS/decoder.sdc
iv) compile ultra

👉 *Copy and paste: commands from the run_dc.tcl file*

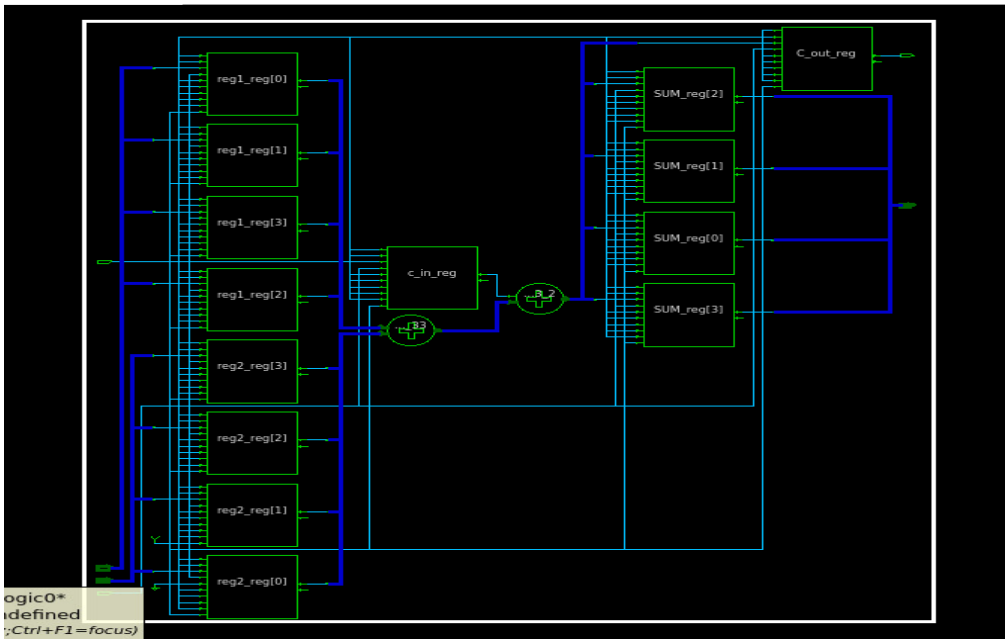
4. In DC Shell give command **start_gui**.

In the gui: Design Vision Block Diagram

Schematic -1



Schematic-2



5. Now, before compiling, the mapped.v file

```
Terminal - ws_262@wsv3:~/RTL2GDSII_TESTCASE/DC
File  Edit  View  Terminal  Tabs  Help
ws_262@wsv3:~/RTL2GDS...   ws_262@wsv3:~/RTL2GDS...   ws_262@wsv3:~/RTL2GDS...

output [3:0] SUM;
input C_in, Clock;
output C_out;
wire   c_in, c_out, N0, N1, N2, N3, N4;
wire   [3:0] reg1;
wire   [3:0] reg2;
wire   [3:0] sum_i;

\**SEQGEN**  \reg1_reg[3]  ( .clear(1'b0), .preset(1'b0), .next_state(A[3]),
      .clocked_on(Clock), .data_in(1'b0), .enable(1'b0), .Q(reg1[3]),
      .synch_clear(1'b0), .synch_preset(1'b0), .synch_toggle(1'b0),
      .synch_enable(1'b1) );
\**SEQGEN**  \reg1_reg[2]  ( .clear(1'b0), .preset(1'b0), .next_state(A[2]),
      .clocked_on(Clock), .data_in(1'b0), .enable(1'b0), .Q(reg1[2]),
      .synch_clear(1'b0), .synch_preset(1'b0), .synch_toggle(1'b0),
      .synch_enable(1'b1) );
\**SEQGEN**  \reg1_reg[1]  ( .clear(1'b0), .preset(1'b0), .next_state(A[1]),
      .clocked_on(Clock), .data_in(1'b0), .enable(1'b0), .Q(reg1[1]),
      .synch_clear(1'b0), .synch_preset(1'b0), .synch_toggle(1'b0),
      .synch_enable(1'b1) );
\**SEQGEN**  \reg1_reg[0]  ( .clear(1'b0), .preset(1'b0), .next_state(A[0]),
      .clocked_on(Clock), .data_in(1'b0), .enable(1'b0), .Q(reg1[0]),
      .synch_clear(1'b0), .synch_preset(1'b0), .synch_toggle(1'b0),
                                                          28,1          17%
```

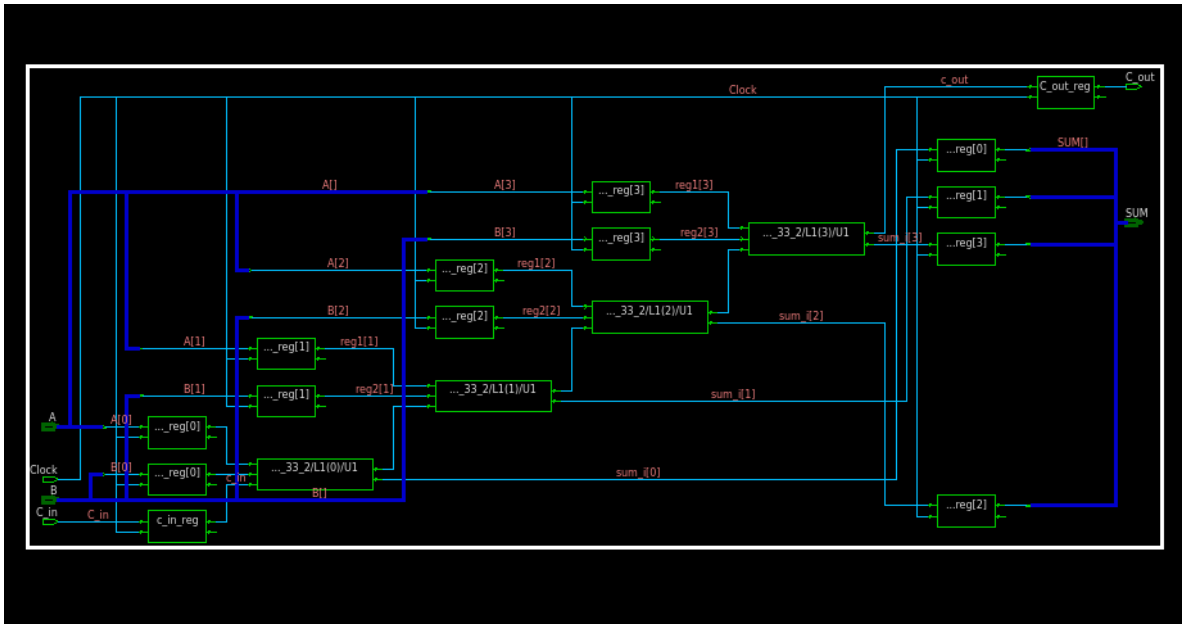
After compiling

In a new tab open **full_adder.mapped.v**

```
[ws_262@wsv3 DC]$ ls results
full_adder.mapped_old.v  full_adder.mapped.v
[ws_262@wsv3 DC]$
```

```
ws_262@wsv3:~/RTL2GDS...   ws_262@wsv3:~/RTL2GDS...   ws_262@wsv3:~/RTL2GDS...
 GNU nano 2.9.8              ./results/full_adder.mapped.v

output [3:0] SUM;
input C_in, Clock;
output C_out;
wire   c_in, c_out;
wire   [3:0] reg1;
wire   [3:0] reg2;
wire   [3:0] sum_i;
wire   [4:1] \add_1_root_add_33_2/carry ;

DFFX1_RVT \reg1_reg[3]  ( .D(A[3]), .CLK(Clock), .Q(reg1[3]) );
DFFX1_RVT \reg1_reg[2]  ( .D(A[2]), .CLK(Clock), .Q(reg1[2]) );
DFFX1_RVT \reg1_reg[1]  ( .D(A[1]), .CLK(Clock), .Q(reg1[1]) );
DFFX1_RVT \reg1_reg[0]  ( .D(A[0]), .CLK(Clock), .Q(reg1[0]) );
DFFX1_RVT \reg2_reg[3]  ( .D(B[3]), .CLK(Clock), .Q(reg2[3]) );
DFFX1_RVT \reg2_reg[2]  ( .D(B[2]), .CLK(Clock), .Q(reg2[2]) );
DFFX1_RVT \reg2_reg[1]  ( .D(B[1]), .CLK(Clock), .Q(reg2[1]) );
DFFX1_RVT \reg2_reg[0]  ( .D(B[0]), .CLK(Clock), .Q(reg2[0]) );
DFFX1_RVT c_in_reg ( .D(C_in), .CLK(Clock), .Q(c_in) );
DFFX1_RVT \SUM_reg[3]  ( .D(sum_i[3]), .CLK(Clock), .Q(SUM[3]) );
                          [ Read 45 lines ]
```

Schematic-3 after Compiling

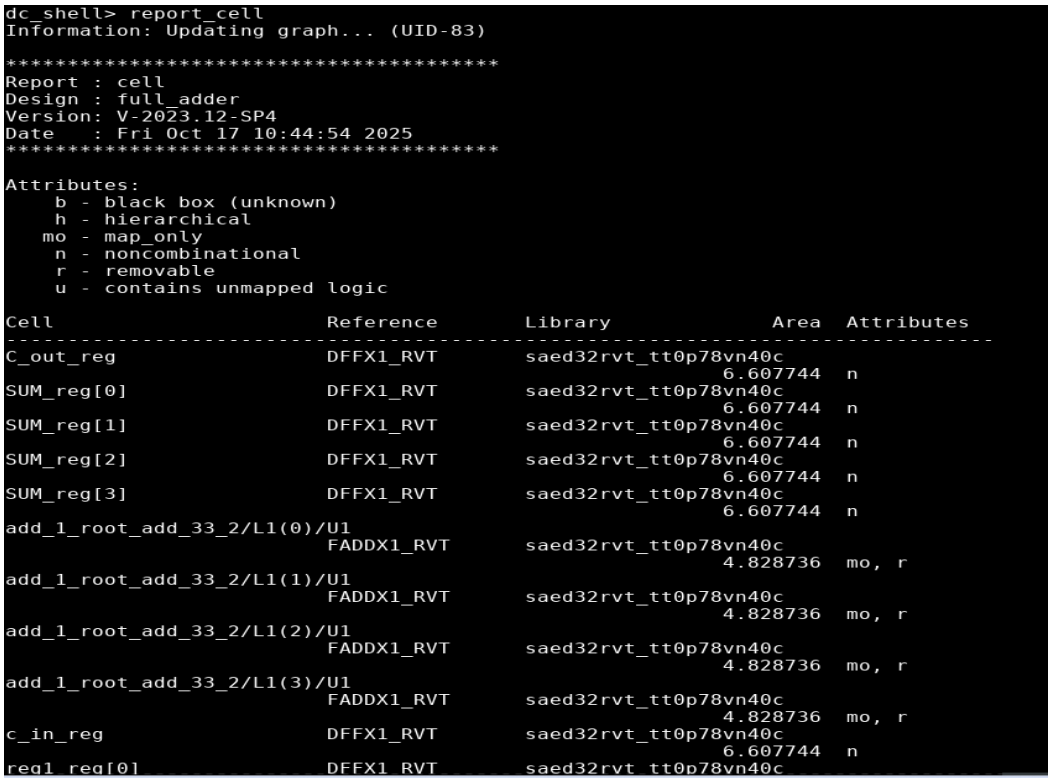


6. Now also check the **cell counts**, **area** and **slack** with commands **report_qor** and **report_timing**.

Picture 1: **Cell Count (report_qor)**

```
Cell Count
---------------------------------------
Hierarchical Cell Count:              0
Hierarchical Port Count:              0
Leaf Cell Count:                     19
Buf/Inv Cell Count:                   1
Buf Cell Count:                       1
Inv Cell Count:                       0
Combinational Cell Count:             5
   Single-bit Isolation Cell Count:                    0
   Multi-bit Isolation Cell Count:                     0
   Isolation Cell Banking Ratio:                       0.00%
   Single-bit Level Shifter Cell Count:                0
   Multi-bit Level Shifter Cell Count:                 0
   Level Shifter Cell Banking Ratio:                   0.00%
   Single-bit ELS Cell Count:                          0
   Multi-bit ELS Cell Count:                           0
   ELS Cell Banking Ratio:                             0.00%
Sequential Cell Count:               14
   Integrated Clock-Gating Cell Count:                 0
   Sequential Macro Cell Count:                        0
   Single-bit Sequential Cell Count:                   14
   Multi-bit Sequential Cell Count:                    0
   Sequential Cell Banking Ratio:                      0.00%
   BitsPerflop:                                        1.00
Macro Count:                          0
---------------------------------------


Area
---------------------------------------
Combinational Area:              21.35
Noncombinational Area:           92.51
Buf/Inv Area:                     2.03
Total Buffer Area:                2.03
Total Inverter Area:              0.00
Macro/Black Box Area:             0.00
Net Area:                            0
```

Picture 2: **Slack (report_timing)**



```
dc_shell> report_cell
Information: Updating graph... (UID-83)

****************************************
Report : cell
Design : full_adder
Version: V-2023.12-SP4
Date   : Fri Oct 17 10:44:54 2025
****************************************

Attributes:
    b - black box (unknown)
    h - hierarchical
   mo - map_only
    n - noncombinational
    r - removable
    u - contains unmapped logic

Cell                      Reference       Library             Area  Attributes
--------------------------------------------------------------------------------
C_out_reg                 DFFX1_RVT       saed32rvt_tt0p78vn40c
                                                            6.607744  n
SUM_reg[0]                DFFX1_RVT       saed32rvt_tt0p78vn40c
                                                            6.607744  n
SUM_reg[1]                DFFX1_RVT       saed32rvt_tt0p78vn40c
                                                            6.607744  n
SUM_reg[2]                DFFX1_RVT       saed32rvt_tt0p78vn40c
                                                            6.607744  n
SUM_reg[3]                DFFX1_RVT       saed32rvt_tt0p78vn40c
                                                            6.607744  n
add_1_root_add_33_2/L1(0)/U1
                          FADDX1_RVT      saed32rvt_tt0p78vn40c
                                                            4.828736  mo, r
add_1_root_add_33_2/L1(1)/U1
                          FADDX1_RVT      saed32rvt_tt0p78vn40c
                                                            4.828736  mo, r
add_1_root_add_33_2/L1(2)/U1
                          FADDX1_RVT      saed32rvt_tt0p78vn40c
                                                            4.828736  mo, r
add_1_root_add_33_2/L1(3)/U1
                          FADDX1_RVT      saed32rvt_tt0p78vn40c
                                                            4.828736  mo, r
c_in_reg                  DFFX1_RVT       saed32rvt_tt0p78vn40c
                                                            6.607744  n
reg1_reg[0]               DFFX1_RVT       saed32rvt_tt0p78vn40c
```

Conclusion:

The RTL design was successfully mapped to standard cells and is ready for **further stages (Floorplanning & PnR)**.

Physical Design using ICC2_Shell

Tool Used:

Synopsys IC Compiler II (ICC2) for backend implementation.

Objective:

Convert the synthesized gate-level netlist into an optimized layout (GDSII) ready for fabrication.

Steps Performed:

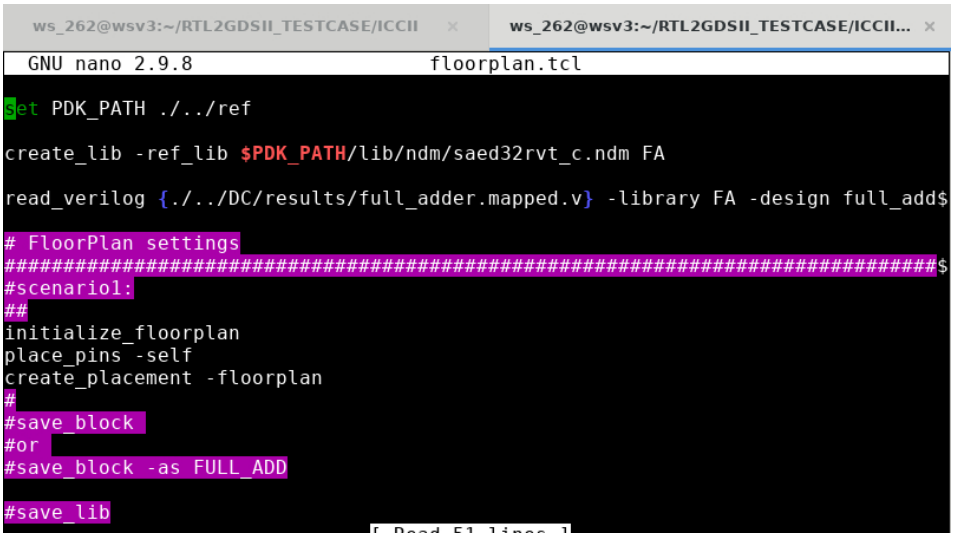
1. Go to the RTL2GDSII directory and further go to the directory **ICCII**.
2. Open the ICC2 Shell by running the command **icc2_shell** and **start_gui** later.

Floorplanning: To define the core area, I/O pin locations, and placement regions for the 4-bit Full Adder design before placement.

3. Now in ICCII directory, we will further open scripts directory. Open **floorplan.tcl** in nano/vi editor and create library.

```
[ws_262@wsv3 ICCII]$ cd scripts
[ws_262@wsv3 scripts]$ ls
clock.tcl  floorplan.tcl  placement.tcl  power_planning.tcl  route.tcl
[ws_262@wsv3 scripts]$ nano floorplan.tcl
```

a. create_lib -ref_lib $PDK_PATH/lib/ndm/saed32rvt_c.ndm FA
b. read_verilog {./../DC/results/full_adder.mapped.v} -library FA -design full_adder -top full_adder



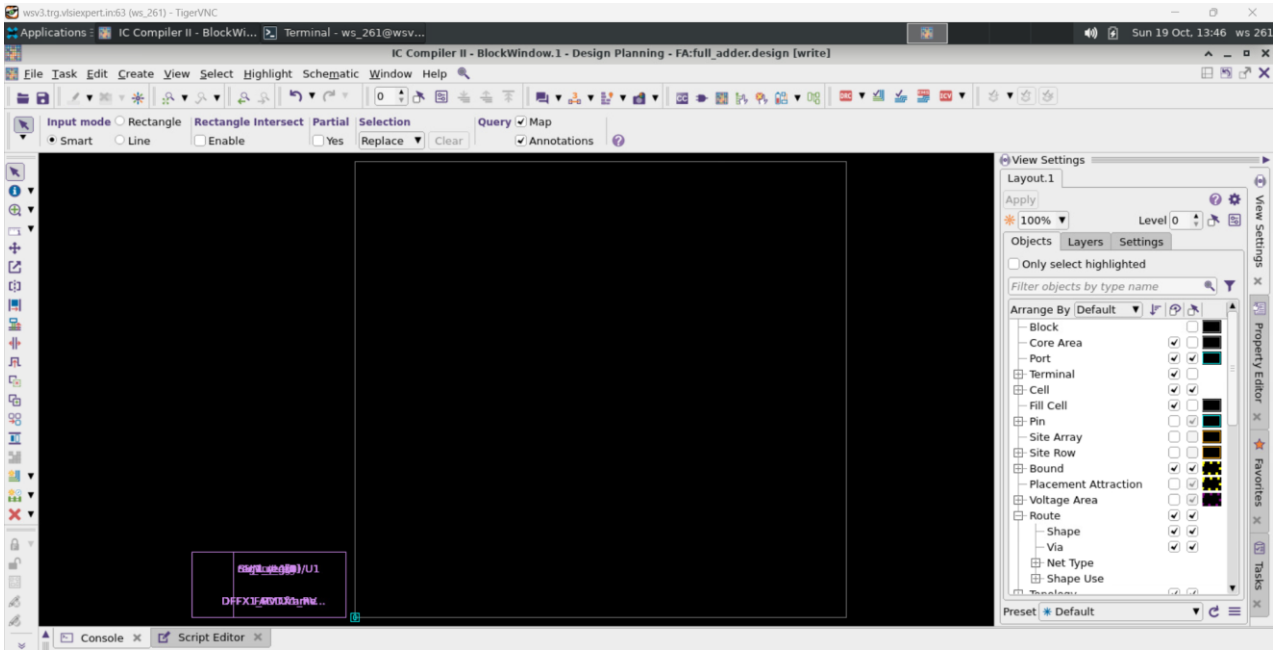
c. open_lib full_adder
d. open_block FA

```
mber of modules read: 1
p level ports: 15
tal ports in all modules: 15
tal nets in all modules: 33
tal instances in all modules: 18
apsed = 00:00:00.00, CPU = 00:00:00.00

c2_shell>
c2_shell> link_block
ing libraries: FA saed32rvt_c
nking block FA:full_adder.design
formation: User units loaded from library 'sa
sign 'full_adder' was successfully linked.

c2_shell> initialize_floorplan
moving existing floorplan objects
eating core...
re utilization ratio = 75.73%
placing all cells...
eating site array...
eating routing tracks...
itializing floorplan completed.
c2_shell> start_gui
```

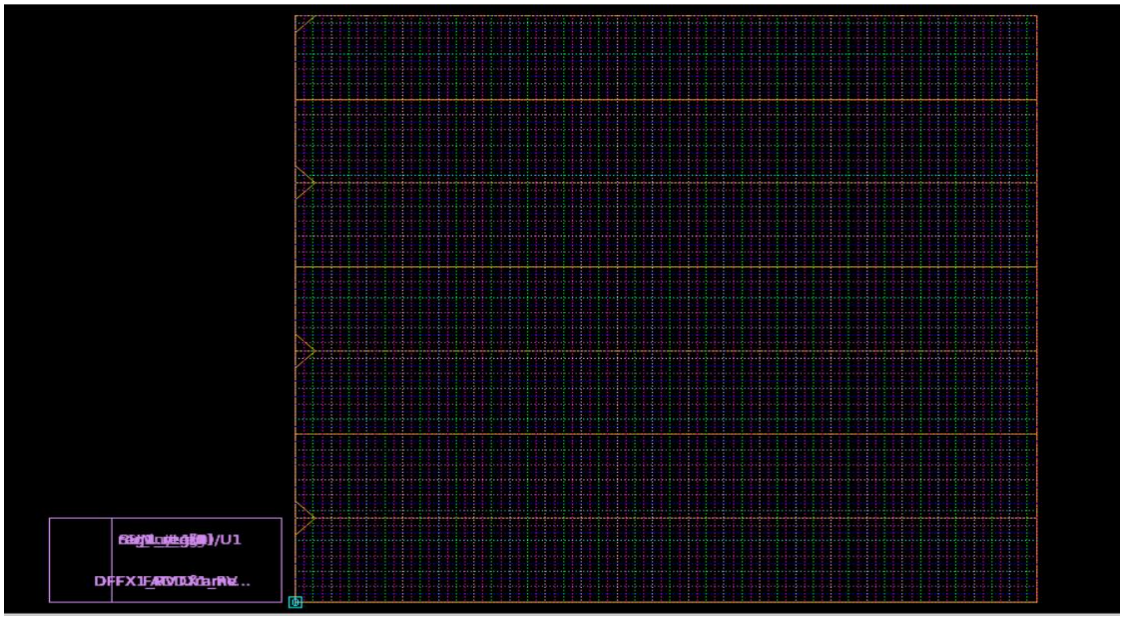
🧩 Assignment / Scenario 1–4

e. Floorplan Settings: Scenario 1

initialize_floorplan

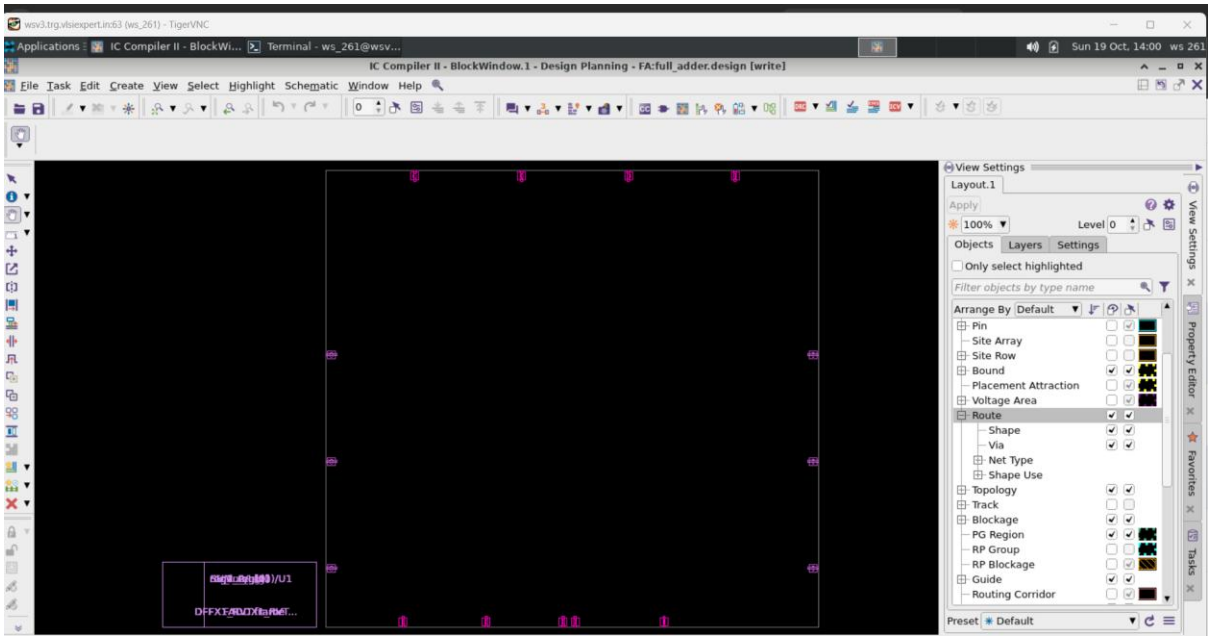


Check the boxes site arrays, site rows, tracks to view:



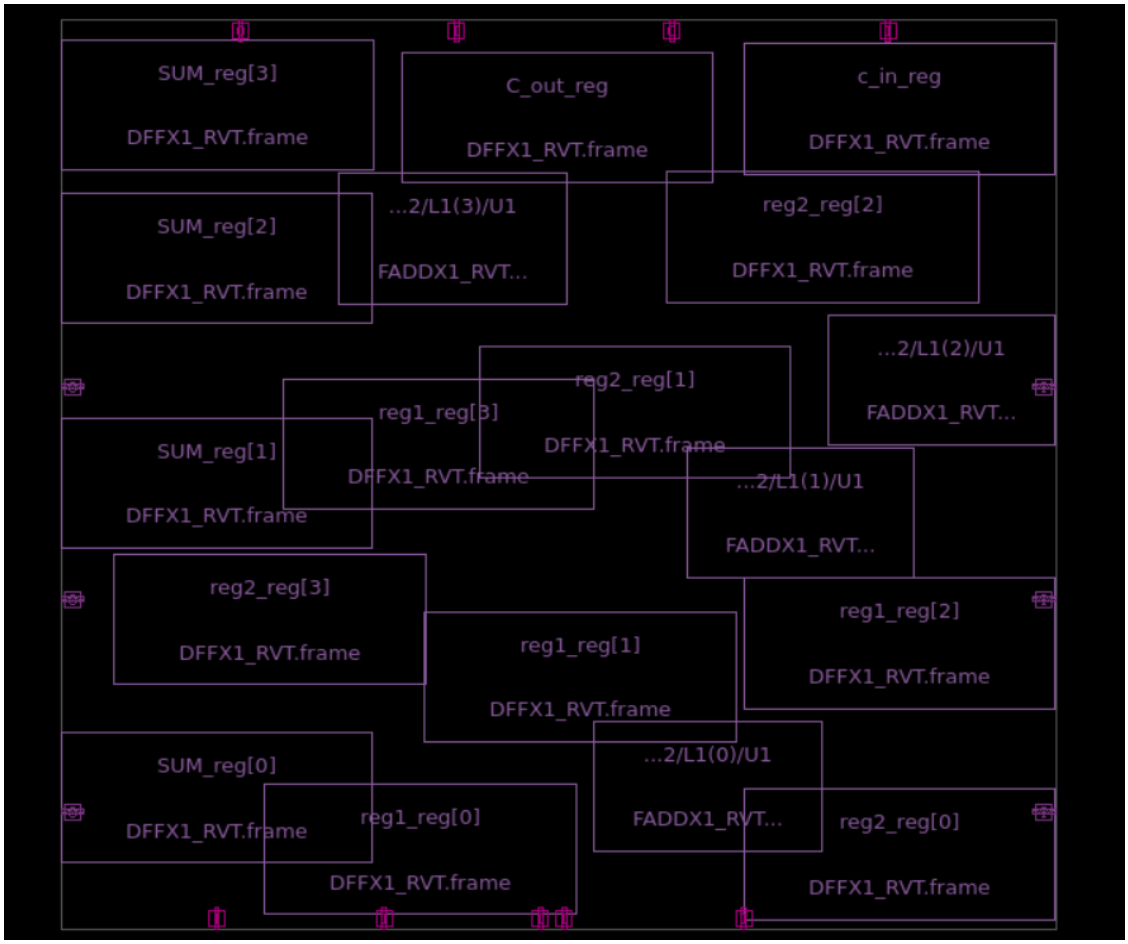
place_pins -self

You could see the pins placed in the block



create_placement -floorplan

This will place the standard cells as per



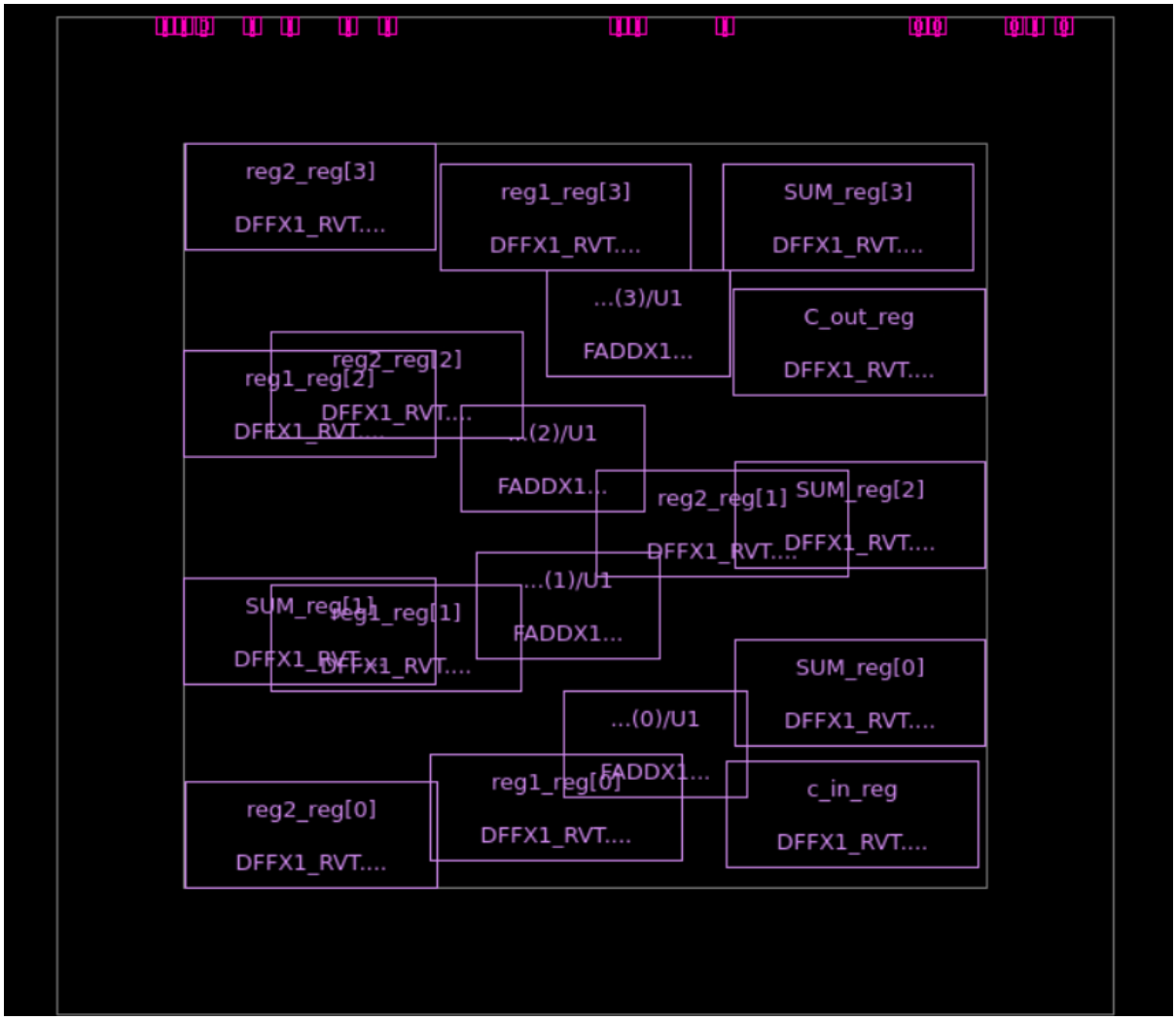
Floorplan Settings: Scenario 2

initialize_floorplan -core_offset 2

set_individual_pin_constraints -ports [get_ports] -sides 2

place_pins -self

create_placement -floorplan -effort very_low



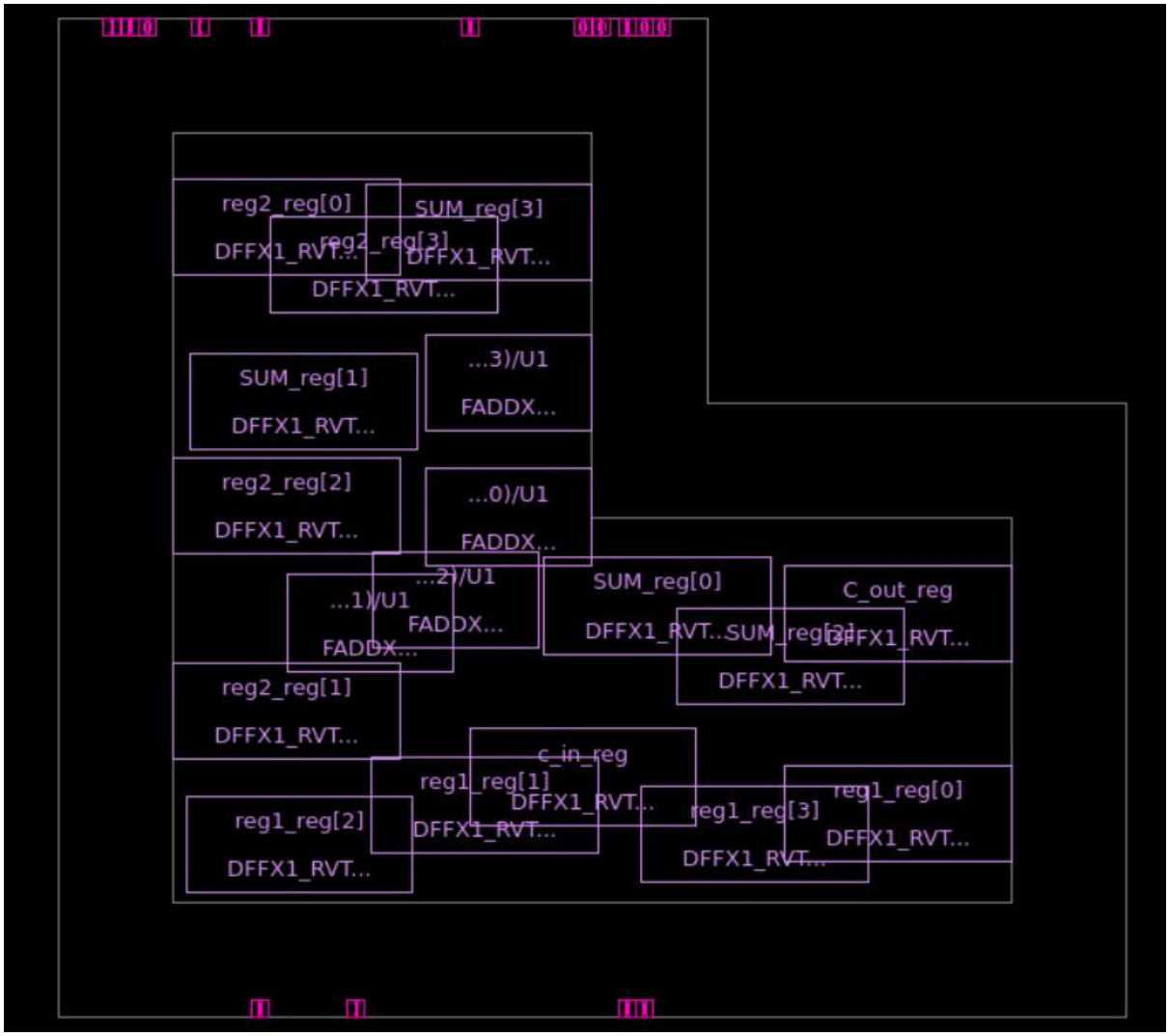
Floorplan Settings: Scenario 3

initialize_floorplan -shape L -core_offset 2 -coincident_boundary true

set_individual_pin_constraints -ports [get_ports {A}] -sides 6

place_pins -self

create_placement -floorplan -effort medium



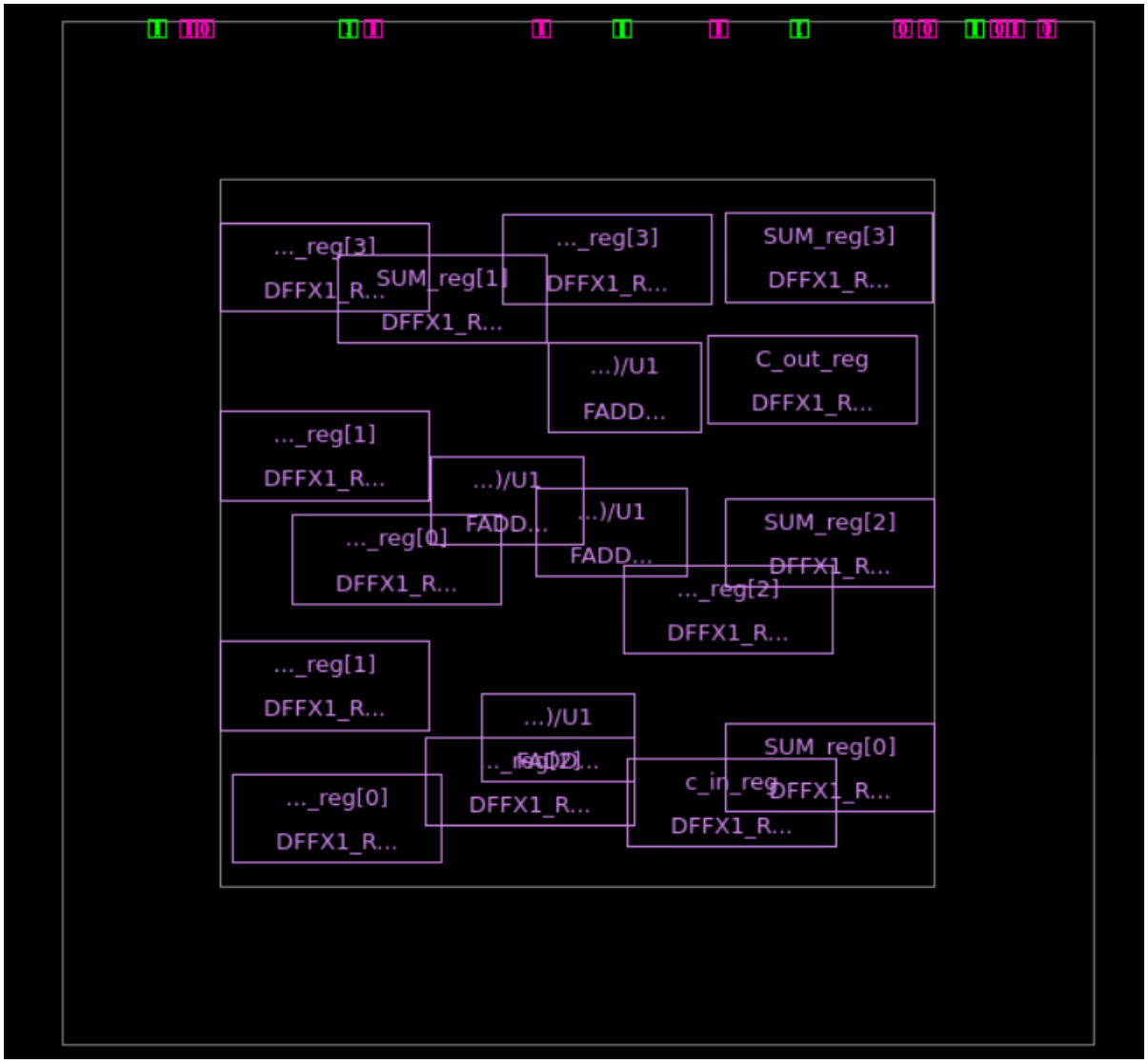
Floorplan Settings: Scenario 3

initialize_floorplan -core_utilization 0.6 -core_offset {3 3} -coincident_boundary false

set_individual_pin_constraints -ports [get_ports {A B}] -sides 2 -pin_spacing_distance 3

place_pins -self

create_placement -floorplan -effort high



f. Now run the **floorplan.tcl** file in **the icc2_shell** and then view the diagram created in icc2_shell.

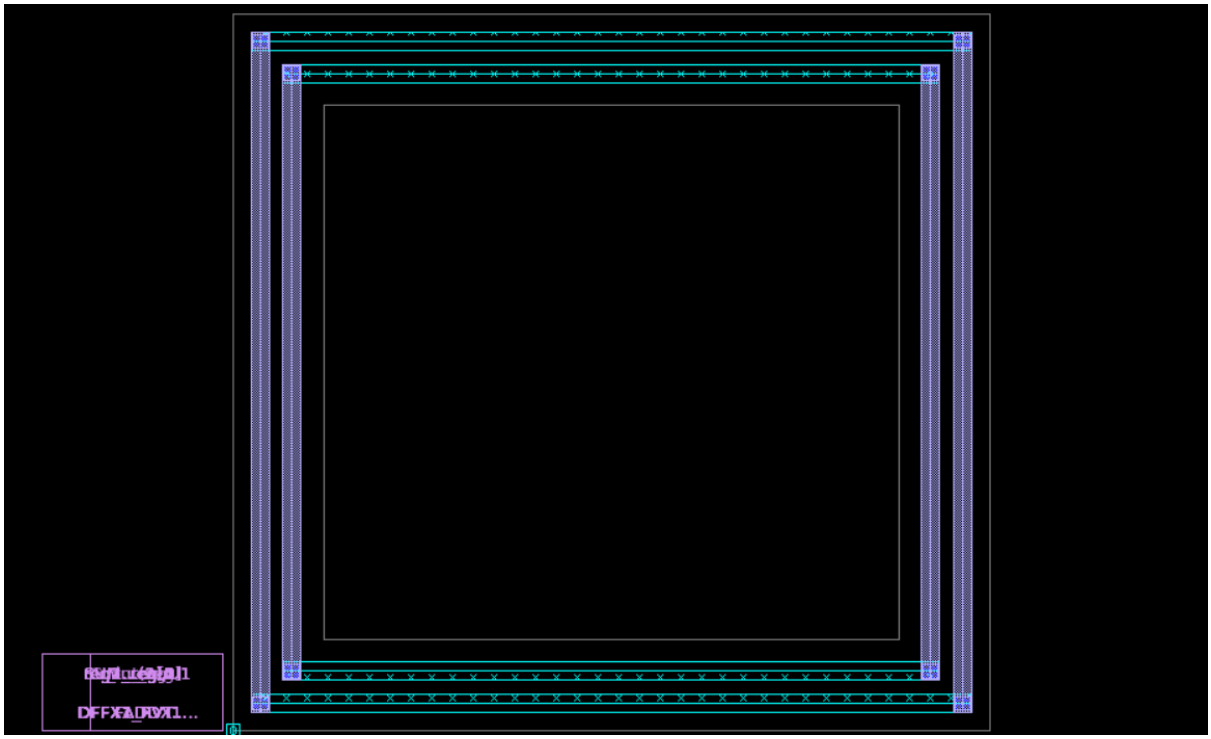
g. Also run following commands: **report_qor, report_power** & **report_units** to get detailed information on various parameters.

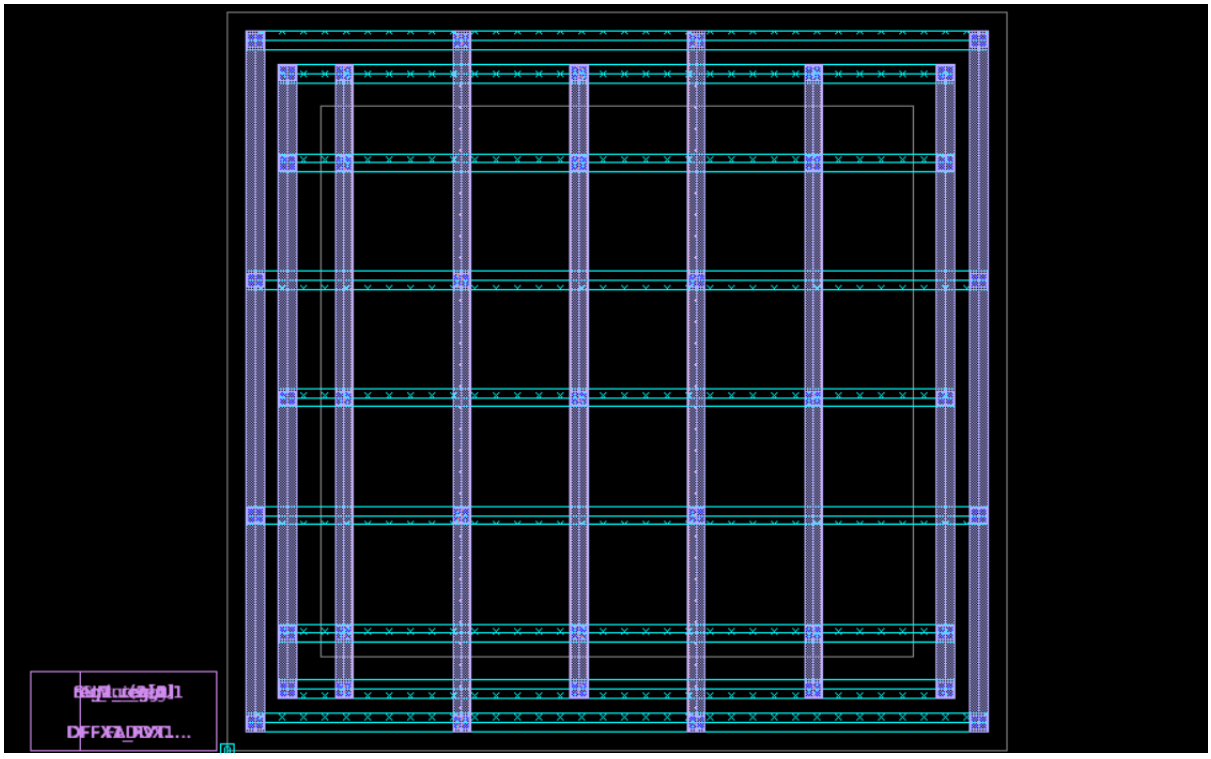
Powerplanning

To ensure proper **power distribution** (VDD and VSS) across the 4-bit Full Adder layout without voltage drop or IR issues.

4. Now run the command: source home/RTL2GDSII_TESTCASE/ICCII/scripts/power_planning.tcl and observe the changes in ICC2 Shell.
    a. Run the power_planning.tcl to create power ring and power mesh.

b. Check the VSS and VDD Connections by selecting the nets and checking.

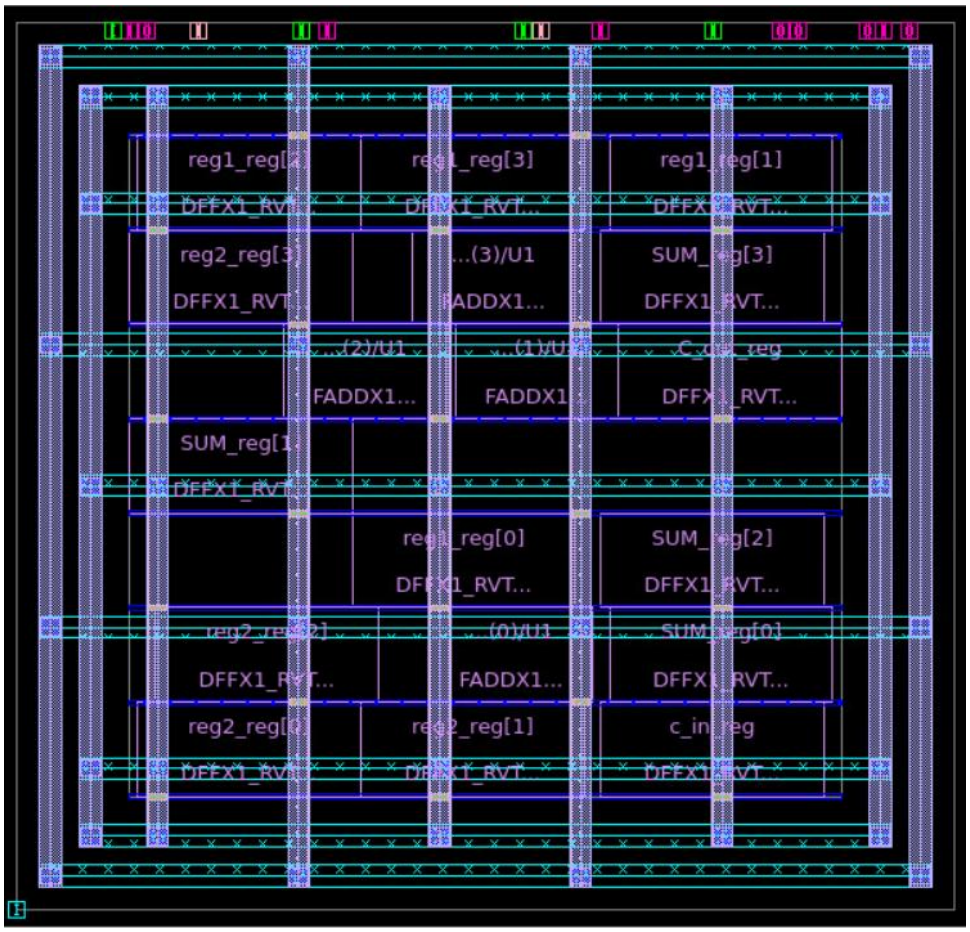




Placement

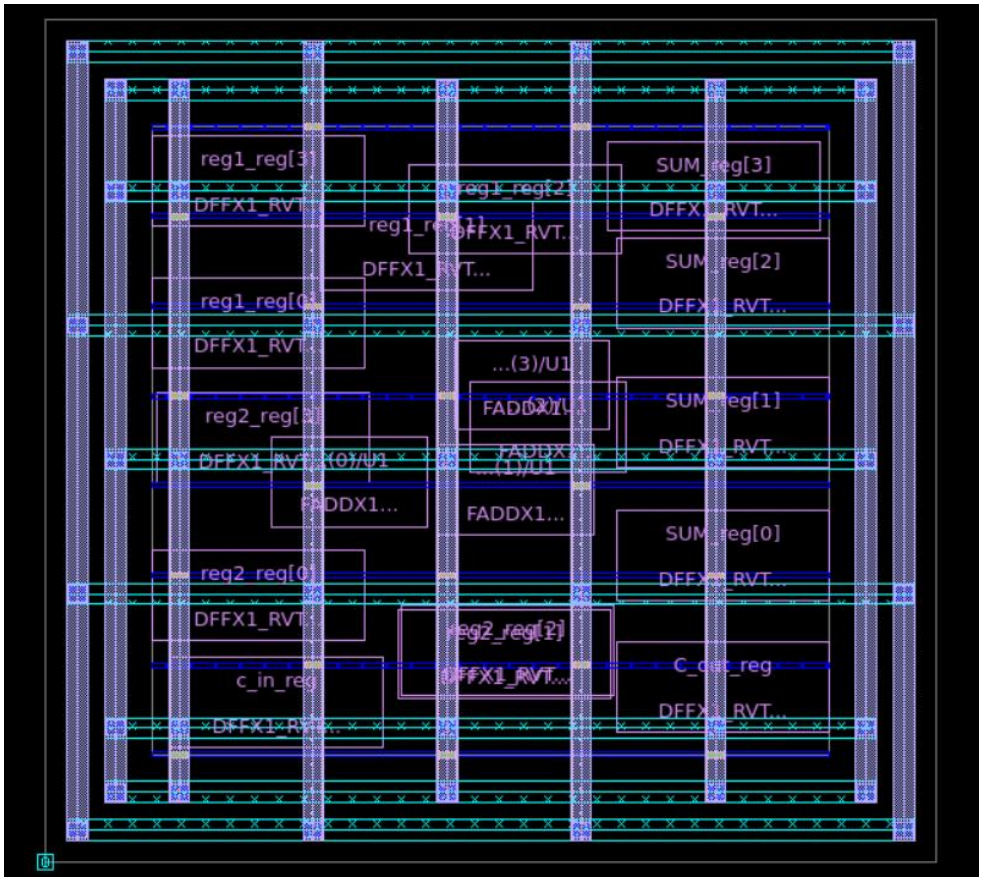
To place all standard cells of the 4-bit Full Adder logically and efficiently within the defined floorplan area.

5. Further, run the command: source home/RTL2GDSII_TESTCASE/ICCII/scripts/placement.tcl and observe the changes in the ICCII Shell.
    a. We need to make some changes in the placement.tcl file
        - Source ./../CONSTRAINTS/full_adder.sdc
    b. Perform the following commands again: report_qor, report_timing, report_power, report_units.
    c. Also do the port-pin verification.



Floorplanning Placement:

create_floorplan -placement



Now close the block and lib

close_block

close_lib

Clock Tree Synthesis

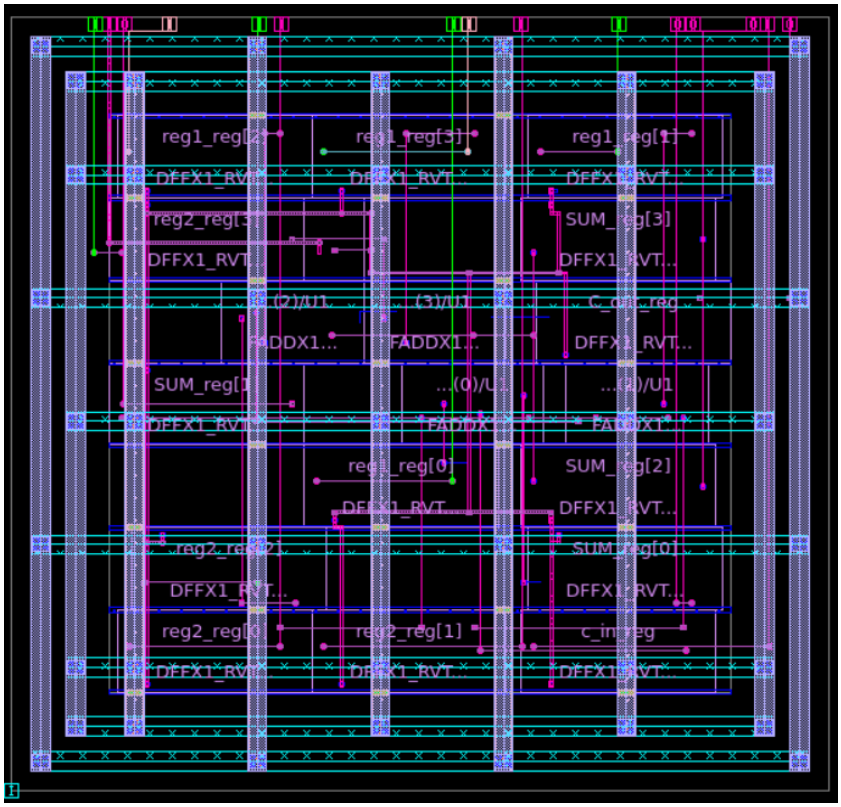
To distribute the clock signal uniformly to all sequential elements (flip-flops) with minimal skew and latency.

6. Now we move on to sourcing the clock.tcl file using the command: source home/RTL2GDSII_TESTCASE/ICCII/scripts/clock.tcl. Also perform the following commands again and observe the changes: report_qor, report_timing, report_power, report_units.

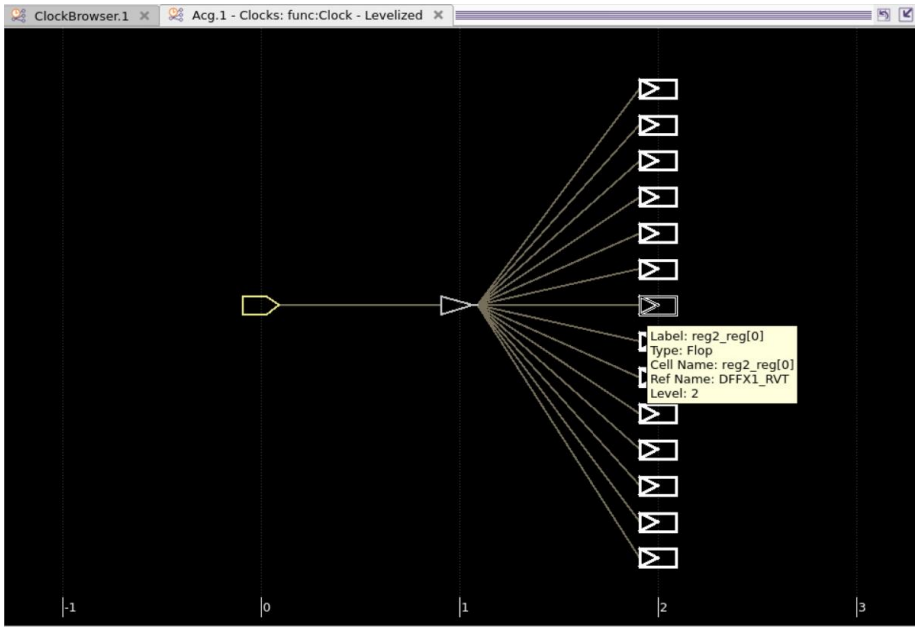
Run the following commands for synthesizing clock tree

```
################################################
##        CTS using CCD  commands
################################################
##
##stage 1:
##

synthesize_clock_tree
##stage 2:
#
set_app_options -name cts.optimize.enable_local_skew -value true
set_app_options -name cts.compile.enable_local_skew -value true
set_app_options -name cts.compile.enable_global_route -value false
set_app_options -name clock_opt.flow.enable_ccd -value true

#stage 3:
###
#
clock_opt

#########################################################
###  to report qor the respective CTS
#########################################################
#
#report_clock_qor

#########################################################
#  to save the block
#########################################################

save_block -as full_adder_cts_CCD
save_lib
```

GUI:



Clock Buffers



Now, report_cells

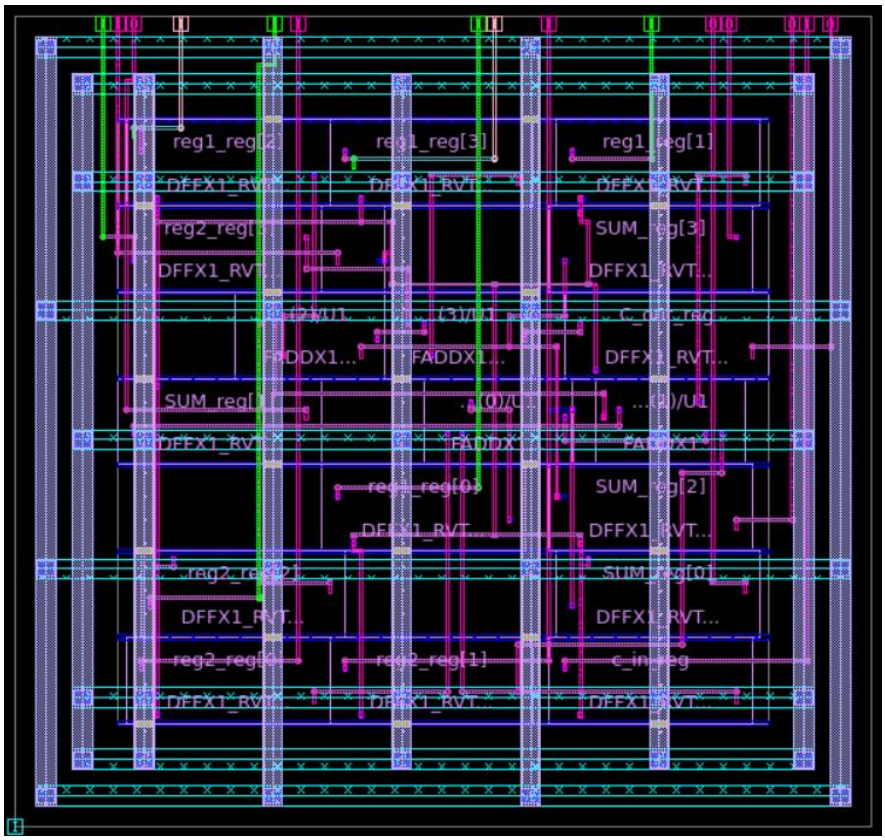
```
Cell                      Reference       Library          Attributes
--------------------------------------------------------------------------------
C_out_reg                 DFFX1_RVT       saed32rvt_c      n, S
SUM_reg[0]                DFFX1_RVT       saed32rvt_c      n, S
SUM_reg[1]                DFFX1_RVT       saed32rvt_c      n, S
SUM_reg[2]                DFFX1_RVT       saed32rvt_c      n, S
SUM_reg[3]                DFFX1_RVT       saed32rvt_c      n, S
add_1_root_add_33_2/L1(0)/U1
                          FADDX1_RVT      saed32rvt_c
add_1_root_add_33_2/L1(1)/U1
                          FADDX1_RVT      saed32rvt_c
add_1_root_add_33_2/L1(2)/U1
                          FADDX1_RVT      saed32rvt_c
add_1_root_add_33_2/L1(3)/U1
                          FADDX1_RVT      saed32rvt_c
c_in_reg                  DFFX1_RVT       saed32rvt_c      n, S
ctosc_drc_inst_211        NBUFFX2_RVT     saed32rvt_c
reg1_reg[0]               DFFX1_RVT       saed32rvt_c      n, S
reg1_reg[1]               DFFX1_RVT       saed32rvt_c      n, S
reg1_reg[2]               DFFX1_RVT       saed32rvt_c      n, S
reg1_reg[3]               DFFX1_RVT       saed32rvt_c      n, S
reg2_reg[0]               DFFX1_RVT       saed32rvt_c      n, S
reg2_reg[1]               DFFX1_RVT       saed32rvt_c      n, S
reg2_reg[2]               DFFX1_RVT       saed32rvt_c      n, S
reg2_reg[3]               DFFX1_RVT       saed32rvt_c      n, S
--------------------------------------------------------------------------------
Total 19 cells

1
icc2_shell>
```

One extra cell is seen as clock tree has been synthesized

Routing

To connect all placed standard cells, macros, and I/O pins using optimized metal interconnects while meeting timing and DRC rules.

7. The last step in ICCII Shell is to do the routing using the route.tcl file. Source the file using the command: source home/RTL2GDSII_TESTCASE/ICCII/scripts/route.tcl.



PrimeTime

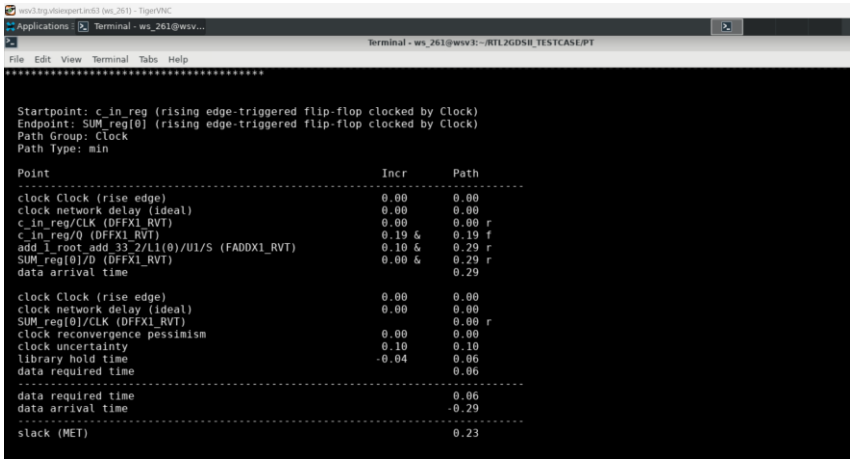
To perform timing signoff of the 4-bit Full Adder design and ensure all setup and hold constraints are met after routing.

8. Run the following commands:

```
set link_path "./../ref/lib/stdcell_rvt/saed32rvt_tt0p78vn40c.db"
read_verilog "./../ICCII/results/full_adder.routed.v"
read_sdc ./../CONSTRAINTS/full_adder.sdc
read_parasitics "./../ICCII/results/full_adder_func::nom.spef.p1_125.spef"
update_timing -full
report_timing
report_timing -delay_type min
report_global_timing
```

Global Timing and Slack:

```
pt_shell> report_global_timing
****************************************
Report : global_timing
        -format { narrow }
Design : full_adder
Version: W-2024.09-SP1
Date   : Mon Oct 20 00:24:05 2025
****************************************

No setup violations found.

No hold violations found.
1
```



GDSII Generation (Final Layout Output)

To generate the final GDSII file of the 4-bit Full Adder design for fabrication after completing all physical and signoff checks.

Result:

- Successfully generated **final GDSII** file ready for tapeout.
- The design represents the complete **RTL to GDS flow** implementation.

9. Use command: ‘write_gds full_adder.gds’ in icc2_shell

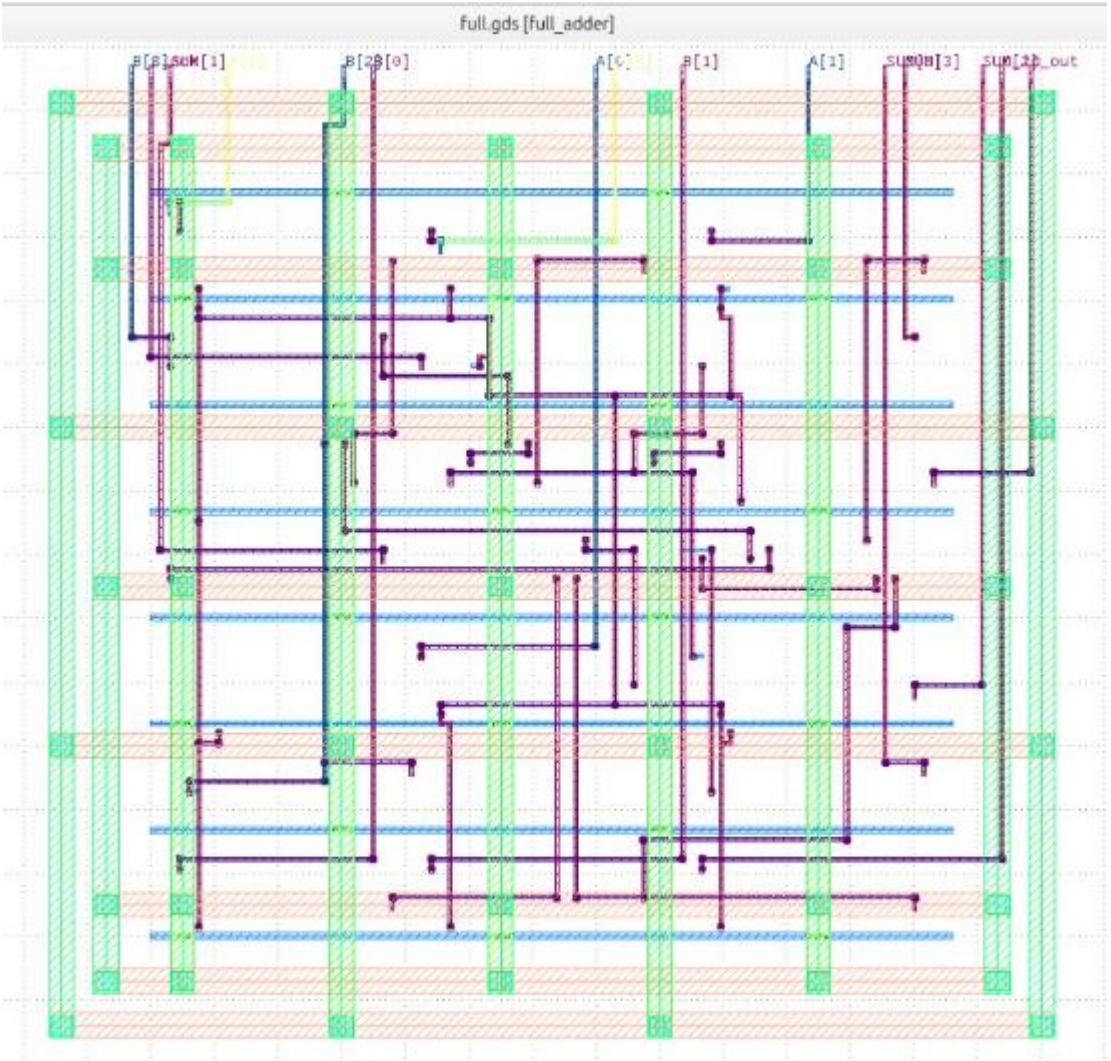


Figure. Final GDSII layout view of the 4-bit Full Adder in Synopsys IC Compiler II.

Key Learnings & Conclusion:

Through this workshop, I gained hands-on experience with the complete **RTL to GDSII flow** using Synopsys tools. I learned how to design, simulate, synthesize, and implement a **4-bit Full Adder** from Verilog RTL to final GDSII layout. The practical use of tools like **VCS**, **Verdi**, **Design Compiler**, **IC Compiler II**, and **PrimeTime** helped me understand every stage - from logic design to physical realization. This experience strengthened my understanding of the **VLSI design process**, timing analysis, and layout verification, providing a strong foundation for future chip design projects.