Design of a 4-bit Simple as Possible (SAP) Computer

Bangladesh University of Engineering and Technology

Course No: EEE 458
Course Name: VLSI Circuits and Design II Laboratory

Report Title

Design of a 4-bit Simple as Possible (SAP) Computer

Submitted to

Dr. A.B.M. Harun-Ur-Rashid	**Rifat Shahriar**
Professor, Department of Electrical and Electronic Engineering, Bangladesh University of Engineering and Technology	Lecturer, Department of Electrical and Electronic Engineering, Bangladesh University of Engineering and Technology

Submitted by

Ramit Dutta	1606006	Department of EEE Group: G1 Level :4 Term: II
Shafin Bin Hamid	1606008	

EDA Playground Link : https://www.edaplayground.com/x/W42N

Table of Contents

List of Figures

List of Tables

1. Introduction

The goal of this project is to

1. Design a 4-bit simple as possible (SAP) computer using VerilogHDL
2. Write appropriate testbenches to verify functionality of the made design
3. Perform RTL logic synthesis using Cadence Genus
4. Use Cadence Innovus for physical design
5. Investigate any errors faced during physical design
6. Perform a trial-and-error check to reduce number of design errors to zero

The rest of the report is organized as follows:

In section 2, we will gradually explain our design architecture. In section 3, we will show the necessary results and waveforms to demonstrate the functional verification of our design using Cadence NC-Sim and SimVision. In section 4, we will present the schematic, power report and timing report using Cadence Genus. In section 5, we will take a progressive approach and detail each part of the complete physical design process. In section 6, we will explore the design rule check (DRC) errors faced at the end of physical design and our attempts to reduce the number of errors to zero.

2. Design Route

2.1. Instruction Set

The instruction set contains 5 instructions in total. Each of them is explained below with a short description:

Table 1: Instruction Set

Instruction	Operation	Opcode
LDA [Address]	Load the accumulator with the content located at the specified address of RAM	0000
ADD [Address]	Add to the accumulator the content located at the specified address of RAM	0001
SUB [Address]	Subtract from the accumulator the content located at the specified address of RAM	0010
OUT	Display the content of the output register	1110
HLT	Halt program execution (stops the program counter from incrementing)	1111

2.2. Architecture

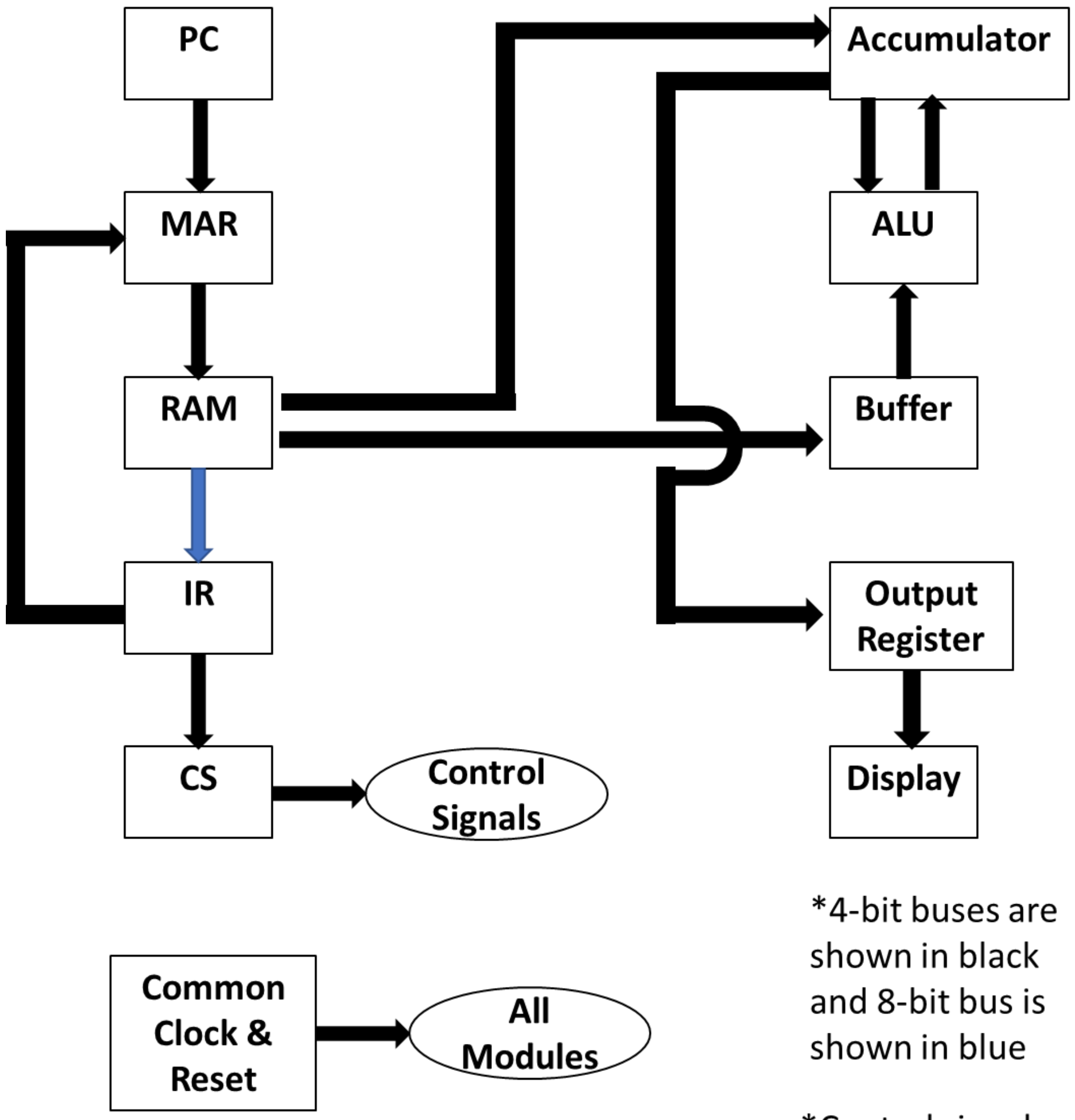
Initially, we have tried to closely follow the SAP-1(Simple as Possible) architecture as our design choice with some modifications. In Figure 1, we have drawn the architecture showing the connections between the different modules used in the construction of our 4-bit Computer. For better understanding of the connections, we have deliberately omitted the control signals from the figure.

From the figure, it is seen that there are 10 different modules in the computer. The SAP diagrams that we are familiar with usually have a common 'Write Bus' for establishing communications between the different modules. But instead of using a common bus for communicating between the different modules, we have used separate buses for all of them. This is the unique feature of our computer.

The reason to do so is twofold. For starters, we thought it is simpler to write Verilog code for such an architecture since we can be certain that on a clock edge, there will be no unwanted overwrites or loss of information in the Bus. This also makes data communication faster because the bus doesn't become a bottleneck at any given time.

Because of using a separate bus for all communication nodes, the amount of control signals used in this architecture is also very high. For example, in the usual SAP-1 architecture, there is only 1 control signal 'Ce' that is used to signal that RAM is being accessed and on next clock edge the addressed data will be put on the 'Common Write Bus'. But, in our architecture, since we are using separate buses to send data from RAM to IR and RAM to A, we are having to use two separate control signals 'Cei' and 'Cea', respectively indicating IR and A as the destination of the content at the addressed RAM location.

Figure 1 can be taken as a steppingstone for what is to follow. We will gradually discuss the functionality of each module and the purpose of the control signals sent to each of them.



*4-bit buses are shown in black and 8-bit bus is shown in blue

*Control signal and manual input to RAM have not been shown for better understanding of the connections

Figure 1: SAP Full Architecture

2.3. Construction of each module and their functionality

2.3.1 Program Counter (PC)

Functionality:

The program counter holds the current 4-bit address of the instruction to be executed. During each machine cycle, the program counter sends this address to the MAR and then incremented by 1. The connections of the Program Counter module is shown in Figure 2.

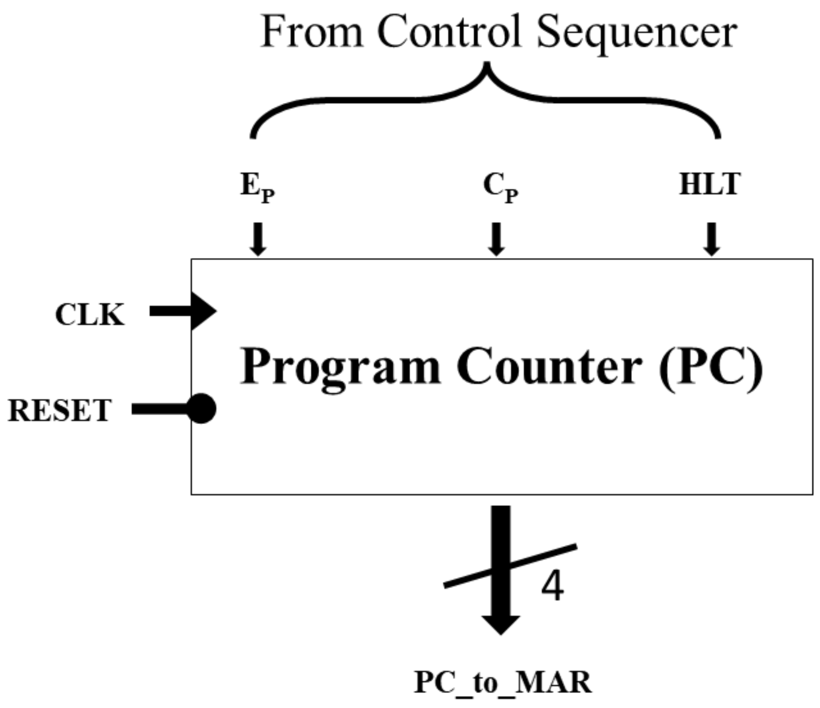


Figure 2: Program Counter (PC)

Inputs:

- Clk and Reset: The common clock and common reset. At the negative edge of reset, the Program Counter is reset to zero.
- E_p: When E_p is set, the current value of the program counter is latched onto output PC_to_MAR and sent to MAR on the next positive clock edge.

- C_p: When C_p is set, the current value of the program counter is incremented by 1 on next positive clock edge. If the current value is 4’b1111, then it is set to 4’b0000.
- HLT: Indicating current instruction is a HLT instruction. When this is set, the program counter first decrements by 1 on next positive clock edge. This happens so because by the time it is realized that the current instruction is a HLT, the value of PC has already incremented by 1. So, the decrementing operation is done to make PC point again to HLT. The program counter therefore keeps fetching and executing the same HLT instruction forever.

Output:

- PC_to_MAR: 4-bit address that indicates the location of the current instruction in the RAM.

2.3.2 Memory Address Register (MAR)

Functionality:

The Memory Address Register is a 4-bit register that stores the address location of from the Program Counter during each machine cycle. This is typical to a MAR we see in SAP-1. The connections of the Memory Address Register module are shown in Figure 3.

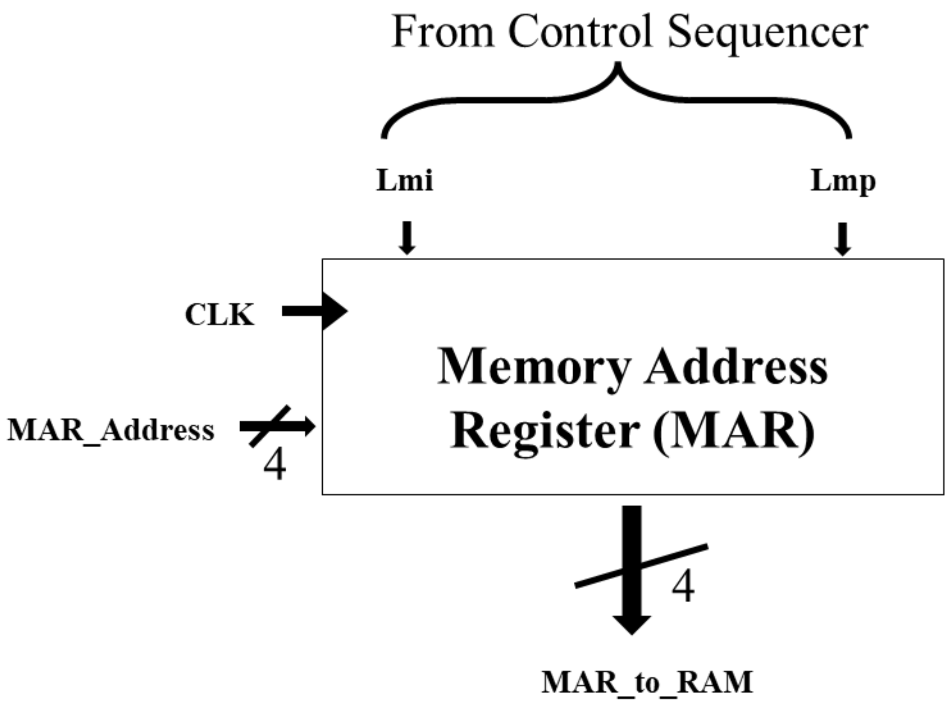


Figure 3: Memory Address Register (MAR)

Inputs:

- Clk and reset: The common clock and common reset. At the negative edge of reset, the Memory Address Register is reset to zero.
- L_{mp}: When L_{mp} is set, the input address location coming from Program Counter (PC_to_MAR) is latched onto MAR and put on the bus to RAM.
- L_{mi}: When L_{mi} is set, the address field/last 4 bits of the Instruction Register (IR_operand) gets loaded into the MAR. An example of this

is when there is a Load instruction. So, the last 4 bits of IR indicates where to look for the data in the RAM that is to be loaded into A.

- MAR_Address: Output of Program Counter or Instruction Register. Used to address into RAM.

Output:

- MAR_to_RAM: The 4-bit address location used to address into RAM. This can come from PC or IR depending on the state of the current instruction.

2.3.3 Random Access Memory (RAM)

Functionality:

The Random-Access Memory (RAM) is used to store both data and instruction in the same 2D structure. We have used a 16*8 RAM, meaning we have in total 16 address locations and each of them is size 1 byte.

Among these 16 locations, the first 9 are used for storing instructions and the next 7 are for storing data. This can be visualized by Figure 4.

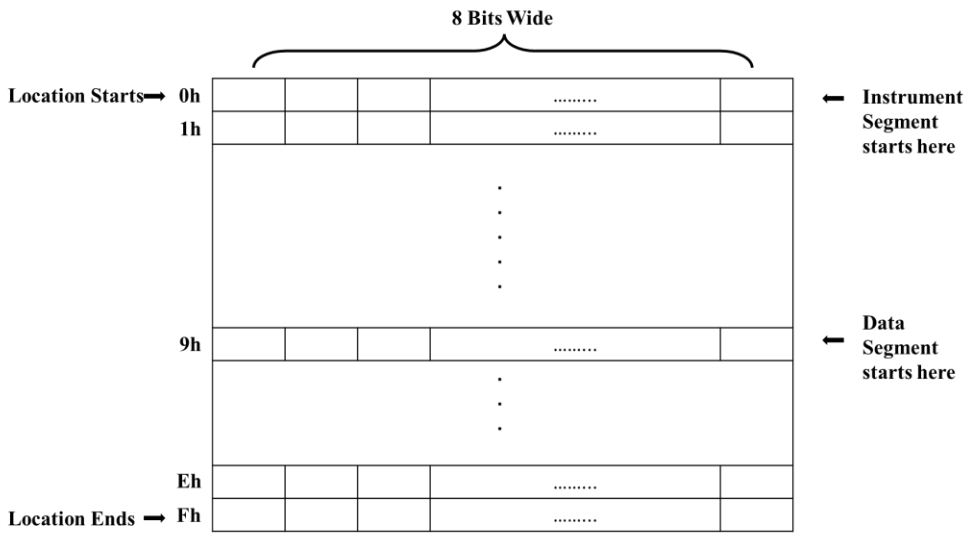


Figure 4: Inside RAM

The connections of the Random-Access Memory module is shown in Figure 5.

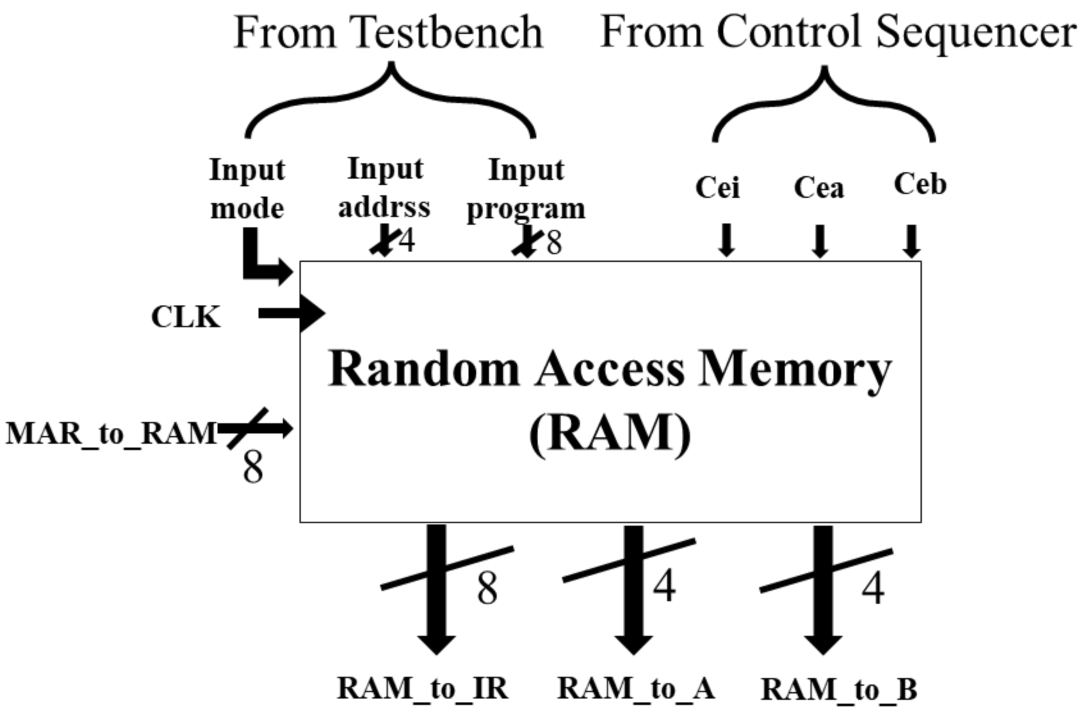


Figure 5: Random Access Memory (RAM)

Inputs:

- Clk: Common Clock
- Input mode: Used to indicate if the computer is ready to take manual input from user
- Input address: When under manual mode, this is a 4-bit input to indicate where to save the instruction/data input
- Input program: When under manual mode, this is a 8-bit input to indicate the instruction/data input to be saved
- MAR_to_RAM: Denotes the current location to be addressed. Same as before.
- C_{ei}: When C_{ei} is set, on the next positive clock edge, the content at the addressed location is put on the bus RAM_to_IR. This is for sending a typical 8-bit instruction to the Instruction Register.

- C_{ea}: When C_{ea} is set, on the next positive clock edge, the content at the addressed location is put on the bus RAM_to_A. This is for 'LDA [Address]' instruction, where the data at the 'Address' is put on the bus connecting RAM to A.
- C_{eb}: When C_{eb} is set, on the next positive clock edge, the content at the addressed location is put on the bus RAM_to_B to send to the Buffer Register (B). This is used for ADD/SUB [Address] instruction to save the value at the RAM [Address] to the buffer.

Output:

- RAM_to_IR: 8-bit output that carries the instruction from RAM to IR.
- RAM_to_A: 4-bit data bus connecting RAM to A used for 'LDA [Address]' instruction.
- RAM_to_B: 4-bit data bus connecting RAM to B used for 'ADD/SUB [Address]' instruction.

2.3.4 Instruction Register (IR)

Functionality:

The Instruction Register (IR) stores the current instruction to be executed, splits it into two equal parts and sends the most significant 4-bits, or the opcode, to the control sequencer (CS). The least significant 4 bits, or the operand, are sent to the PC (for 'JZ Address') or MAR (for 'MOV') depending on the type of instruction. The connection of the Instruction Register module is shown in Figure 6: Instruction RegisterFigure 6.

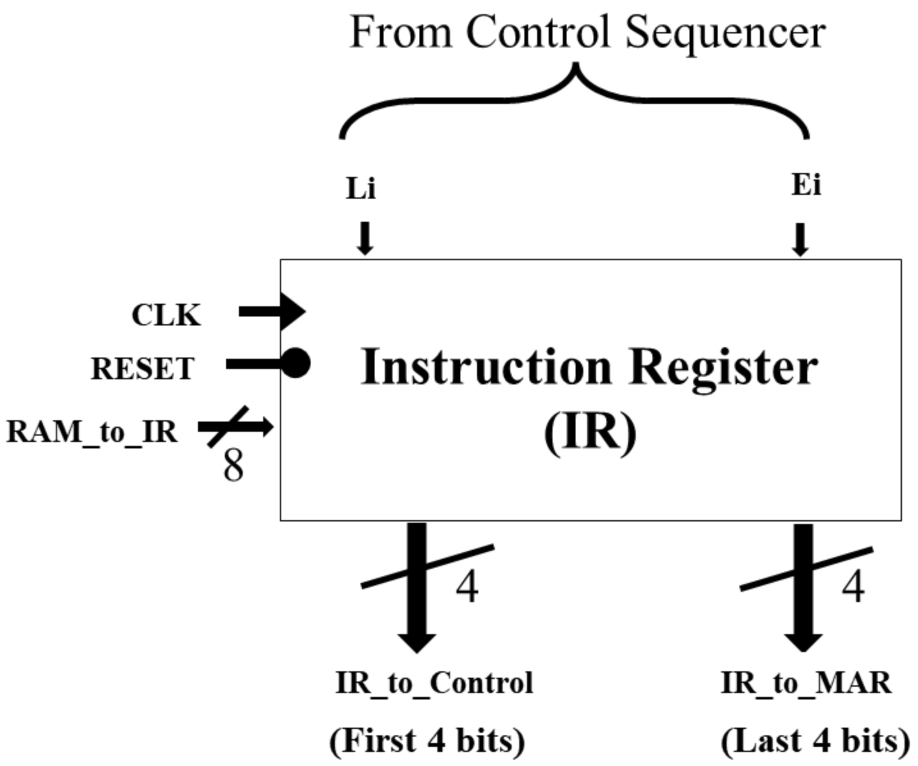


Figure 6: Instruction Register

Inputs:

- Clk and Reset: The common clock and common reset. At the negative edge of reset, the Instruction Register is reset to high impedance, so it does not produce any invalid control signals by passing 4’b0000 to Control Sequencer.
- L_i: For each instruction, when L_i is set, the instruction is loaded into the instruction register.
- E_i: For any instruction that needs memory referencing, e.g. LDA [Address], this signal is set. And, then the last 4-bits of the current instruction in IR is put on the bus to MAR/PC depending on the type of instruction.
- RAM_to_IR: 8-bit instruction fed to IR by RAM

Outputs:

- IR_to_Control: The first 4-bits or, the opcode of the instruction is sent to the Control Sequencer to generate control signals and these bits are put on the bus IR_to_Control
- IR_operand: The last 4-bits, or the operand of the instruction is sent to MAR/PC and put on the bus IR_operand

2.3.5 Accumulator Register

The accumulator is a 4-bit register. It is used for the following purposes:

a) To get data from the RAM
b) To store the results from ADD/SUB instructions
c) To display data using OUT

We will understand more about its functionalities through the simulation of the instruction set later. The connections of the Accumulator module are shown in Figure 7.

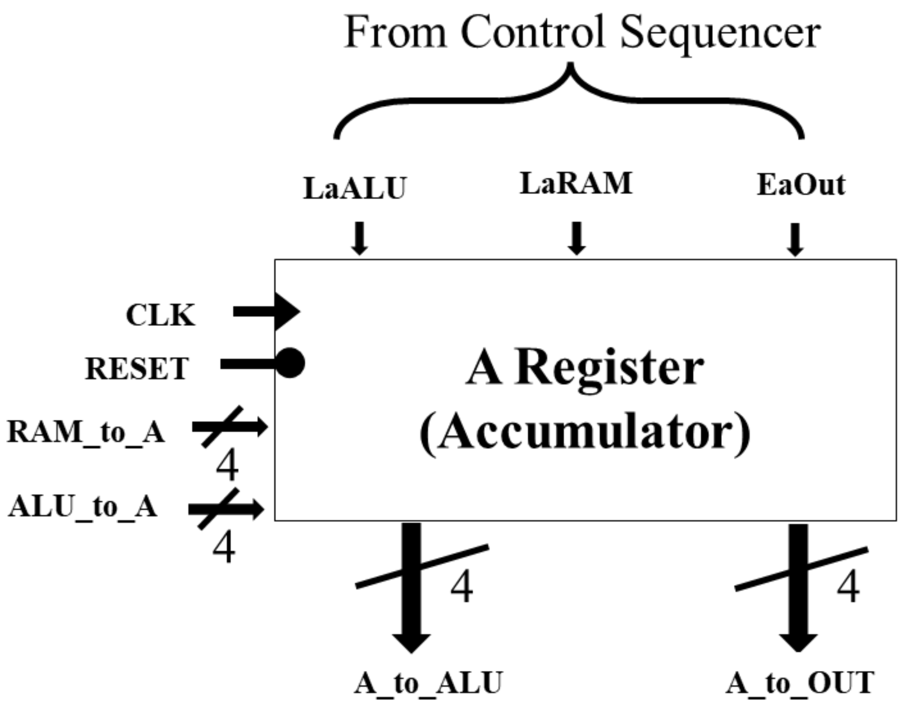


Figure 7: Accumulator Register

<u>Inputs:</u>

- Clk and Reset: Common clock and reset
- RAM_to_A: 4-bit data bus carrying the data from the RAM. Same as before.
- ALU_to_A: 4-bit data bus carrying the result from the ALU. Always set to the ALU result.
- L_{aALU}: For any instruction that performs ALU operation and stores result into A, e.g. ADD, SUB this signal is set. The data coming from the ALU is latched onto the Accumulator register.
- L_{aRam}: For 'LDA [Address]' instruction, when L_{aRam} is set, the data coming from the RAM is latched onto the Accumulator register.
- E_{aOut}: Set for 'OUT' instruction. The current value stored in A is put on the bus to Output Register (A_to_OUT).

<u>Outputs</u>:

- A_to_ALU: 4-bit data bus connecting A to ALU. Always holds the current value in A and continuously feeds the ALU.
- A_to_OUT: 4-bit data bus connecting A to Output Register (TMP). Carries data from A to OUT. Used for 'OUT'

2.3.6 Buffer Register

Functionality:

The Buffer Register is a 4-bit register. It is used to hold the data sent from RAM for ADD/SUB [Address] instructions. The connections of the Buffer Register module are shown in **Error! Reference source not found.**.

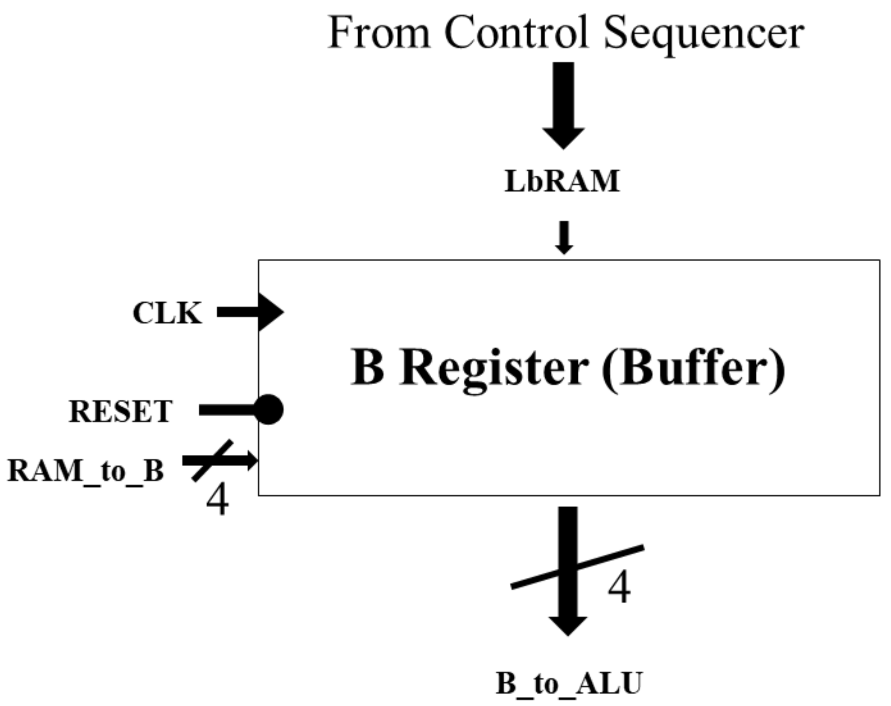


Figure 8: Buffer Register

Inputs:

- Clk and Reset: Common clock and reset
- RAM_to_B: 4-bit data bus carrying the data from the RAM.

Outputs:

- B_to_ALU: 4-bit data bus connecting B to ALU. Always holds the current value in B and continuously feeds the ALU.

2.3.7 Arithmetic Logic Unit (ALU)

Functionality:

The ALU performs all arithmetic instructions in the computer. All these instructions are synchronous, that is, they are performed on the positive edge of the clock. It must be mentioned here that an adder-subtractor module has been written separately to handle the ADD/SUB operations. This module is reported in Figure 9. It is the same as the adder-subtractor module typically used in a digital electronics circuit.

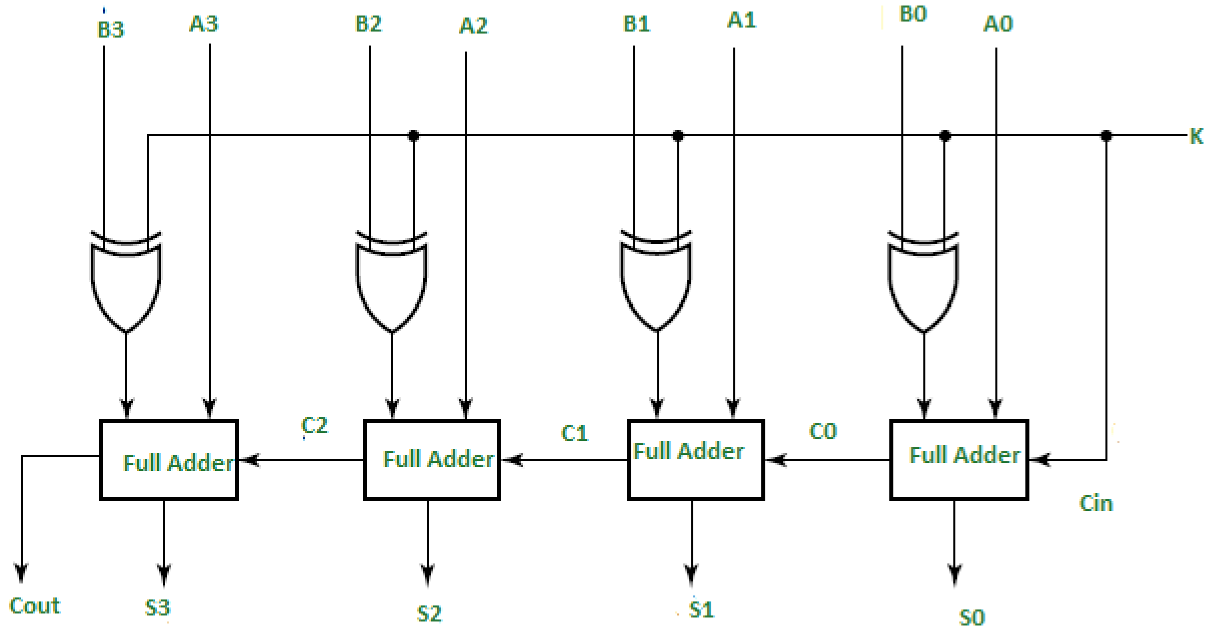


Figure 9: Adder Subtractor of ALU

The connections of the Arithmetic Logic Unit are shown in Figure 10.

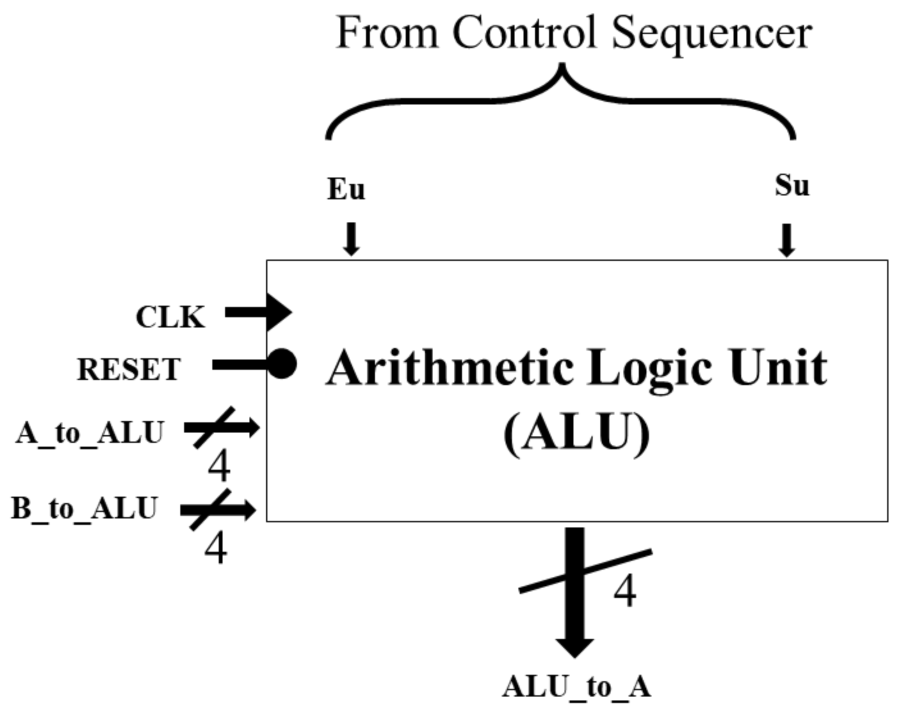


Figure 10: Arithmetic Logic Unit

2.3.8 Output Register

<u>Functionality</u>:

This register is used for ‘OUT’ operation. It takes the data from A and sends it through the output port for display purposes. The connections of the Output Register module are shown in Figure 11.

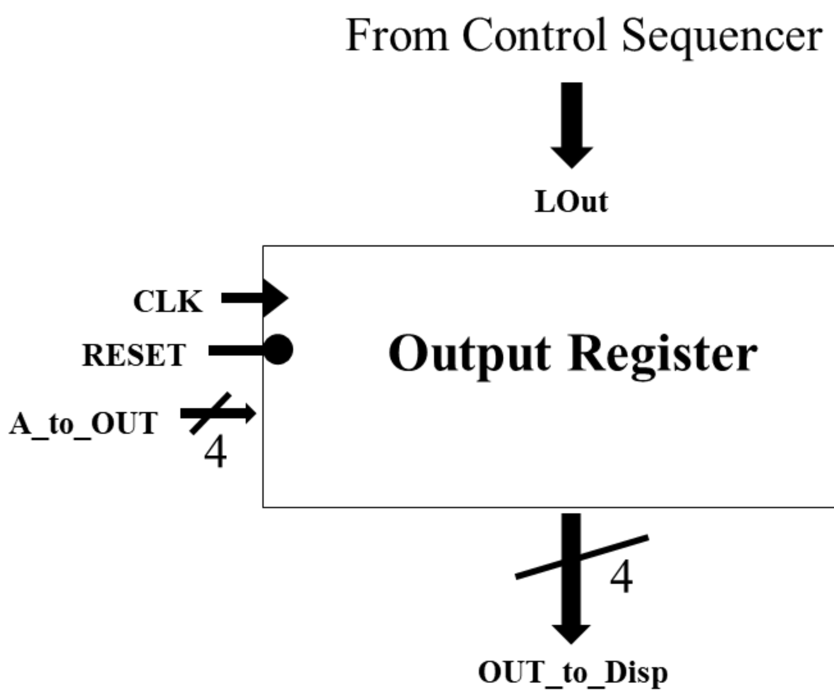


Figure 11: Output Register

<u>Inputs</u>:

- Clk and reset: The common clock and common reset.
- L_{aOut}: For 'OUT' operation, this control signal is set and data coming from A is loaded onto the output register.
- A_to_Out: 4-bit data bus carrying the data stored in A to Output Register

<u>Outputs</u>:

- OUT_to_disp: 4-bit data bus carrying the data stored in Output Register to display for 'OUT' operation.

2.3.9 Control Sequencer (CS)

<u>Functionality</u>:

This is the brain of the computer. All control signals are generated here and sent to the different modules. The process of generating control signals is explained below:

a) From the common clock, a ring counter is used inside this module to generate the six states for a machine cycle. The states have a general form such as the following: T_6 T_5 T_4 T_3 T_2 T_1, where only one of these is a '1' during a given state. So, for state 1, we have '000001'.
b) Depending on the current state and the type of instruction decoded using the opcode, control signals are set.
c) Once the signals are set, internally the state is changed to the next state by shifting the bit pattern one bit to the left. From '000001', we go to '000010' next and then to '000100' and so on.
d) When the state is '100000', the state resets to '000001'
e) When there is manual reset, the state within the module changes to '000001'.

The ring counter output looks like Figure 12. For the first 3 states, the generated control signals are the same. For the following three states, the control signals change based on the opcode/type of instruction.

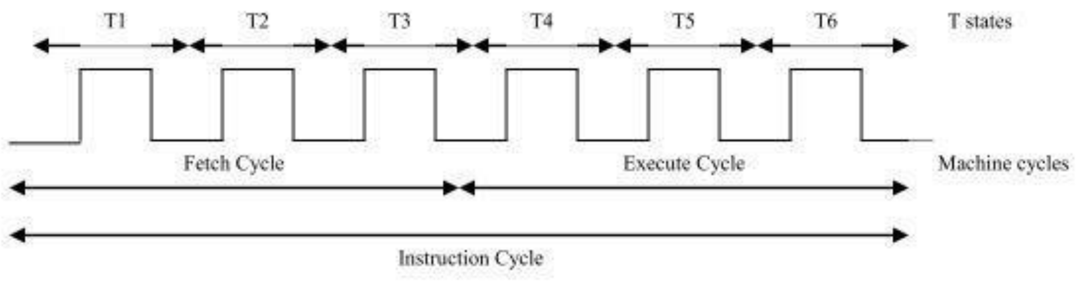


Figure 12: Output of Ring Counter used for different timing states

The connections of the Control Sequencer are shown in Figure 13.

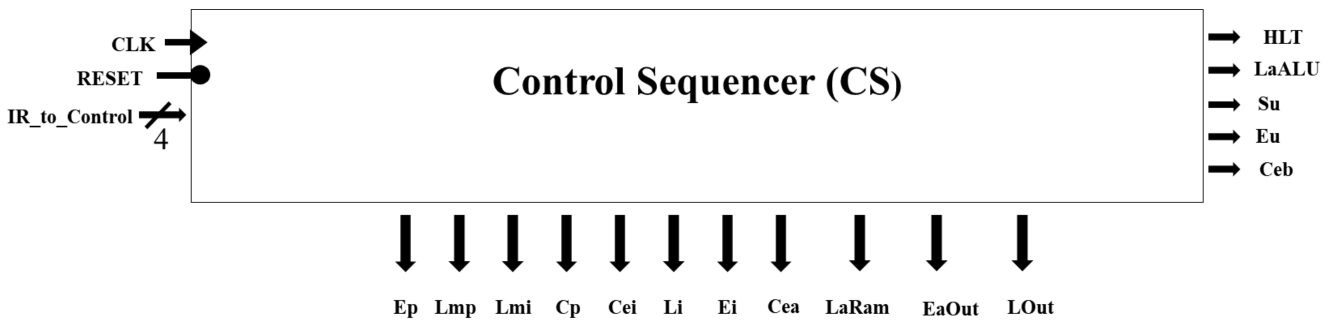


Figure 13: Control Sequencer

<u>Inputs</u>:

- Clk and reset: The common clock and common reset. At the negative edge of reset, the all the control flags are set to zero.
- IR_to_Control: The first 4-bits/opcode coming from the Instruction Register (IR). This denotes the form of instruction that is to be executed next

<u>Outputs</u>:

All control signals are generated by the control sequencer. There are in total **16** of them, and all of them have been discussed in the description of their respective destination modules.

3. Functional Verification of the Design

3.1. EDA Playground

In this section, the waveforms of the different instruction sets will be presented. For each instruction, a testbench has been written to fully visualize everything that is happening while that instruction is being executed.

Here we will mainly focus on the significant information such as which control signal is being turned on, and how the different values are changing.

There are six machine states that form a machine cycle. It is also equal to the instruction cycle, so the time to fetch and execute each instruction is kept the same. For any instruction, in the first three states, the control signals generated are the same. They are presented below in Table 2:

Table 2: Operation in first three timing states

Timing State	**Control Signals Set**	**Purpose**
$\mathbf{T_1}$	E_p, L_{mp}	Send the address of the current instruction to the MAR
$\mathbf{T_2}$	C_p (except for HLT in which case this is off)	Increment the program counter by 1
$\mathbf{T_3}$	C_{ei}, L_i	Send the instruction at the addressed location to the instruction register and generate control signals in CS

<u>Testbench:</u>

We will demonstrate the functional verification using only one testbench that contains all types of instructions. The assembly code is presented below:

```
LDA 9h;
OUT;
ADD Ah;
OUT;
SUB Bh;
OUT;
HALT;
ADD Ah;
OUT;
```

In Ram, at location 9h we have decimal 11, at location Ah, we have decimal 1, at location Bh, we have decimal 3. We will sequentially present the results of the instructions as they appear in the testbench. The waveforms will be followed by a table that contains the operations performed during each timing state.

3.3.1 LDA [Address]

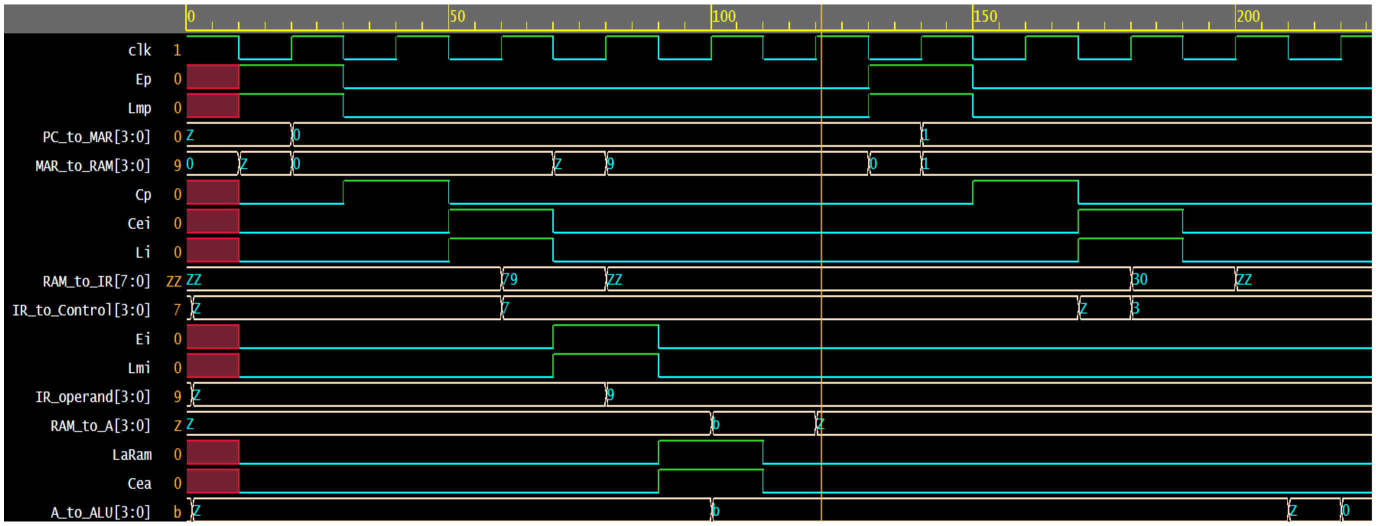


Figure 14: LDA Verification using EDA Tools

Table 3: Operation in final three timing states for LDA

Timing State	Control Signals Set	Purpose
T_4	E_i, L_{mi}	Send the address location of the data that is to be loaded into A to MAR
T_5	C_{ea}, L_{aRam}	Send the data from the addressed location at RAM and load into A
T_6	None	NOP

3.3.2 OUT

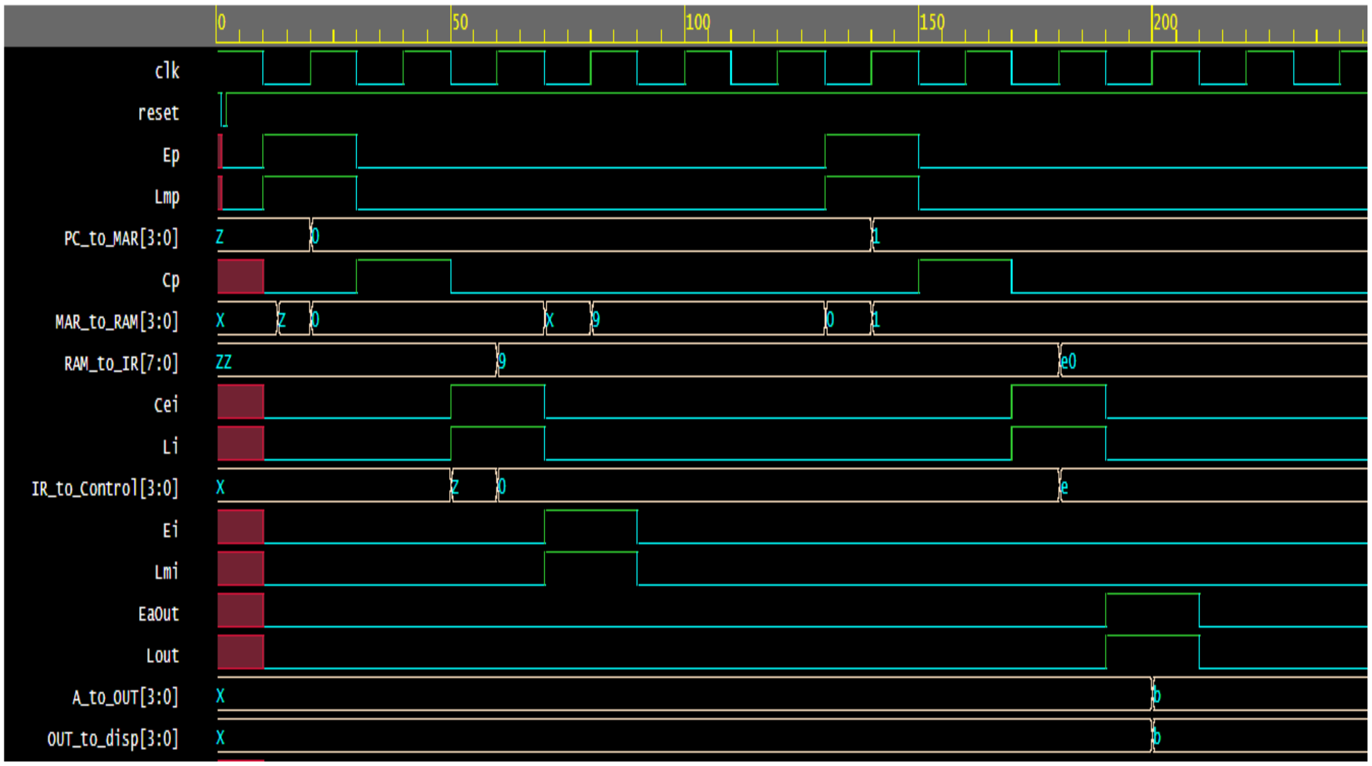


Figure 15: OUT Verification using EDA Tools

Table 4: Operation in final three timing states for OUT

Timing State	**Control Signals Set**	**Purpose**
T_4	E_{aOut}, L_{Out}	Send the data in A and load output register with that data
T_5	None	NOP
T_6	None	NOP

3.3.3 ADD [Address]

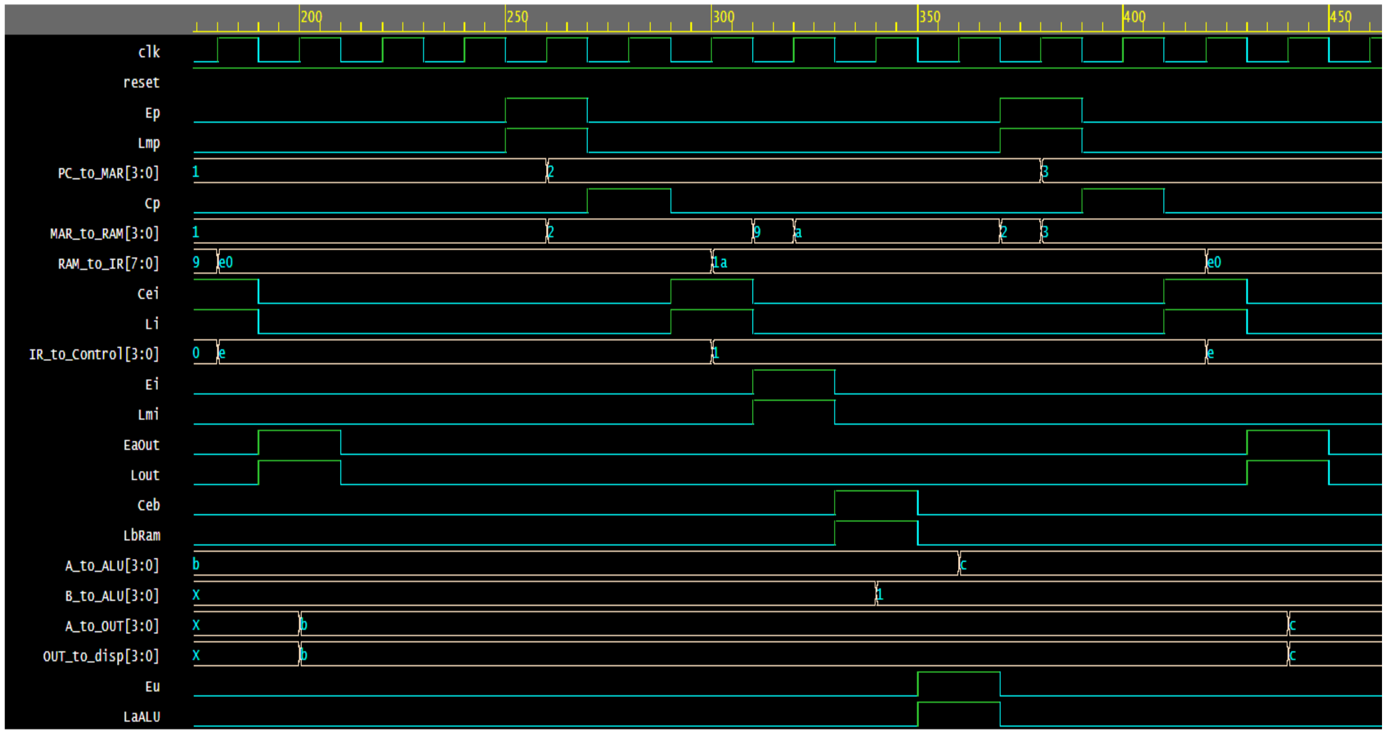


Figure 16: ADD Verification using EDA Tools

Table 5: Operation in final three timing states for ADD

Timing State	Control Signals Set	Purpose
T_4	E_i, L_{mi}	Send the address location of the data that is to be loaded into A to MAR
T_5	C_{eb}, L_{bRam}	Send the data from the addressed location at RAM and load into B
T_6	E_u, L_{aALU}	Performs addition operation (A+B) at ALU and stores result at A

3.3.4 SUB [Address]

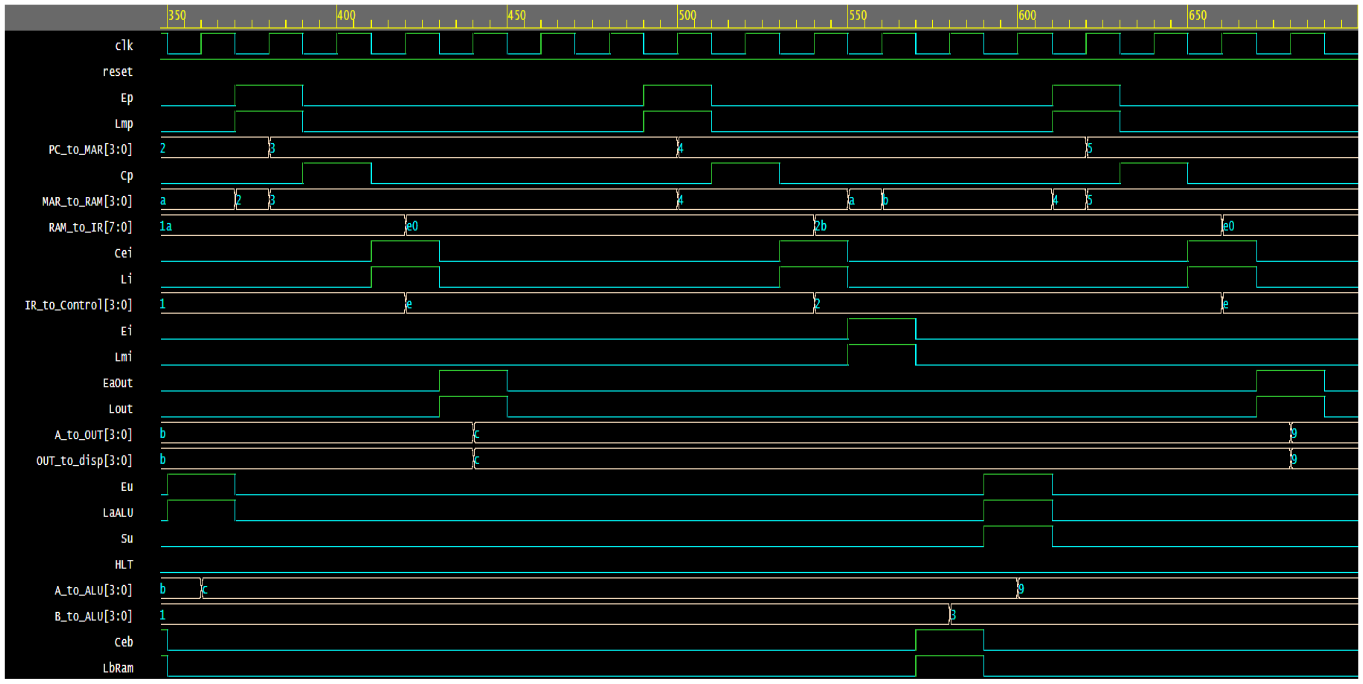


Figure 17: SUB Verification using EDA Tools

Table 6: Operation in final three timing states for SUB

Timing State	**Control Signals Set**	**Purpose**
T_4	E_i, L_{mi}	Send the address location of the data that is to be loaded into A to MAR
T_5	C_{eb}, L_{bRam}	Send the data from the addressed location at RAM and load into B
T_6	E_u, S_u, L_{aALU}	Performs subtraction operation (A-B) at ALU and stores result at A

3.3.5 HALT

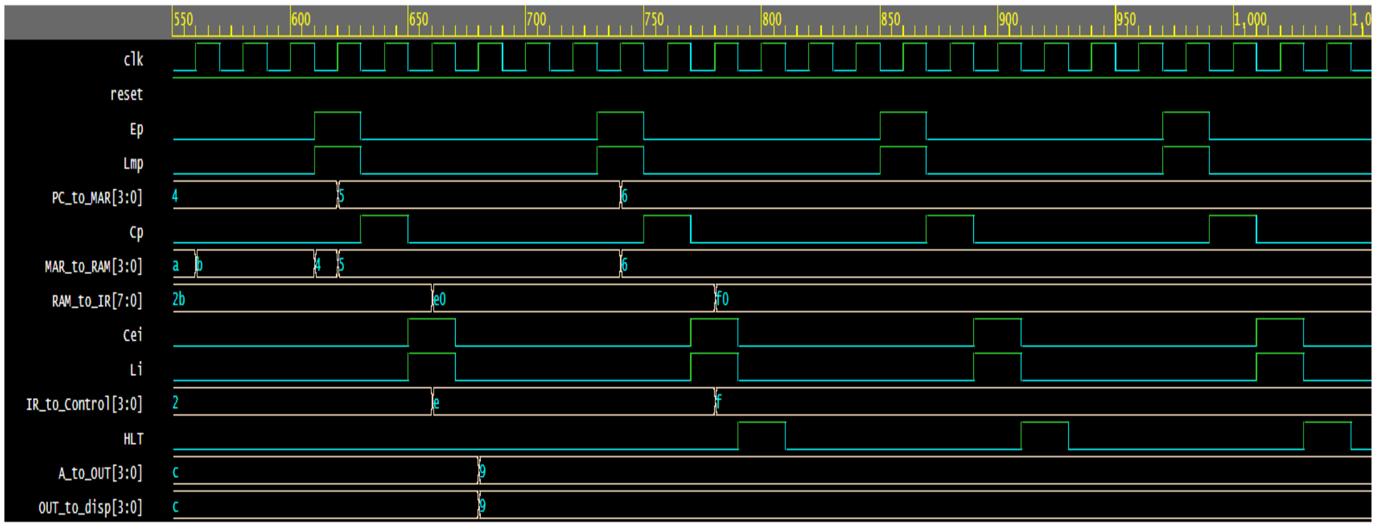


Figure 18: HALT Verification using EDA Tools

Table 7: Operation in final three timing states for HALT

Timing State	**Control Signals Set**	**Purpose**
T_4	HLT	The program counter is decremented by 1
T_5	None	NOP
T_6	None	NOP

3.2. Challenges of multiple modules

Due to the usage of multiple modules, our design could not be synthesized all at once. We can synthesize each module separately, but if we try to synthesize using multiple modules, then the netlist file remains blank. To solve this situation, we converted the code into a single module.

The inputs are clock, reset, input mode, input addresses and input instructions. The output is a 4-bit data output that is to be displayed from the output register. From here on, all results will be discussed based on aforementioned inputs and outputs.

3.3. Simulation using Cadence NC-Sim and Sim Vision

3.3.1. Inputs

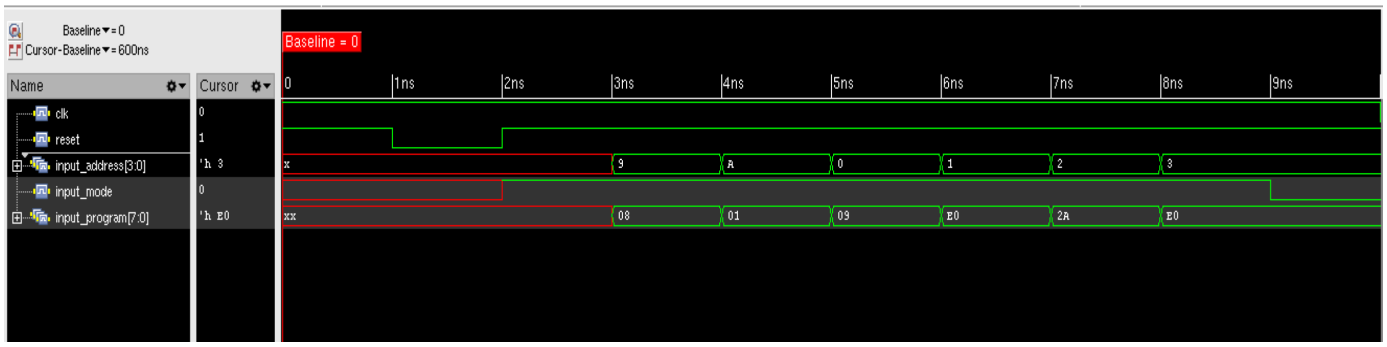


Figure 19: Inputs of testbench in SimVision

Before the computer starts executing instructions, the data and instructions are all loaded together into the RAM. We can see from Figure 19 that at location 9, decimal 8 is loaded and at location 10, decimal 1 is loaded. The first instruction is an LDA 9h instruction followed by OUT. Then, there is an ADD Ah instruction followed by another OUT.

3.3.2. LDA+OUT Testbench

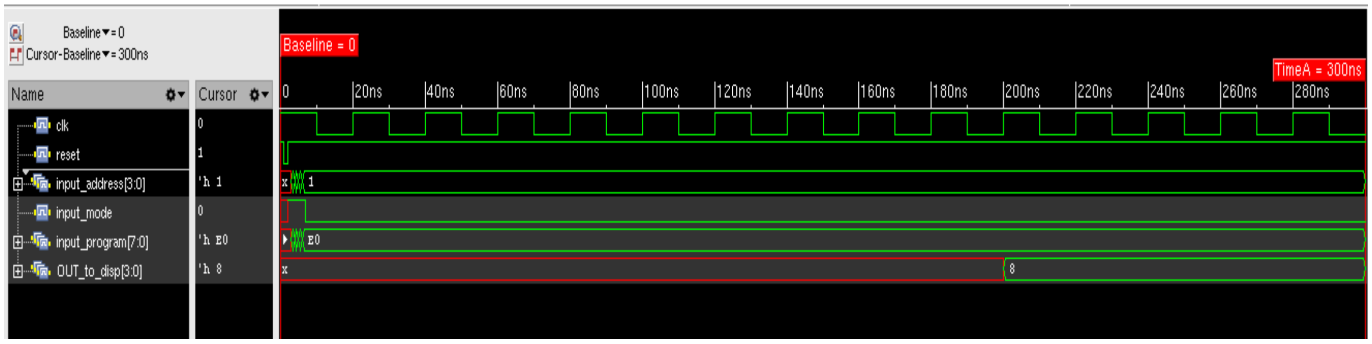


Figure 20: LDA+OUT Verification in SimVision

We have loaded A with whatever was stored in 9th location of RAM. We have seen that decimal 8 was stored there, and after OUT execution, we can see from Figure 20 decimal 8 at the OUT_to_disp wire.

3.3.3. ADD+OUT Testbench

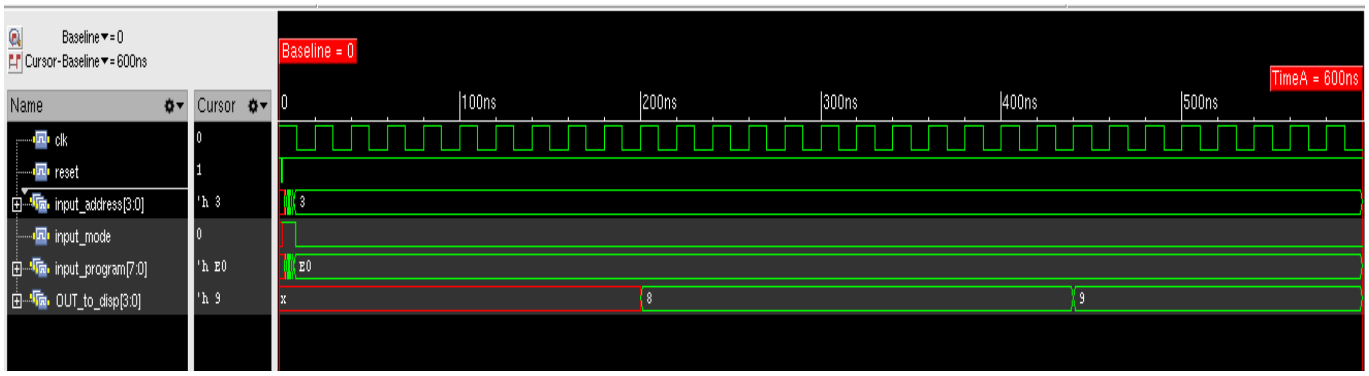


Figure 21: ADD+OUT Verification in SimVision

Now, we are adding to A whatever was stored in 10th location of RAM. We have seen that decimal 1 was stored there, and after OUT execution, we can see from Figure 21 decimal 9 at the OUT_to_disp wire.

3.3.4. SUB+OUT Testbench

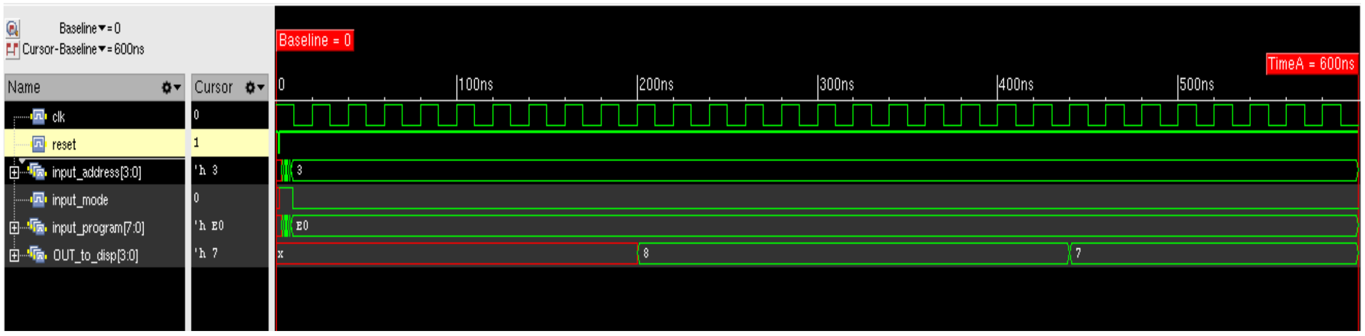


Figure 22: SUB+OUT Verification in SimVision

We have replaced the ADD instruction with the SUB instruction keeping everything else fixed. Now, we are subtracting from A whatever was stored in 10th location of RAM. We have seen that decimal 1 was stored there, and after OUT execution, we can see from Figure 22 decimal 7 at the OUT_to_disp wire.

3.3.5. HALT Testbench

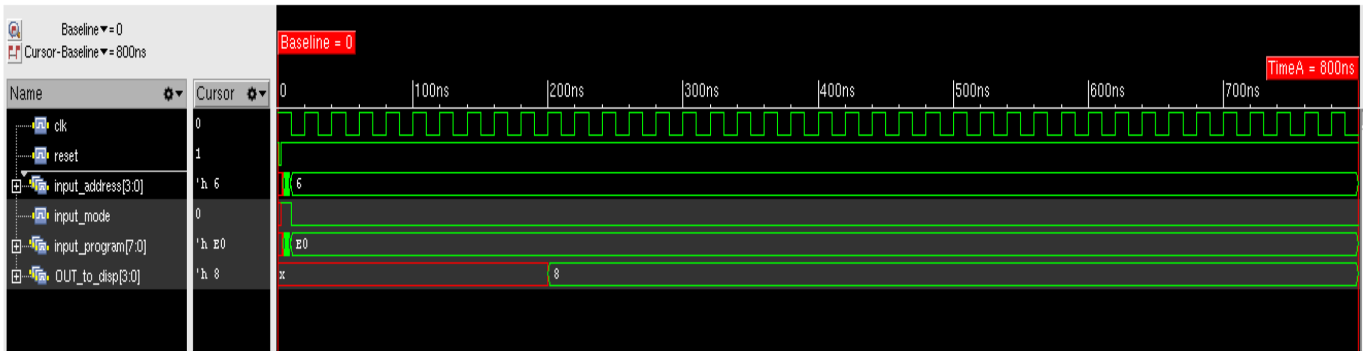


Figure 23: HALT Verification in SimVision

In the HALT testbench, there is a HALT instruction after the first LDA 9h. Then, the next ADD/SUB instruction never gets executed as demonstrated in Figure 23.

4. RTL Synthesis using Cadence Genus

For RTL logic synthesis, we have used:

1. Standard cell library for both slow corner (setup check) and fast corner (hold check) operation
2. Process technology .lef file for 45 nm tech node
3. Standard cell physical information for 45 nm tech node

The schematic, power and timing report are presented sequentially.

4.1. Schematics

The schematic of the RTL design is presented in Figure 24.

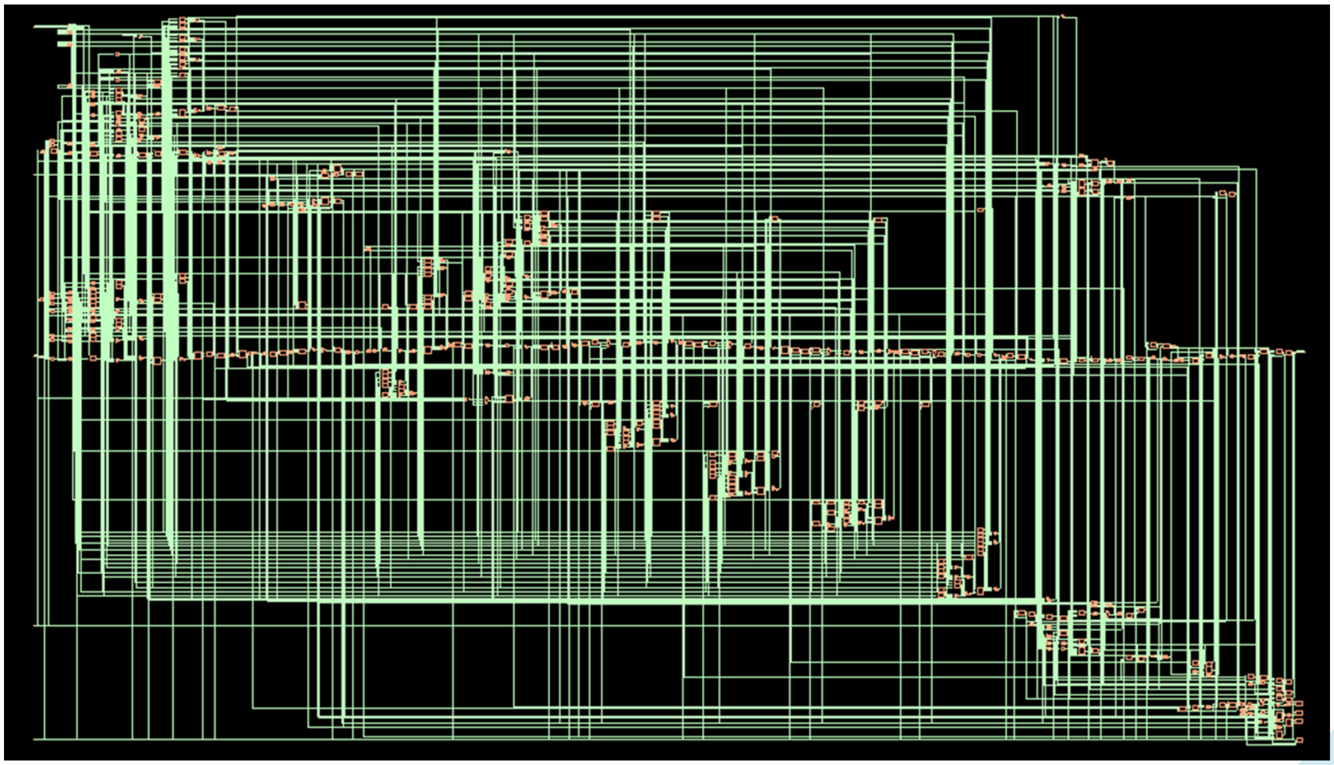
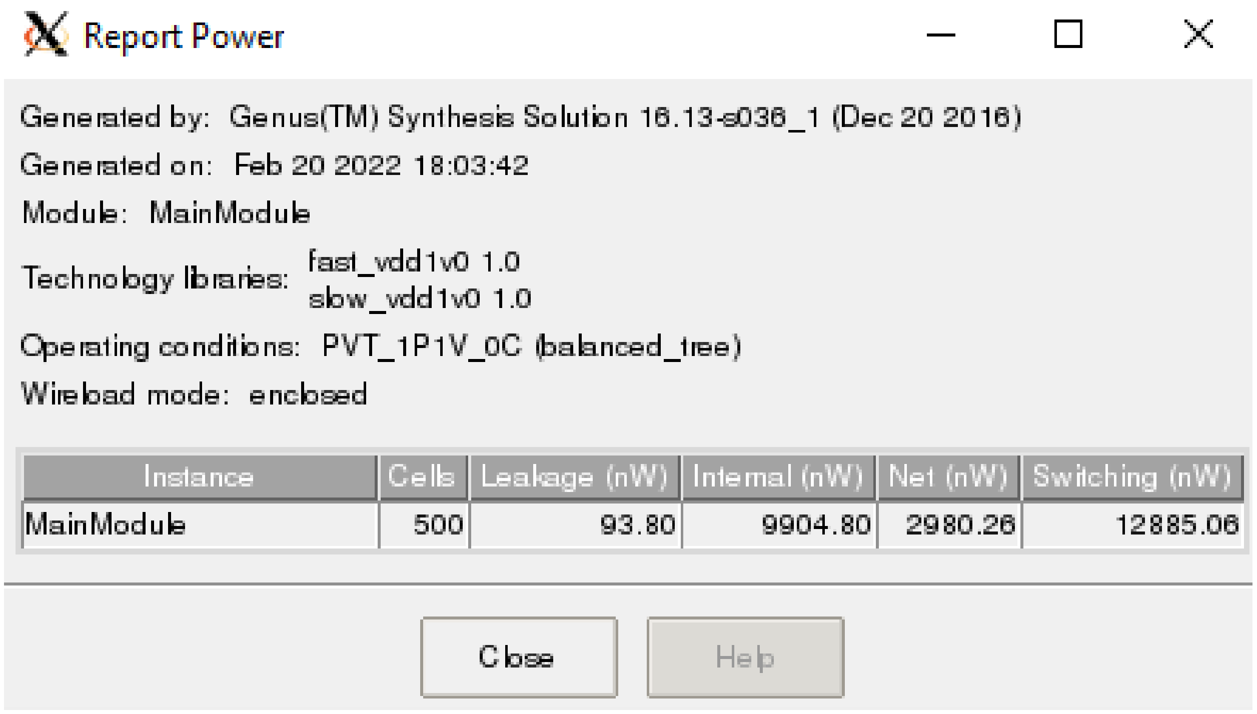


Figure 24: Schematic of final design after RTL synthesis

4.2. Power Report

The power report is presented in Figure 25.


Report Power

Generated by: Genus(TM) Synthesis Solution 16.13-s036_1 (Dec 20 2016)

Generated on: Feb 20 2022 18:03:42

Module: MainModule

Technology libraries: fast_vdd1v0 1.0
slow_vdd1v0 1.0

Operating conditions: PVT_1P1V_0C (balanced_tree)

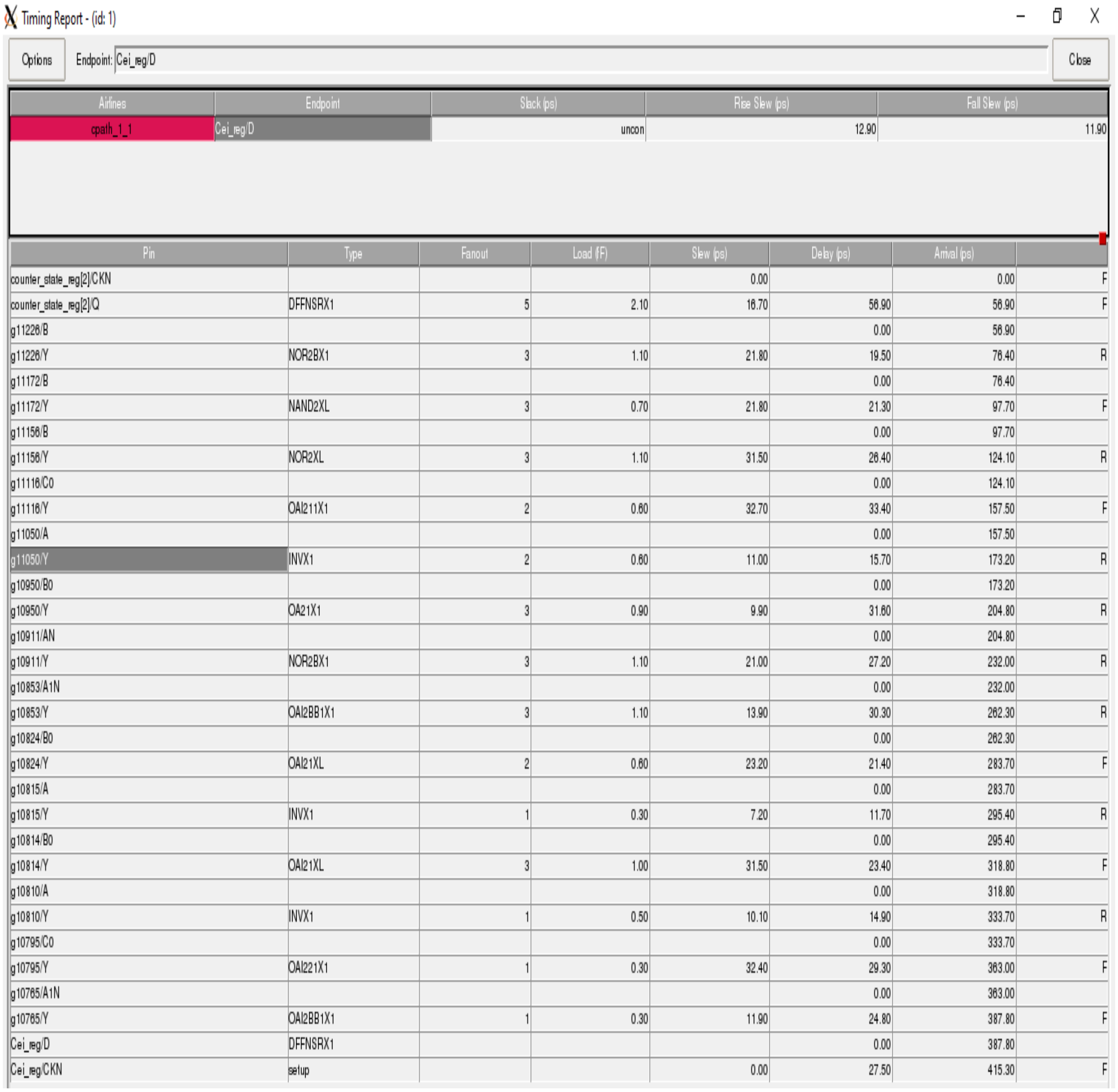
Wireload mode: enclosed

Instance	Cells	Leakage (nW)	Internal (nW)	Net (nW)	Switching (nW)
MainModule	500	93.80	9904.80	2980.26	12885.06

Figure 25: Power report

4.3. Timing Report

The timing report is presented in Figure 26.



Timing Report - (id: 1)

Options | Endpoint: Cei_reg/D | Close

Airlines	Endpoint	Slack (ps)	Rise Slew (ps)	Fall Slew (ps)
cpath_1_1	Cei_reg/D	uncon	12.90	11.90

Pin	Type	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)	
counter_state_reg[2]/CKN				0.00		0.00	F
counter_state_reg[2]/Q	DFFNSRX1	5	2.10	16.70	56.90	56.90	F
g11226/B					0.00	56.90	
g11226/Y	NOR2BX1	3	1.10	21.80	19.50	76.40	R
g11172/B					0.00	76.40	
g11172/Y	NAND2XL	3	0.70	21.80	21.30	97.70	F
g11156/B					0.00	97.70	
g11156/Y	NOR2XL	3	1.10	31.50	26.40	124.10	R
g11116/C0					0.00	124.10	
g11116/Y	OAI211X1	2	0.60	32.70	33.40	157.50	F
g11050/A					0.00	157.50	
g11050/Y	INVX1	2	0.60	11.00	15.70	173.20	R
g10950/B0					0.00	173.20	
g10950/Y	OA21X1	3	0.90	9.90	31.60	204.80	R
g10911/AN					0.00	204.80	
g10911/Y	NOR2BX1	3	1.10	21.00	27.20	232.00	R
g10853/A1N					0.00	232.00	
g10853/Y	OAI2BB1X1	3	1.10	13.90	30.30	262.30	R
g10824/B0					0.00	262.30	
g10824/Y	OAI21XL	2	0.60	23.20	21.40	283.70	F
g10815/A					0.00	283.70	
g10815/Y	INVX1	1	0.30	7.20	11.70	295.40	R
g10814/B0					0.00	295.40	
g10814/Y	OAI21XL	3	1.00	31.50	23.40	318.80	F
g10810/A					0.00	318.80	
g10810/Y	INVX1	1	0.50	10.10	14.90	333.70	R
g10795/C0					0.00	333.70	
g10795/Y	OAI221X1	1	0.30	32.40	29.30	363.00	F
g10765/A1N					0.00	363.00	
g10765/Y	OAI2BB1X1	1	0.30	11.90	24.80	387.80	F
Cei_reg/D	DFFNSRX1				0.00	387.80	
Cei_reg/CKN	setup			0.00	27.50	415.30	F

Figure 26: Timing report after RTL synthesis

5. Physical Design

In integrated circuit design, physical design is a step in the standard design cycle which follows the circuit design. At this step, circuit representations of the components (devices and interconnects) of the design are converted into geometric representations of shapes which, when manufactured in the corresponding layers of materials, will ensure the required functioning of the components. This geometric representation is called integrated circuit layout. This step is usually split into several sub-steps, which include both design and verification and validation of the layout.

5.1. Design of the layout

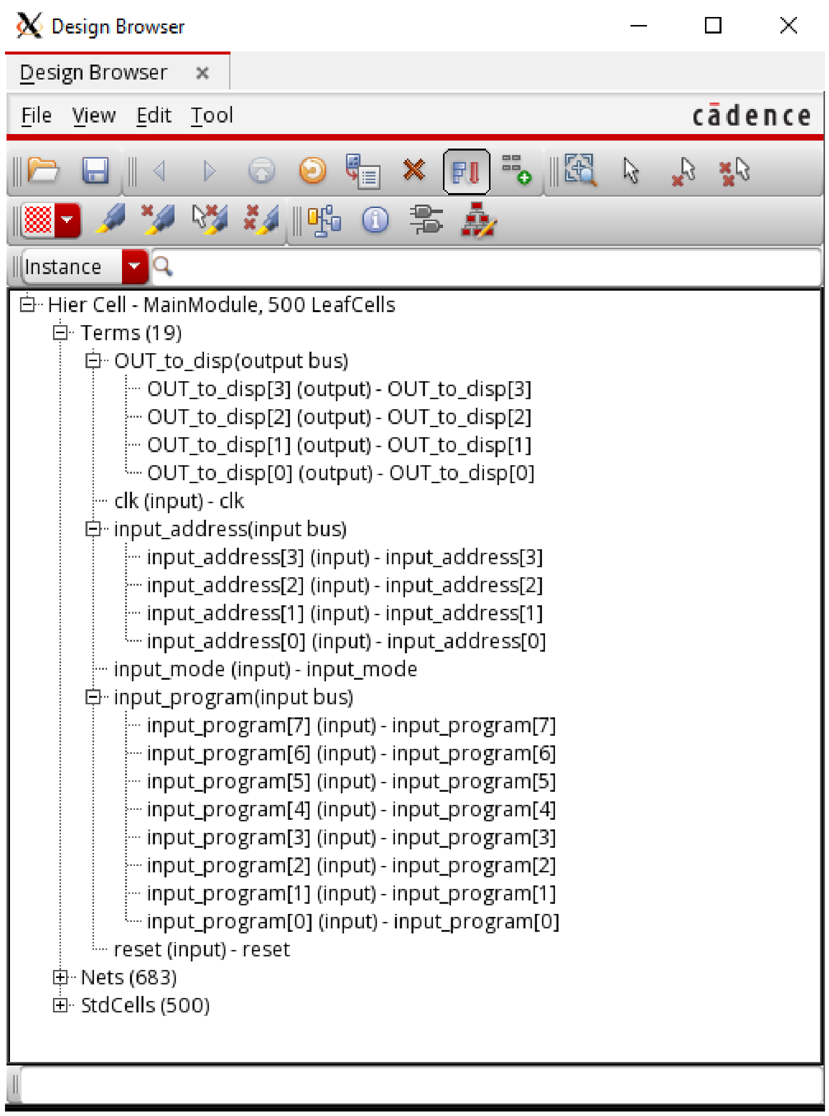
5.1.1. Floorplan

Floor-planning is the area budgeting of the chip. We can specify floorplan by explicitly specifying the width and height of the die/core.

<u>Floorplan parameters:</u>
Die size is 100 by 100
Core to left/right = 3
Core to top/bottom = 5

Then we import the synthesized netlist. The purpose of this step is to import following things and bind them to the design (checking coherence among them).
1) Our synthesized Verilog netlist
2) Technology .lef file
3) Standard cell .lef file
4) Standard cell liberty files (through setting up timing mode)

From the design browser, we can see that the total number of input pins is 15 and the total number of output pins is 4. Also, we can see from Figure 27 that our design contains 683 nets and 500 standard cells.



```
Hier Cell - MainModule, 500 LeafCells
  Terms (19)
    OUT_to_disp(output bus)
      OUT_to_disp[3] (output) - OUT_to_disp[3]
      OUT_to_disp[2] (output) - OUT_to_disp[2]
      OUT_to_disp[1] (output) - OUT_to_disp[1]
      OUT_to_disp[0] (output) - OUT_to_disp[0]
    clk (input) - clk
    input_address(input bus)
      input_address[3] (input) - input_address[3]
      input_address[2] (input) - input_address[2]
      input_address[1] (input) - input_address[1]
      input_address[0] (input) - input_address[0]
    input_mode (input) - input_mode
    input_program(input bus)
      input_program[7] (input) - input_program[7]
      input_program[6] (input) - input_program[6]
      input_program[5] (input) - input_program[5]
      input_program[4] (input) - input_program[4]
      input_program[3] (input) - input_program[3]
      input_program[2] (input) - input_program[2]
      input_program[1] (input) - input_program[1]
      input_program[0] (input) - input_program[0]
    reset (input) - reset
  Nets (683)
  StdCells (500)
```

Figure 27: Design Hierarchy

After specifying floor-planning information, we have added IO ports in the design. In our design, we have set the i/o pin spacing and width, depth as follows:

Input pins spacing = 5
Output pins spacing = 20
Pinwidth = 0.14
Pindepth = 0.28

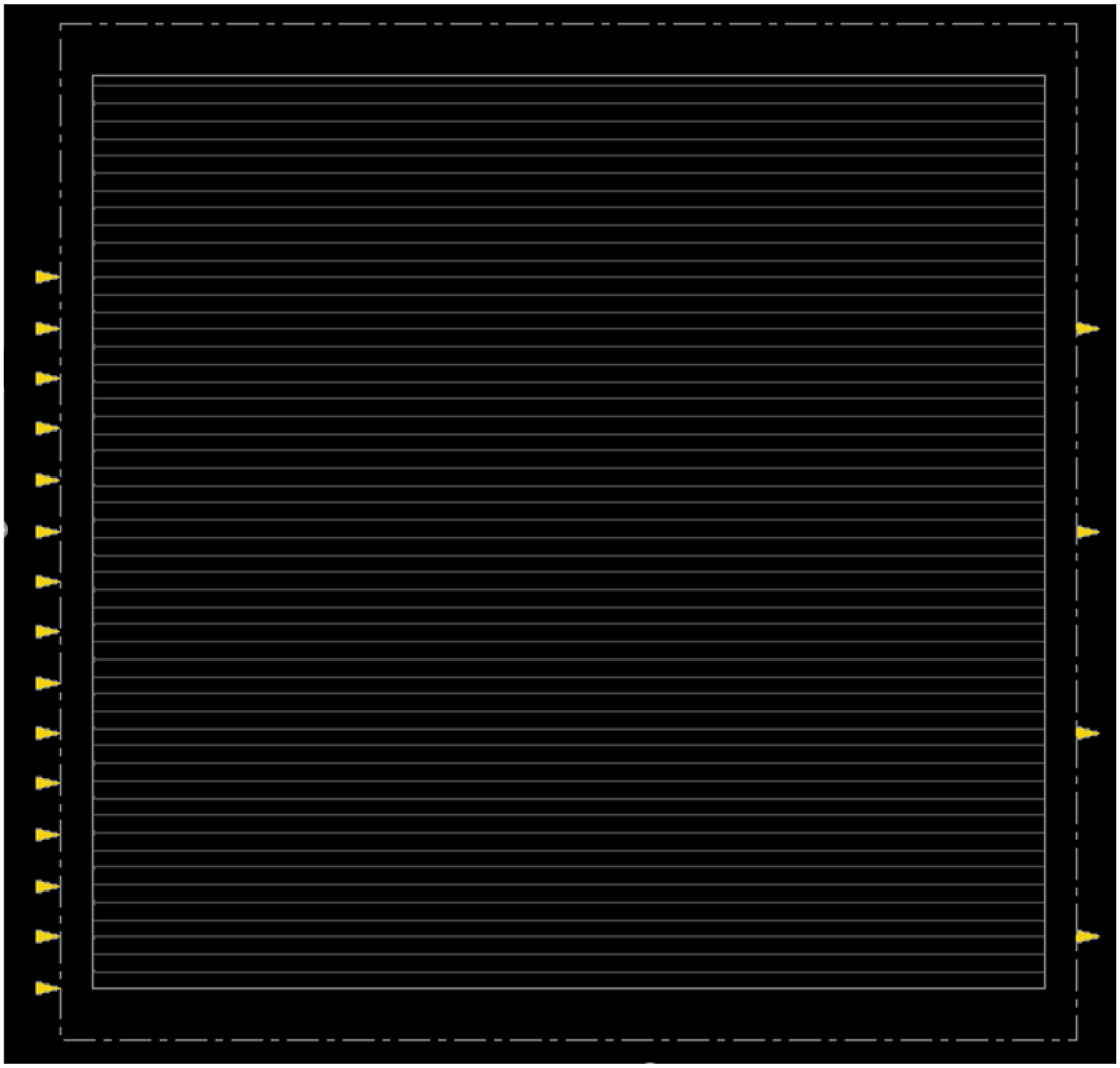


Figure 28: Layout after floorplan with i/o pins

After floor-planning is finished, the design looks like Figure 28.

5.1.2. Adding power stripe & routing

To make sure power uniformity throughout the design separate power stripes for VDD and VSS need to be created in higher metal layer. In our design, the power stripe design parameters are as follows:

- Metal4 layer is used
- Width of each strip = 0.5
- Spacing between VDD and VSS = 2
- Set to set distance = 5
- First strip starts a distance 3 away from left core boundary

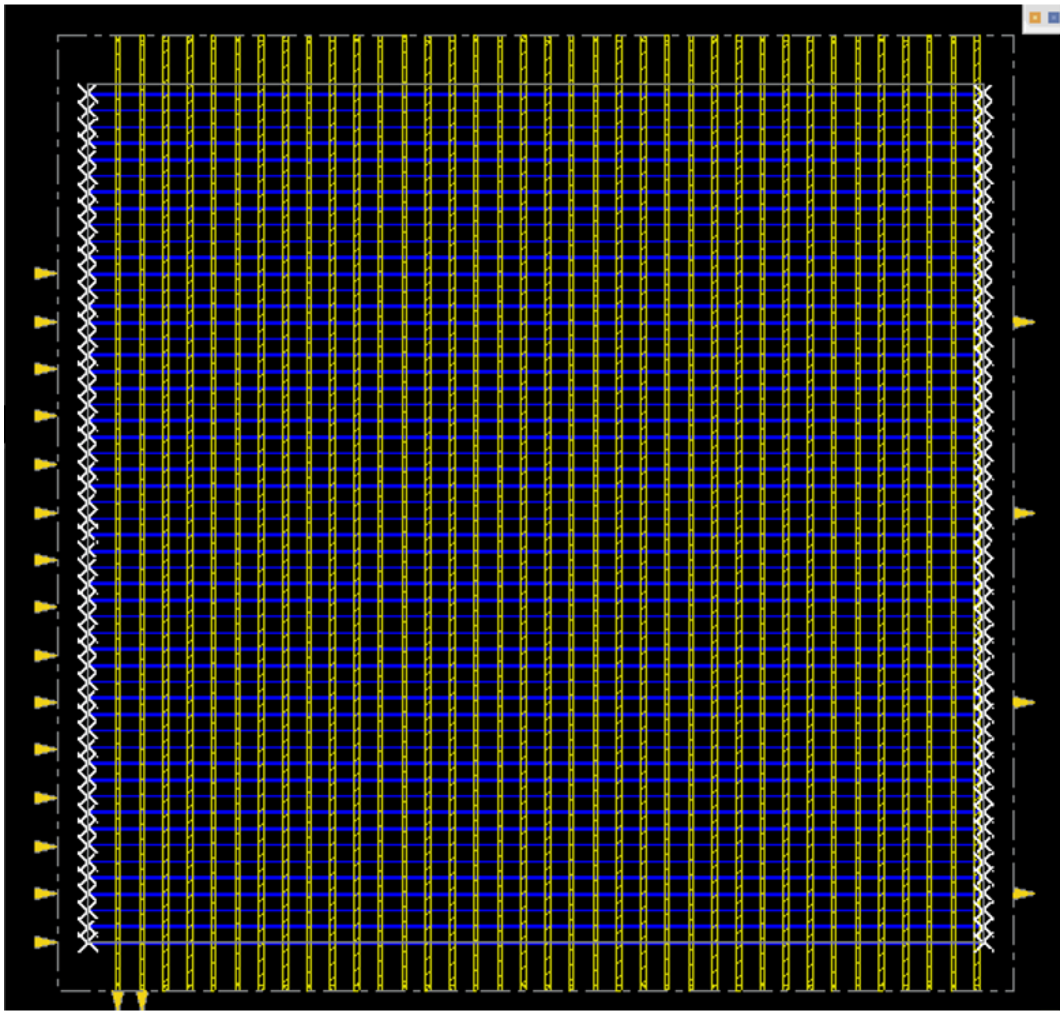


Figure 29: Layout after adding power stripe and routing

Then we have added standard cell M1 layer rail in the design. For routing, the design parameters are as follows:

- For top layer, Metal4 layer is used
- For bottom layer, Metal1 is used

After adding power stripe and routing, the design looks like Figure 29.

5.1.3. Adding optimum design

Now, we have placed standard cells in the design and optimize the placement. After this step, our design looks like Figure 30.

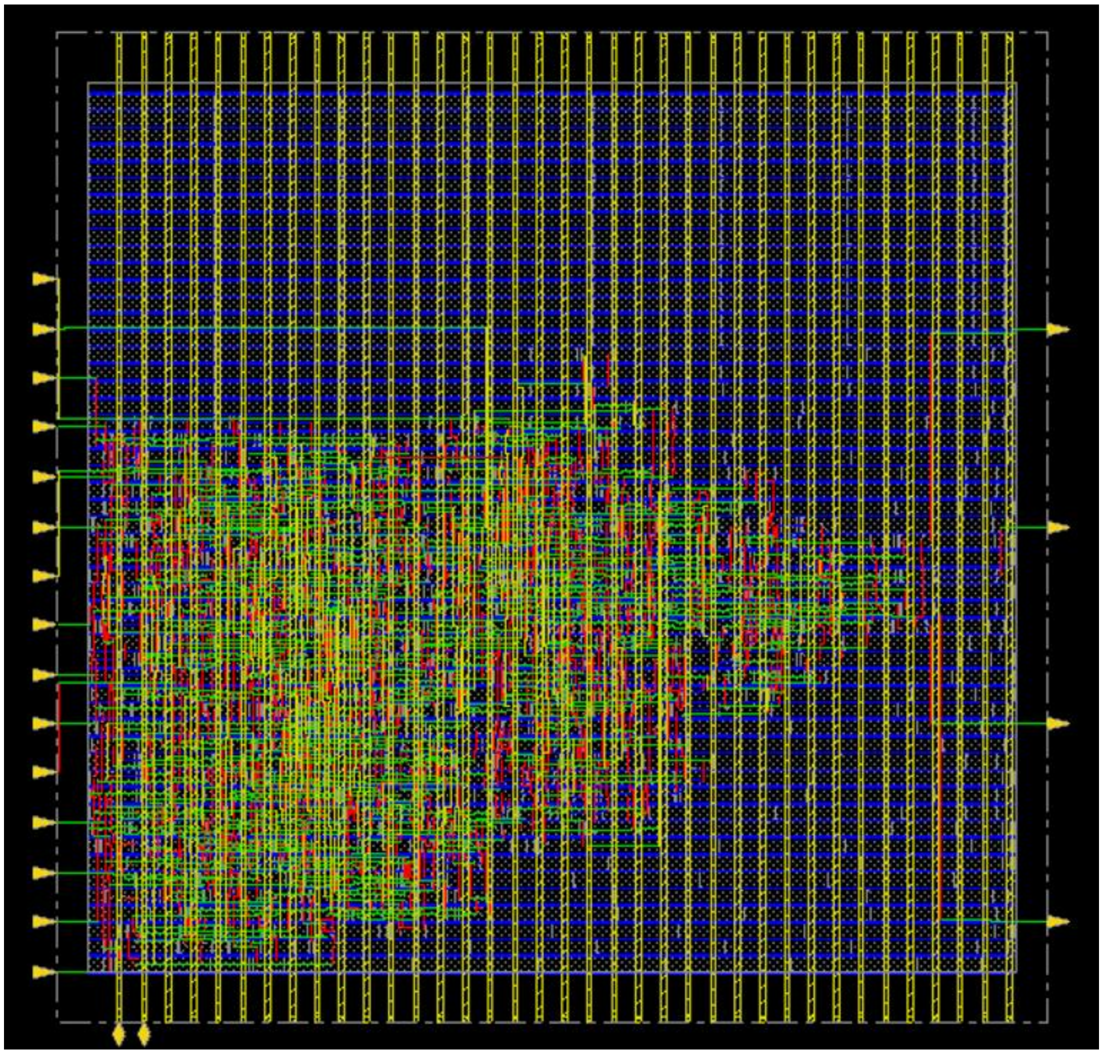
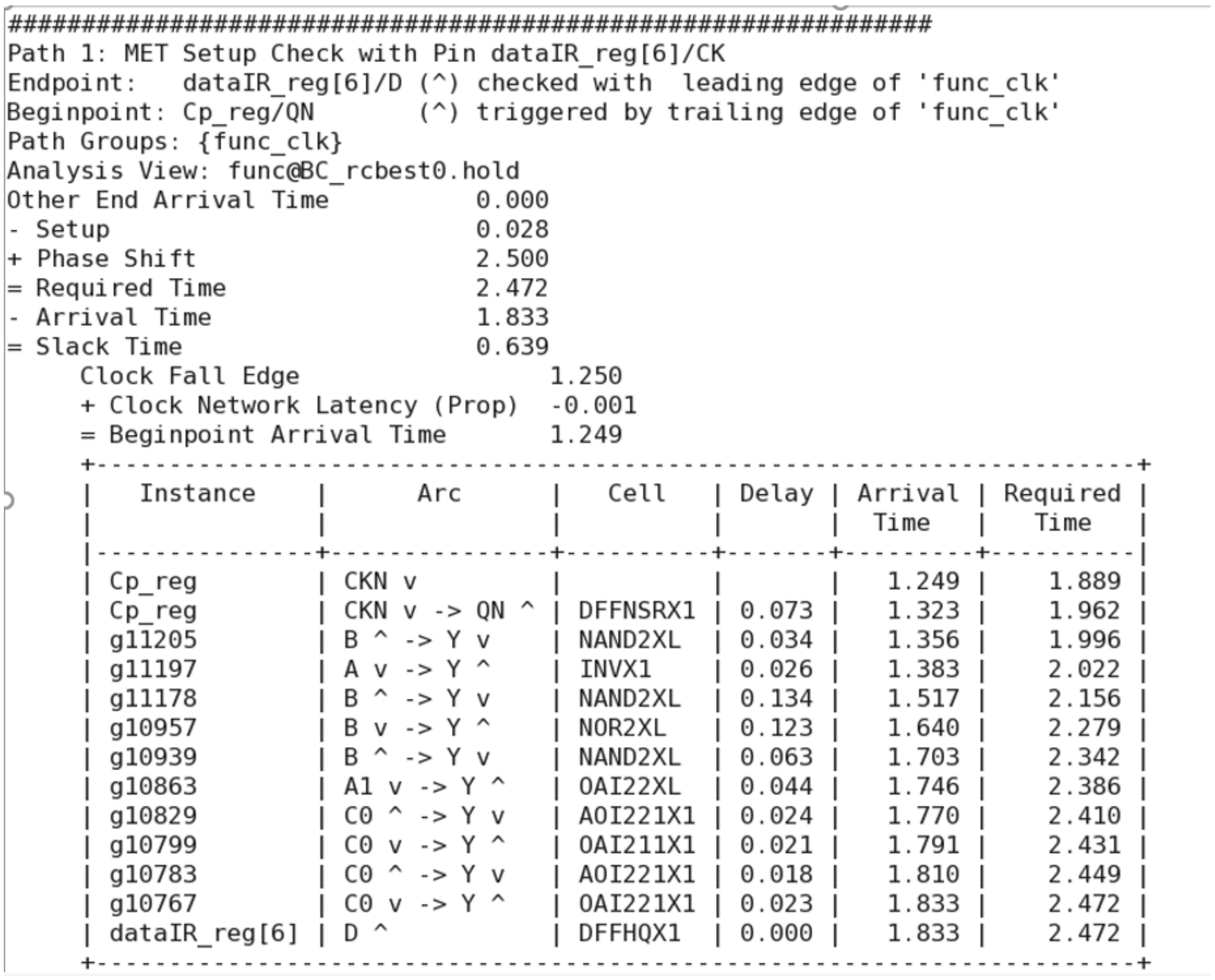


Figure 30: Layout after placing standard cells

5.1.4. Timing Report

Then we have synthesized the clock tree for our design. After the post CTS optimization and timing analysis we have generated a timing report as presented in Figure 31.



```
###############################################################
Path 1: MET Setup Check with Pin dataIR_reg[6]/CK
Endpoint:   dataIR_reg[6]/D (^) checked with  leading edge of 'func_clk'
Beginpoint: Cp_reg/QN       (^) triggered by trailing edge of 'func_clk'
Path Groups: {func_clk}
Analysis View: func@BC_rcbest0.hold
Other End Arrival Time          0.000
- Setup                         0.028
+ Phase Shift                   2.500
= Required Time                 2.472
- Arrival Time                  1.833
= Slack Time                    0.639
     Clock Fall Edge                 1.250
     + Clock Network Latency (Prop)  -0.001
     = Beginpoint Arrival Time       1.249
     +-------------------------------------------------------------------------+
     |   Instance    |      Arc      |    Cell    | Delay | Arrival | Required |
     |               |               |            |       |  Time   |   Time   |
     |---------------+---------------+------------+-------+---------+----------|
     | Cp_reg        | CKN v         |            |       |   1.249 |    1.889 |
     | Cp_reg        | CKN v -> QN ^ | DFFNSRX1   | 0.073 |   1.323 |    1.962 |
     | g11205        | B ^ -> Y v    | NAND2XL    | 0.034 |   1.356 |    1.996 |
     | g11197        | A v -> Y ^    | INVX1      | 0.026 |   1.383 |    2.022 |
     | g11178        | B ^ -> Y v    | NAND2XL    | 0.134 |   1.517 |    2.156 |
     | g10957        | B v -> Y ^    | NOR2XL     | 0.123 |   1.640 |    2.279 |
     | g10939        | B ^ -> Y v    | NAND2XL    | 0.063 |   1.703 |    2.342 |
     | g10863        | A1 v -> Y ^   | OAI22XL    | 0.044 |   1.746 |    2.386 |
     | g10829        | C0 ^ -> Y v   | AOI221X1   | 0.024 |   1.770 |    2.410 |
     | g10799        | C0 v -> Y ^   | OAI211X1   | 0.021 |   1.791 |    2.431 |
     | g10783        | C0 ^ -> Y v   | AOI221X1   | 0.018 |   1.810 |    2.449 |
     | g10767        | C0 v -> Y ^   | OAI221X1   | 0.023 |   1.833 |    2.472 |
     | dataIR_reg[6] | D ^           | DFFHQX1    | 0.000 |   1.833 |    2.472 |
     +-------------------------------------------------------------------------+
```

Figure 31: Timing report of the design

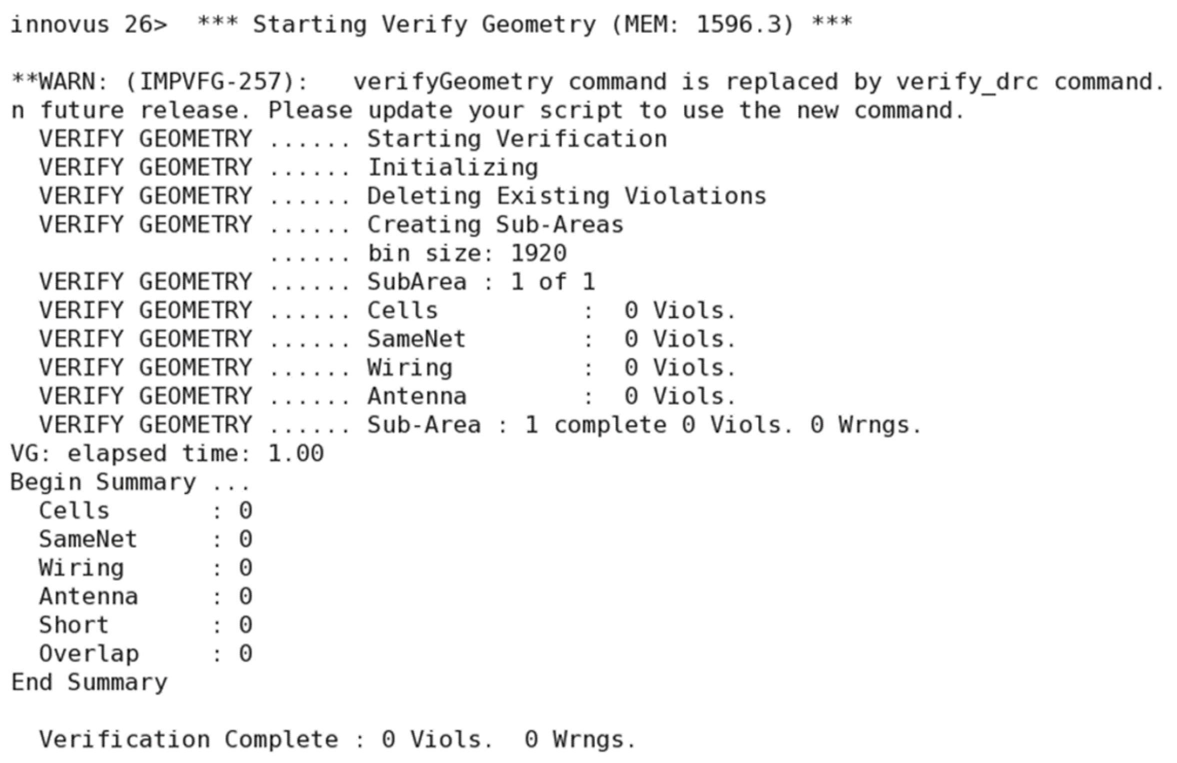
From this timing report, we can see that **Slack time is positive and less than the required time.**

5.2. Verification and Validation

To prevent, analyze and fix crosstalk, the design is run with global and detail routing using NanoRoute. When running NanoRoute, we enable both the Timing Driven and SI Driven (Signal Integrity) options in the NanoRoute form. Then the design is fully routed and at the Innovus prompt, we have run post-Route timing analysis to report any setup or hold violations.

5.2.1. Geometry Verification

After the post-route timing and SI optimization, we have verified design for design rule checks (DRC). From the geometry verification report presented in Figure 32, we can see that there is 0 violation in our design.



```
innovus 26>  *** Starting Verify Geometry (MEM: 1596.3) ***

**WARN: (IMPVFG-257):   verifyGeometry command is replaced by verify_drc command.
n future release. Please update 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.
```

Figure 32: Geometry verification report

5.2.2 After Metal Fill

In order to avoid DRC errors later, it is usually a good idea to place fill cell to fill in the gaps between the placed standard cells. It provides mechanical stability and substrate uniformity of the chip. In our design, all the filler cell libraries are selected. After the metal fill of empty space, the layout looks like Figure 33.

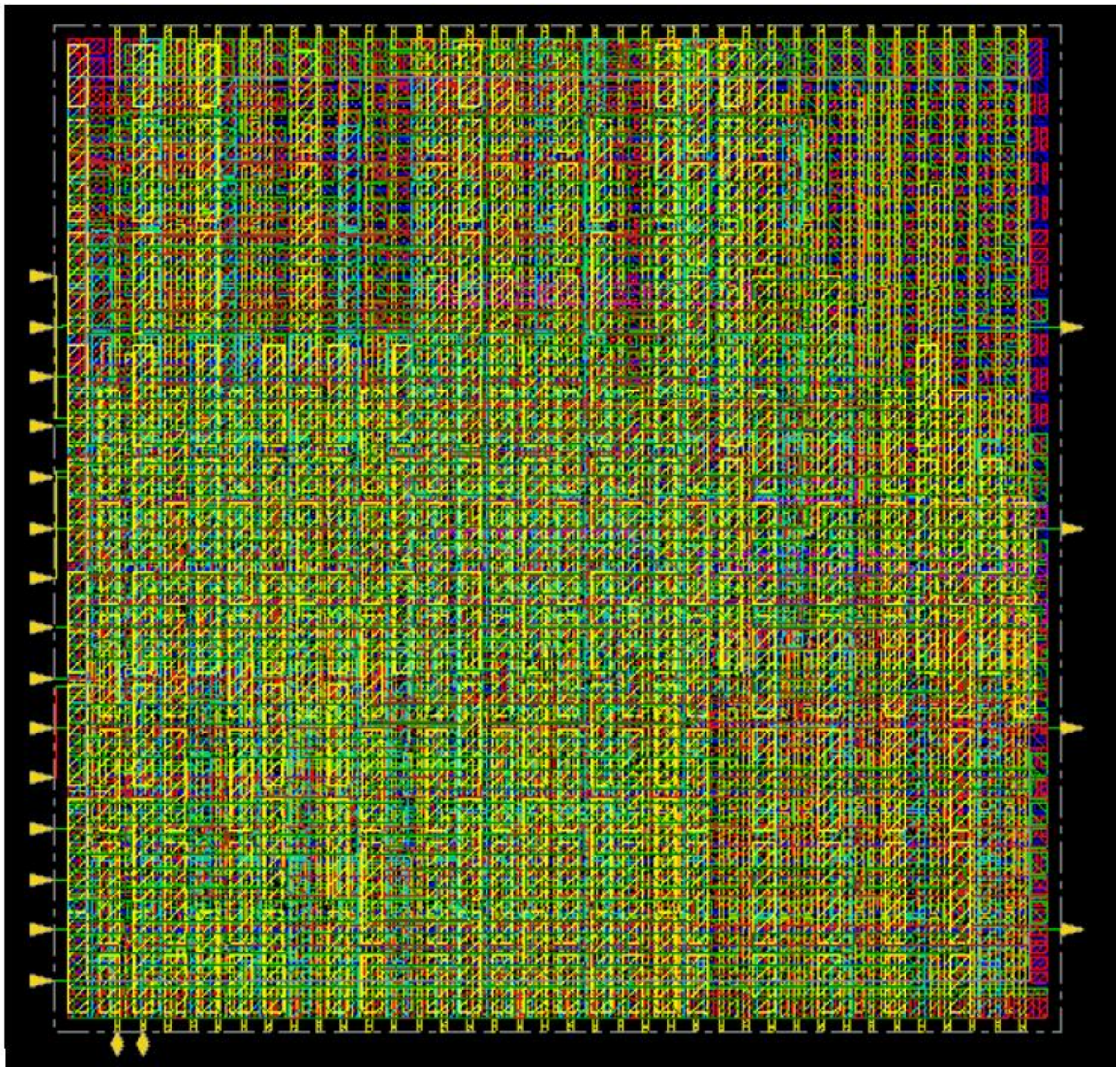
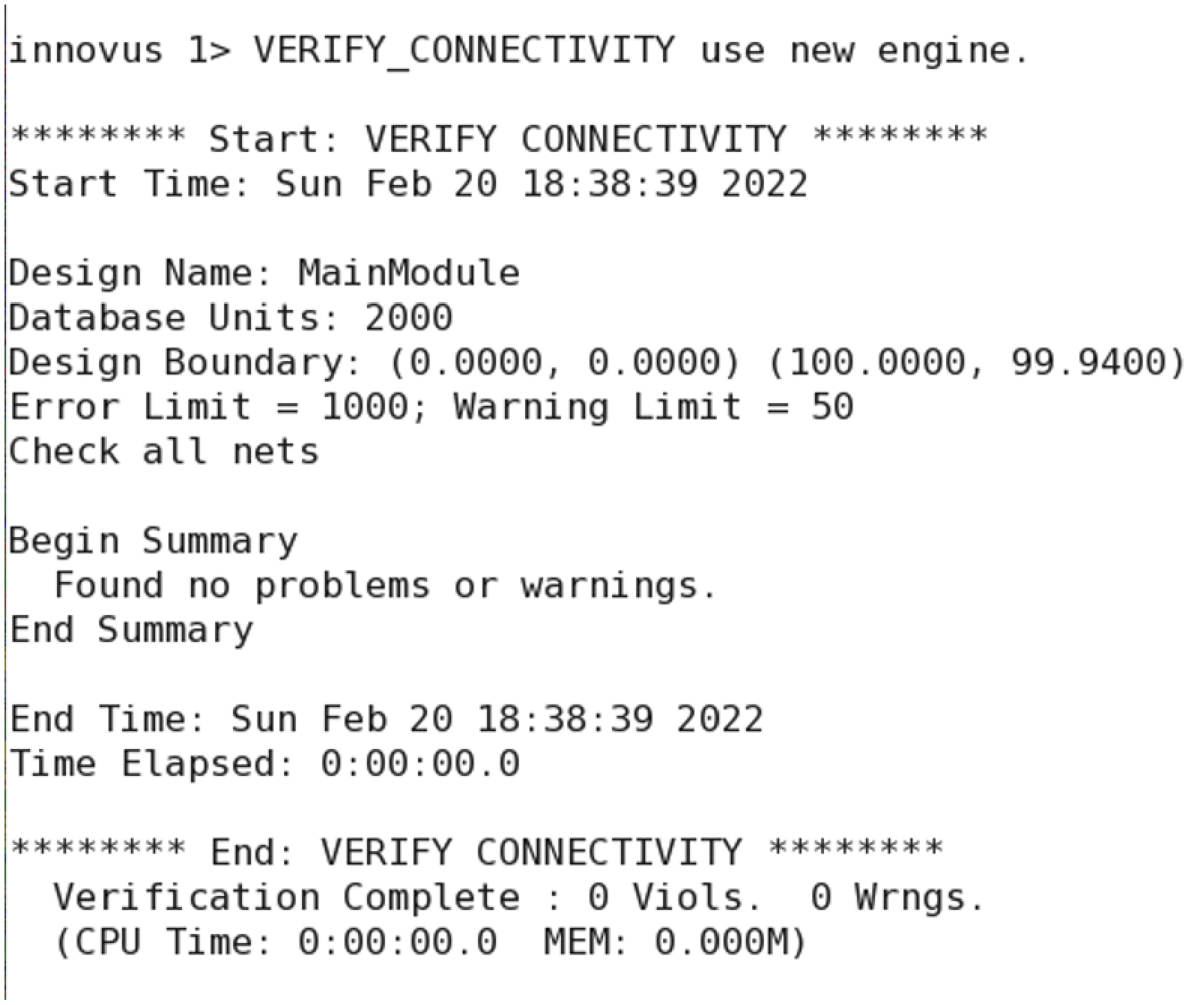


Figure 33: Layout design after metal filling

5.2.3 Connectivity Verification

After adding the filler cells and metal filling, we have verified the connectivity again. We have got 0 violations in this verification as shown in Figure 34.

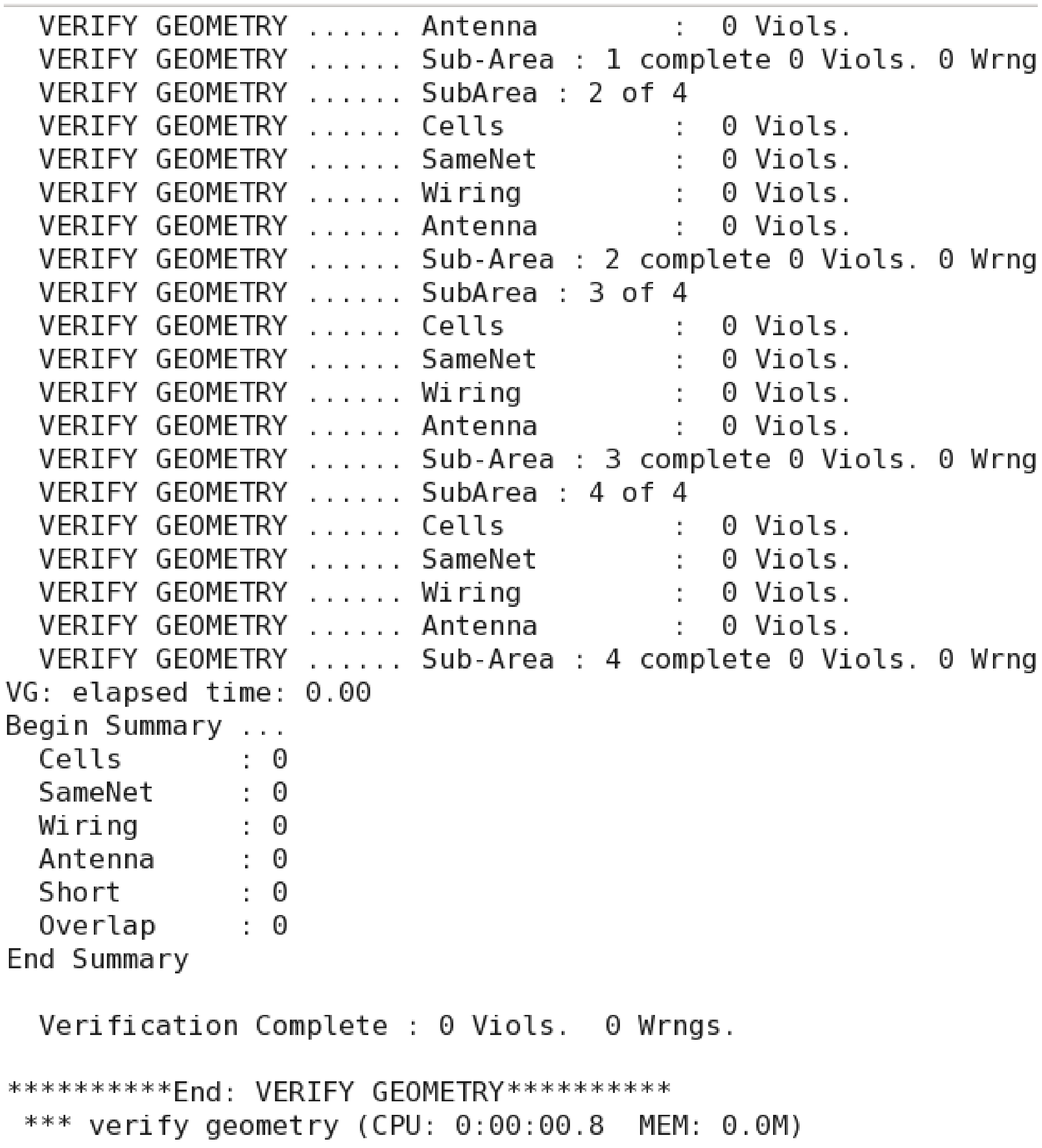


```
innovus 1> VERIFY_CONNECTIVITY use new engine.

******** Start: VERIFY CONNECTIVITY ********
Start Time: Sun Feb 20 18:38:39 2022

Design Name: MainModule
Database Units: 2000
Design Boundary: (0.0000, 0.0000) (100.0000, 99.9400)
Error Limit = 1000; Warning Limit = 50
Check all nets

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Sun Feb 20 18:38:39 2022
Time Elapsed: 0:00:00.0

******** End: VERIFY CONNECTIVITY ********
  Verification Complete : 0 Viols.  0 Wrngs.
  (CPU Time: 0:00:00.0  MEM: 0.000M)
```

Figure 34: Final connectivity verification of design

5.2.4 Final Geometry Verification

Then we have verified the geometry again and we can see from Figure 35 that the geometry has 0 violation error.



```
  VERIFY GEOMETRY ...... Antenna        :  0 Viols.
  VERIFY GEOMETRY ...... Sub-Area : 1 complete 0 Viols. 0 Wrng
  VERIFY GEOMETRY ...... SubArea : 2 of 4
  VERIFY GEOMETRY ...... Cells          :  0 Viols.
  VERIFY GEOMETRY ...... SameNet        :  0 Viols.
  VERIFY GEOMETRY ...... Wiring         :  0 Viols.
  VERIFY GEOMETRY ...... Antenna        :  0 Viols.
  VERIFY GEOMETRY ...... Sub-Area : 2 complete 0 Viols. 0 Wrng
  VERIFY GEOMETRY ...... SubArea : 3 of 4
  VERIFY GEOMETRY ...... Cells          :  0 Viols.
  VERIFY GEOMETRY ...... SameNet        :  0 Viols.
  VERIFY GEOMETRY ...... Wiring         :  0 Viols.
  VERIFY GEOMETRY ...... Antenna        :  0 Viols.
  VERIFY GEOMETRY ...... Sub-Area : 3 complete 0 Viols. 0 Wrng
  VERIFY GEOMETRY ...... SubArea : 4 of 4
  VERIFY GEOMETRY ...... Cells          :  0 Viols.
  VERIFY GEOMETRY ...... SameNet        :  0 Viols.
  VERIFY GEOMETRY ...... Wiring         :  0 Viols.
  VERIFY GEOMETRY ...... Antenna        :  0 Viols.
  VERIFY GEOMETRY ...... Sub-Area : 4 complete 0 Viols. 0 Wrng
VG: elapsed time: 0.00
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: 0:00:00.8  MEM: 0.0M)
```

Figure 35: Final geometry verification of the design

5.2.5 Layout at Virtuoso

After the connectivity and geometry verification, we have generated a .gds file and imported in Cadence Virtuoso without any error. The layout window looks as shown in Figure 36.

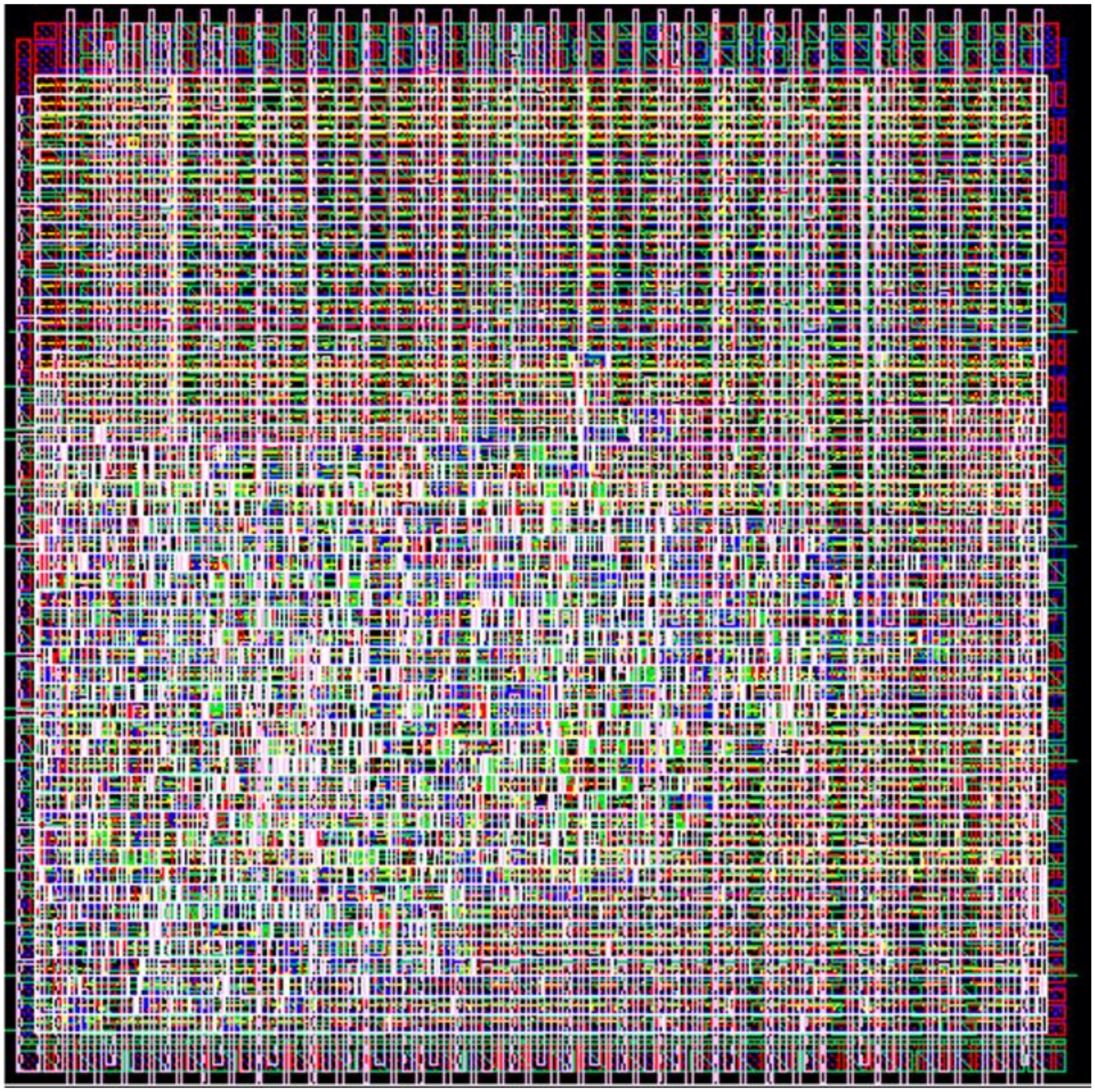


Figure 36: Final layout of the design

5.2.6 DRC Error List

Now, we have invoked PVS plugin to run DRC check. After the DRC check, we can see from Figure 37 that we have 0 DRC error in our design.

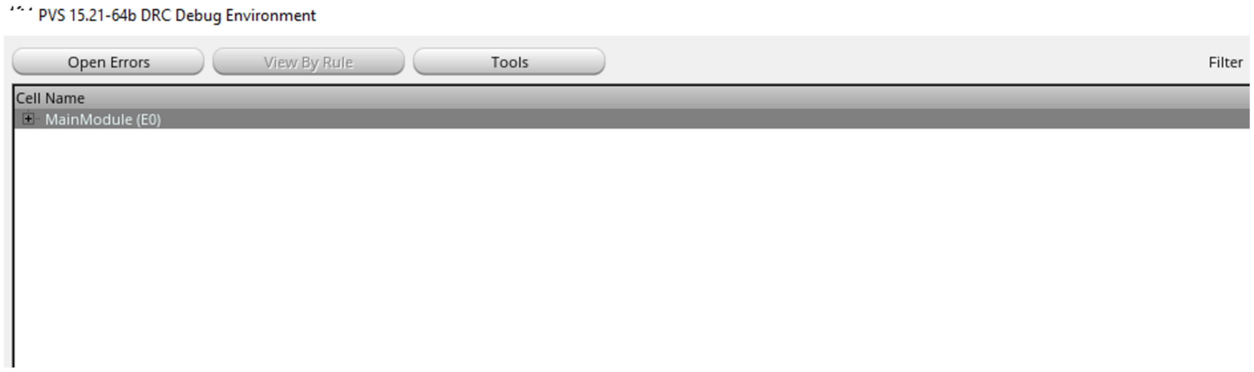


Figure 37: DRC error list

6. Design Rules Check (DRC) Errors

6.1. Terminologies

The type of DRC errors faced at the end of our physical design and their explanation is presented below [1].

- **NIMP.SP.1** → Minimum N+ Implant Space >= 0.12 um
- **PIMP.SP.1** → Minimum P+ Implant Space >= 0.12 um
- **NW.SP.1** → Minimum N-well spacing to N-well (same potential) >= 0.3um
- **NW.W.1** → Minimum N-well width >= 0.3um
- **Offgrid check** → Drawn layer is not aligned with the grid settings
- **VIA1.E.3** → At least 2 Via1 must be used to join Metal1 and Metal2 when they are within 3.0 um of a metal plate(Metal1/Metal2) when the metal plate size has width>1.5 and length > 1.5
- **VIA2.E.3** → At least 2 Via2 must be used to join Metal2 and Metal3 when they are within 3.0 um of a metal plate(Metal2/Metal3) when the metal plate size has width>1.5 and length > 1.5
- **VIA3.E.3** → At least 2 Via3 must be used to join Metal23and Metal4 when they are within 3.0 um of a metal plate(Metal3/Metal4) when the metal plate size has width>1.5 and length > 1.5

6.2. Trial and Error

Table 8: Trial and Error

Case No	Physical Design Parameter	DRC Errors	Comment
1	Die Size: 100*100 Input pin spacing: 2 Output pin spacing: 5 Other design parameters are followed from lab-sheet	NIMP.SP.1 → 34 NW.SP.1 → 29 NW.W.1 → 5 PIMP.SP.1 → 31 Offgrid check → 98	Total Error = 196 Offgrid check error due to Geometry Violation error
2	Same Die Size, Input pin spacing & Output pin spacing <u>Core to IO boundary</u> Core to Left/Right: 3 Core to Top/Bottom: 4.94 Power Stripe Width: 0.5 Power Stripe Spacing: 2 First Power Stripe (X from Left): 3	NIMP.SP.1 → 39 NW.SP.1 → 26 NW.W.1 → 2 PIMP.SP.1 → 35 VIA1.E.3 → 1 VIA2.E.3 → 2 VIA3.E.3 → 1	Total Error = 106 Offgrid check errors are gone after increasing power stripe width and spacing, but VIA errors are introduced
3	Die Size: 100*100 Input pin spacing: 5 Output pin spacing: 20 Other design parameters are followed from case 2	NIMP.SP.1 → 25 NW.SP.1 → 25 NW.W.1 → 6 PIMP.SP.1 → 29	Total Error = 85 VIA errors are gone after increasing i/o pin spacing
4	Die Size: 200*150 Other design parameters are followed from case 2	NIMP.SP.1 → 19 NW.SP.1 → 11 NW.W.1 → 9 PIMP.SP.1 → 14	Total Error = 43 Spacing error decreases but well error increases due to increasing die size

5	Die Size: 200*150 Other design parameters are followed from case 4 & Adding all the filler cell	0	Total Error = 0 All errors are gone after increasing fill density
6	Die Size: **100*100** Other design parameters are followed from case 3 & Adding all the filler cell	0	Total Error = 0 Still no error after optimizing die size

7. Conclusion

In this project, we have built a 4-bit simple as possible computer using VerilogHDL. Initially, we wrote separate modules for each block but after facing issues during synthesis, we decided to rewrite the code and convert it into single module. To demonstrate that the computer is working properly, we have written testbenches for all instructions separately and presented the waveforms both using EDA tools and Cadence SimVision. Using Cadence Genus allowed us to see the full view of the schematic along with the power and timing report. Then, using Cadence Innovus, we designed the physical layout and during this process, we faced multiple errors. We investigated the meaning of the errors and performed a rigorous trial and error to finally get rid of them. The trial and error also helped us identify which design parameter can affect a particular type of error to increase or decrease.

8. Acknowledgement

We would like to thank Dr. A.B.M. Harun-Ur-Rashid and Rifat Shahriar for their guidance in this project.

List of References

[1]https://perso.telecom-paristech.fr/mathieu/ICS904_TP_LAYOUT/html/_static/doc/gpdk045_drc.pdf