MANIPAL ACADEMY *of* HIGHER EDUCATION

(Deemed to be University under Section 3 of the UGC Act, 1956)

MANIPAL SCHOOL OF INFORMATION SCIENCES

(A Constituent unit of MAHE, Manipal)

Design [RTL] and physical design of Asynchronous FIFO

Reg. Number	Name	Branch
221038034	NAGARAJ M S	ME [VLSI DESIGN]
221038035	SIREESH O J	ME [VLSI DESIGN]

Under the guidance of

Dr. Madhushankara M

Associate Professor

Manipal School of Information Sciences,

MAHE, MANIPAL

09/05/2023

MANIPAL SCHOOL OF INFORMATION SCIENCES

MANIPAL

(A constituent unit of MAHE, Manipal)

CONTENTS

LIST OF FIGURES

1. Introduction

In computer programming, FIFO (first in, first-out) is an approach to handling program work requests from queues or stacks so that the oldest request is handled first. In hardware it is either an array of flops or Read/Write memory that store data given from one clock domain and on request supplies with the same data to another clock domain following the first in first out logic. The clock domain that supplies data to FIFO is often referred as WRITE OR INPUT LOGIC and the clock domain that reads data from the FIFO is often referred as READ OR OUTPUT LOGIC. FIFOs are used in designs to safely pass multi-bit data words from one clock domain to another or to control the flow of data between source and destination side sitting in the same clock domain. If read and write clock domains are governed by same clock signal the FIFO is said to be SYNCHRONOUS and if read and write clock domains are governed by different (asynchronous) clock signals FIFO is said to be ASYNCHRONOUS

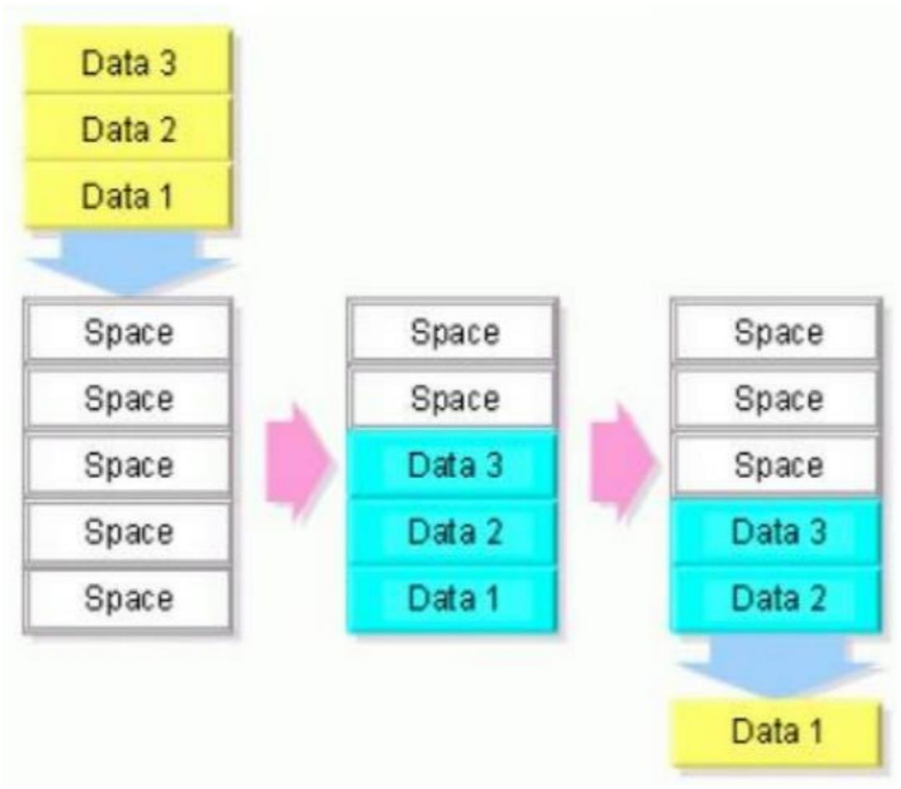


Figure 1: FIFO [First In First Out]

FIFO full and FIFO empty flags are of great concern as no data should be written in full condition and no data should be read in empty condition, as it can lead to loss of data or generation of non-relevant data The full and empty conditions of FIFO are controlled using binary or gray pointers.

Asynchronous FIFO

An asynchronous FIFO refers to a FIFO design where data values are written sequentially into a FIFO buffer using one clock domain, and the data values are sequentially read from the same FIFO buffer using another clock domain, where the two clock domains are asynchronous to each other.

2. Objective(s)

- RTL Design of Asynchronous FIFO.
- Generating the netlist by doing the synthesis.
- LEC [Logical Equivalence Check].
- Physical design.

3. Block diagram

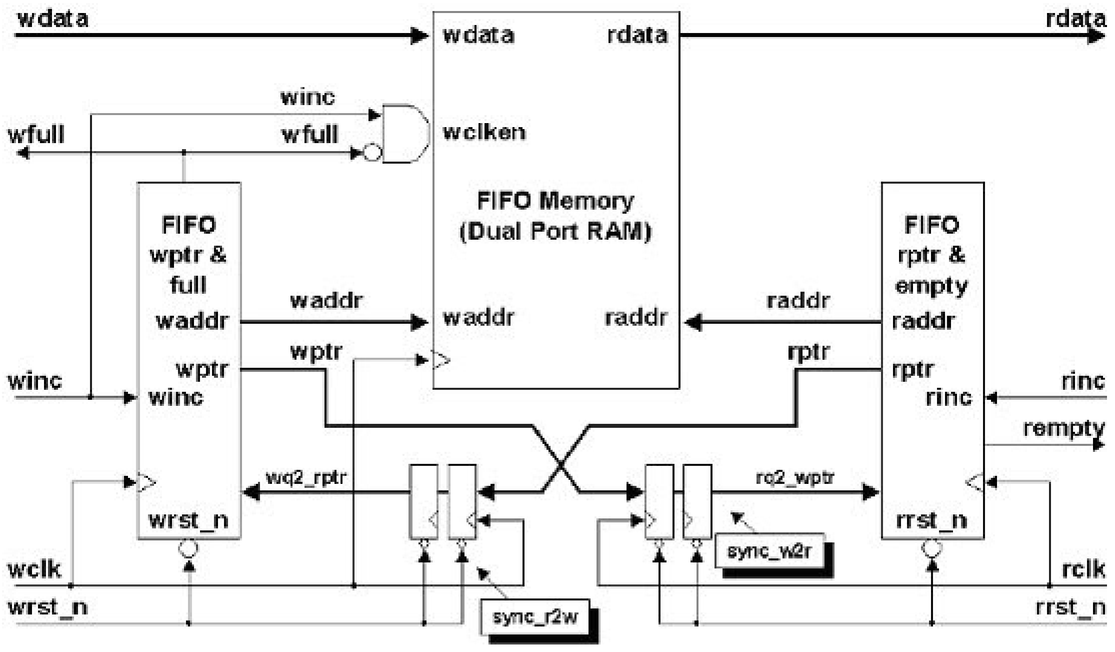


Figure 2: Block diagram of Asynchronous FIFO

Above figure shows the block diagram of asynchronous fifo in this design there are five modules each module is explained below.

3.1 Memory array

- It is an array of flip-flops, which stores data.
- Number of data words that the memory array can store is often referred as DEPTH of the FIFO.
- Length of the data word is referred as WIDTH of the FIFO.
- Besides flop-array it comprises read and write address decoding logic.

3.2 Read-domain to write-domain synchronizer[sync_r2w]

This is a simple synchronizer module, used to pass an n-bit pointer from the read clock domain to the write clock domain, through a pair of registers that are clocked by the FIFO write clock.

3.3 Write-domain to read-domain synchronizer[sync_w2r]

This is a simple synchronizer module, used to pass an n-bit pointer from the write clock domain to the read clock domain, through a pair of registers that are clocked by the FIFO read clock.

3.4 Read pointer & empty generation logic.

The read pointer is a dual n-bit Gray code counter. The n-bit pointer (rptr) is passed to the write clock domain through the sync_r2w module. The (n-1)-bit pointer (raddr) is used to address the FIFO buffer.
To efficiently register the rempty output, the synchronized write pointer is compared against the rgraynext (the next gray code that will be registered into the rptr)

3.5 Write pointer & full generation logic.

The write pointer is a dual n-bit Gray code counter. The n-bit pointer (wptr) is passed to the read clock domain through the sync_w2r module. The (n-1)-bit pointer (waddr) is used to address the FIFO buffer.
synchronizing the rptr into the wclk domain and then there are three conditions that are all necessary for the FIFO to be full.

(1) The wptr and the synchronized rptr MSB's are not equal (because the wptr must have wrapped one more time than the rptr).

(2) The wptr and the synchronized rptr 2nd MSB's are not equal (because an inverted 2nd MSB from one pointer must be tested against the un-inverted 2nd MSB from the other pointer.

(3) All other wptr and synchronized rptr bits must be equal.
In order to efficiently register the wfull output, the synchronized read pointer is actually compared against the wgnext (the next Gray code that will be registered in the wptr).

4. Simulation waveform

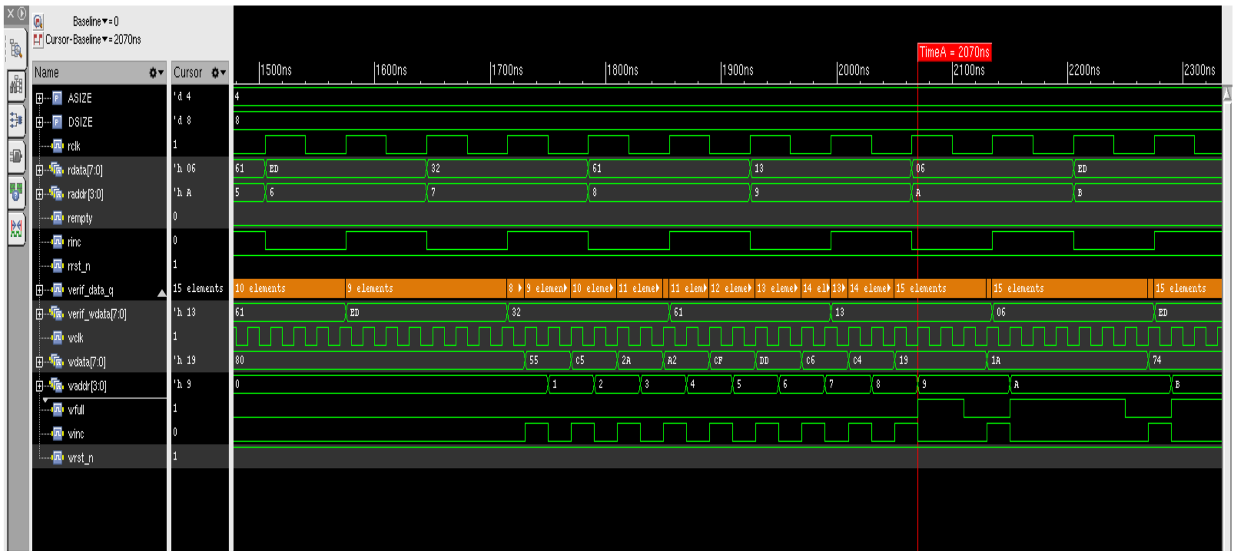


Figure 3:waveform showing that FIFO is full.

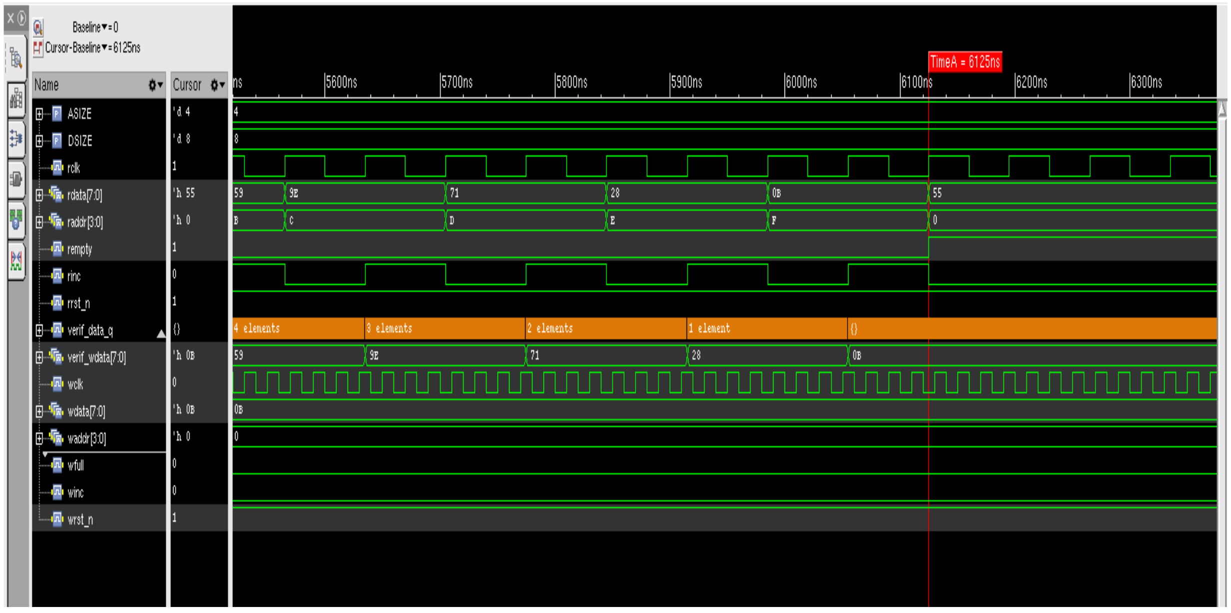


Figure 4: waveform showing FIFO empty.

5. RTL file to Netlist

Commands to covert RTL to NETLIST

- **set_attribute lib_search_path ./library**
  - Setting the path to read the library file.
- **set_attribute library fast.lib**
  - Reading the timing library.
- **set_attribute hdl_search_path ./rtl**
  - Setting the path to read the RTL file.
- **read_hdl -sv design.sv**
  - Reading the design file.
- **Elaborate**
  - Hierarchy of design is extracted.
- **syn_generic**
  - Covert the netlist to generic gates netlist.
- **write_hdl > ./unmapped/fifo_generic.sv**
  - Before mapping writing netlist into file.
- **read_sdc ./constraints/fifo.sdc**
  - Reading the constraints file.
- **syn_map**
  - Mapping the generic gates to gates present in technology library.
- **syn_opt**
  - Optimizing the design.
- **write_hdl > ./mapped/fifo_mapped.sv**
  - Writing the netlist into the file.
- **write_sdc > ./mapped/fifo_mapped.sdc**
  - Writing the constraints into the file .
- **report power > ./reports/fifo_power.txt**
  - Writing the power report into the file.

- **report area > ./reports/fifo_area.txt**
  - Writing the area report into the file.

- **report timing > ./reports/fifo_timing.txt**
  - Writing the timing report into the file.

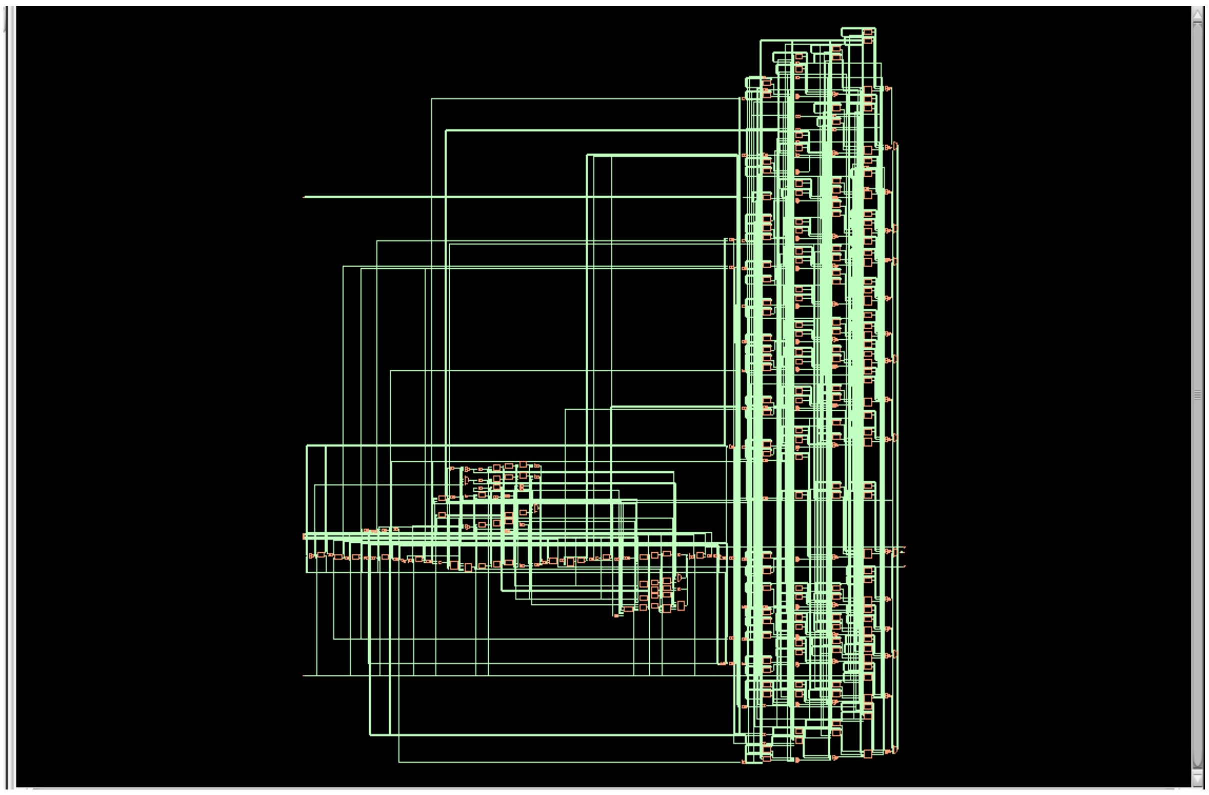


Figure 5:Netlist Schematic

6. LEC (Logical Equivalence Check)

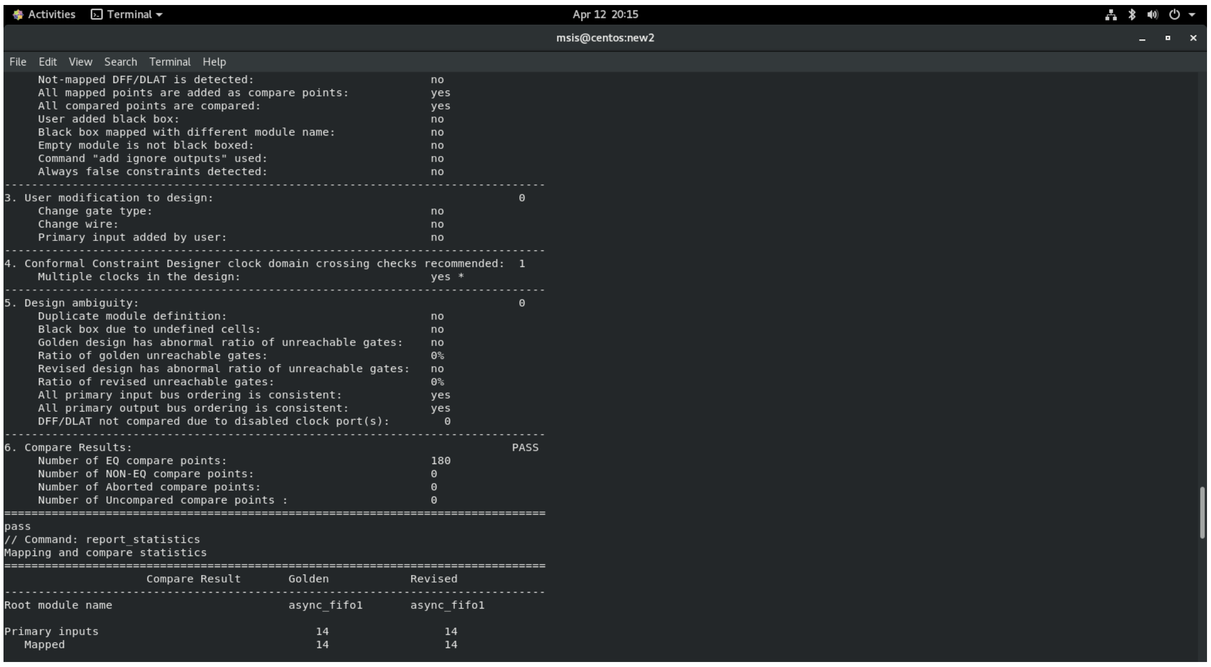
Logic equivalence checking (LEC) looks at the combinatorial structure of the design to determine if the structure of two alternative implementations will exhibit the same behavior. Our RTL design and netlist file generated is compared in this and will gives us about both are equivalent or not.

Command to generate the dofile

Write_do_lec -golden_design rtl_file.v -revised_design netlist.v > rtltofinal.lec.do

Command to check equivalence.

Lec -XL -nogui -dofile rtltofinal.lec.do



```
     Not-mapped DFF/DLAT is detected:                          no
     All mapped points are added as compare points:            yes
     All compared points are compared:                         yes
     User added black box:                                     no
     Black box mapped with different module name:              no
     Empty module is not black boxed:                          no
     Command "add ignore outputs" used:                        no
     Always false constraints detected:                        no
--------------------------------------------------------------------------------
3. User modification to design:                                             0
     Change gate type:                                         no
     Change wire:                                              no
     Primary input added by user:                              no
--------------------------------------------------------------------------------
4. Conformal Constraint Designer clock domain crossing checks recommended:  1
     Multiple clocks in the design:                            yes *
--------------------------------------------------------------------------------
5. Design ambiguity:                                                        0
     Duplicate module definition:                              no
     Black box due to undefined cells:                         no
     Golden design has abnormal ratio of unreachable gates:    no
     Ratio of golden unreachable gates:                        0%
     Revised design has abnormal ratio of unreachable gates:   no
     Ratio of revised unreachable gates:                       0%
     All primary input bus ordering is consistent:             yes
     All primary output bus ordering is consistent:            yes
     DFF/DLAT not compared due to disabled clock port(s):        0
--------------------------------------------------------------------------------
6. Compare Results:                                                        PASS
     Number of EQ compare points:                              180
     Number of NON-EQ compare points:                          0
     Number of Aborted compare points:                         0
     Number of Uncompared compare points :                     0
================================================================================
pass
// Command: report_statistics
Mapping and compare statistics
================================================================================
                     Compare Result       Golden            Revised
--------------------------------------------------------------------------------
Root module name                          async_fifo1       async_fifo1

Primary inputs                                14                14
   Mapped                                     14                14
```

Figure 6:LEC(Logical Equivalence Check

7. Physical design process

7.1 Floorplan

Floor-plan design is an important step in physical design of VLSI circuits to plan the positions of a set of circuit modules on a chip to optimize the circuit performance. Floor planning is the process of creating an area for macros and standard cells to be placed.

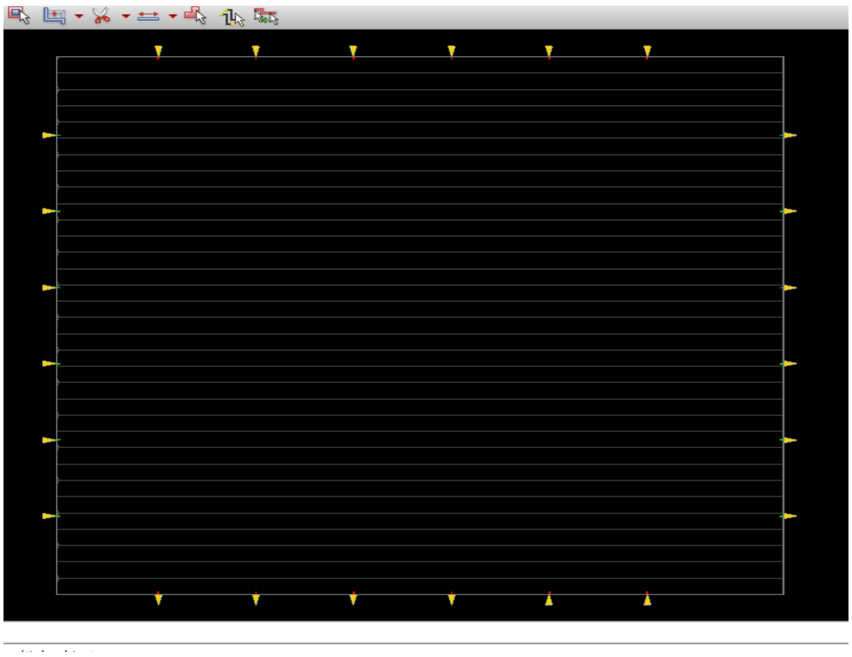


Figure 7:Floor Plan

Aspect ratio: Aspect ratio will decide the size and shape of the chip. ratio of height and width of core.

Aspect ratio = width/height

Core utilization: Utilization will define the area occupied by the standard cells, macros, and other cells. If core utilization is 0.8 (80%) that means 80% of the core area is used for placing the standard cells, macros, and other cells, and the remaining 20% is used for routing purposes.

core utilization = (macros area + std cell area)/ total core area

7.2 Power Planning

Power planning means to provide power to every macro, standard cells, and all other cells are present in the design. Power and Ground nets are usually laid out on the metal layers.

- Rings: It Carries VDD and VSS around the chip
- Stripes: It Carries VDD and VSS from Rings across the chip
- Rails: It connects VDD and VSS to the standard cell VDD and VSS.

- Trunk: The connection between Pad and Ring
- Pad: Interface from IC to the outside world.

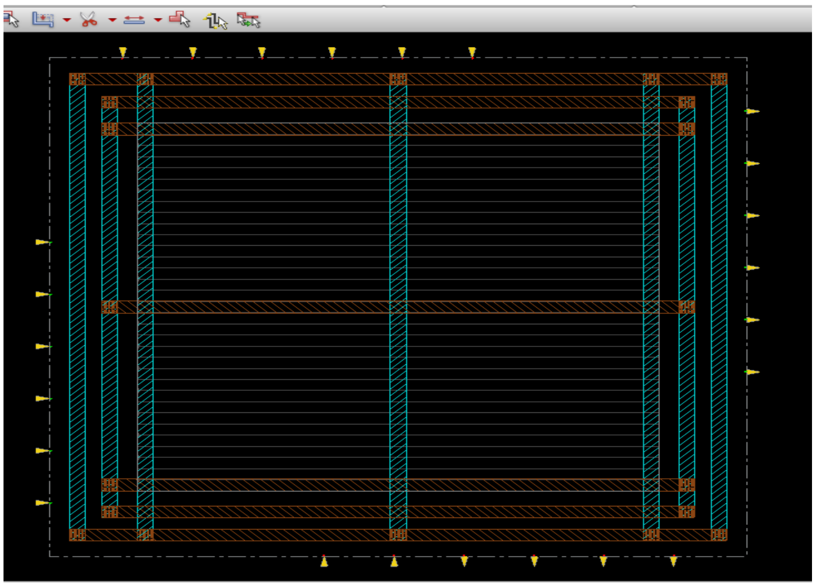


Figure 8:Power Plan

7.3 Placement

Before going into placement step special route is done to take the power and ground rails into the core area.
Placement is the process of placing the standard cells inside the core boundary in an optimal location. The tool tries to place the standard cell in such a way that the design should have minimal congestions and the best timing. Every PnR tool provides various commands/switches so that users can optimize the design in a better way in terms of timing, congestion, area, and power as per their requirements.

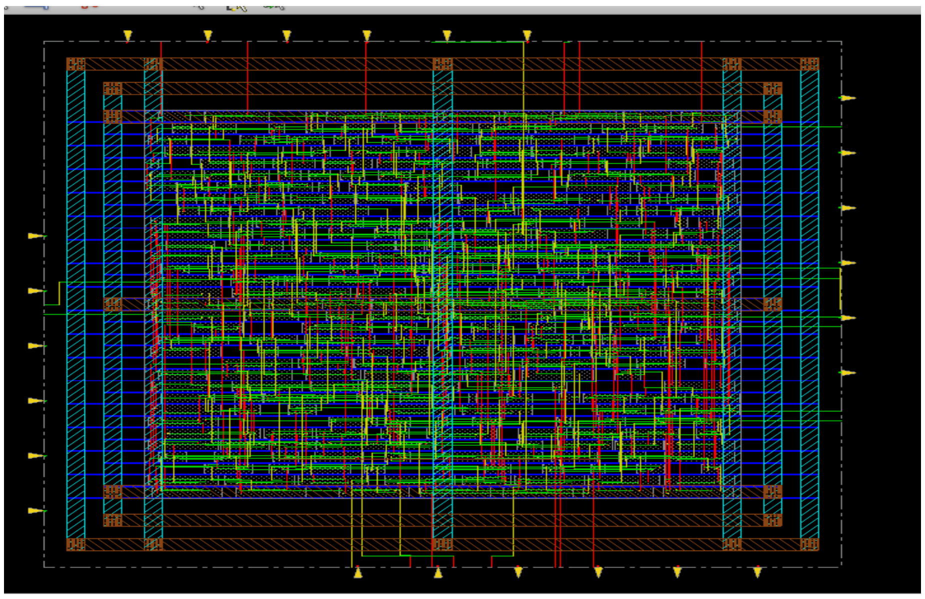


Figure 9:Placement

7.4 Clock tree synthesis

The process of distributing the clock and balancing the load is called CTS. Basically, delivering the clock to all sequential elements. CTS is the process of insertion of buffers and inverters along the clock paths of ASIC design to achieve zero or minimum skew or balanced skew.

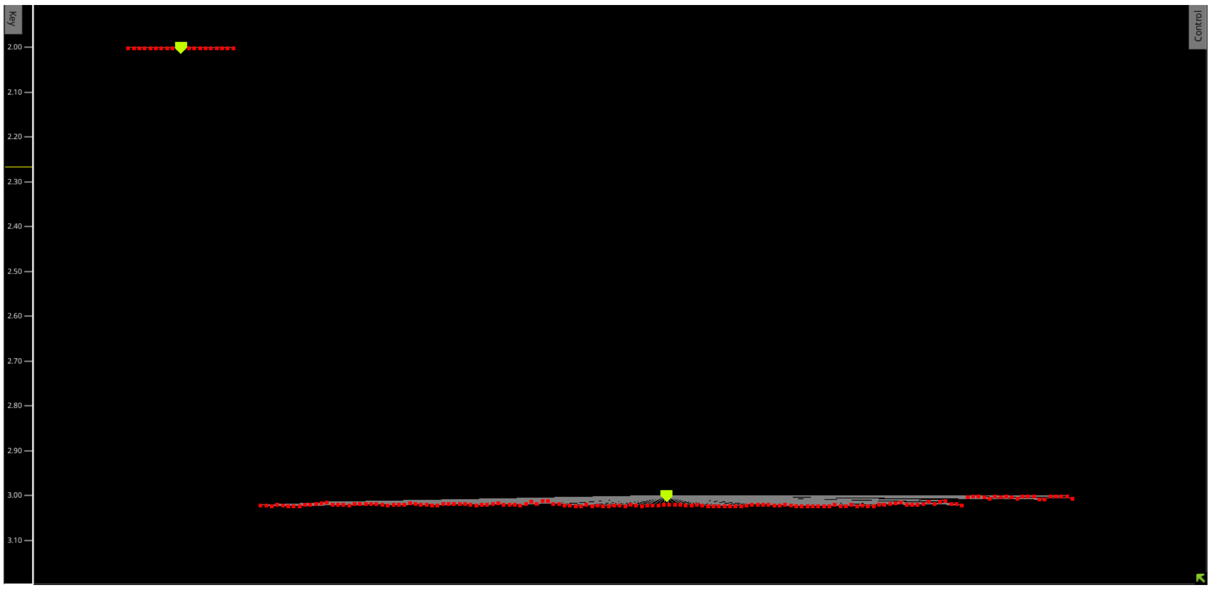


Figure 10:Clock Tree Debugger

7.5 Routing

Making physical connections between signal pins using metal layers are called routing. Routing is the stage after CTS and optimization where exact paths for the interconnection of standard cells.

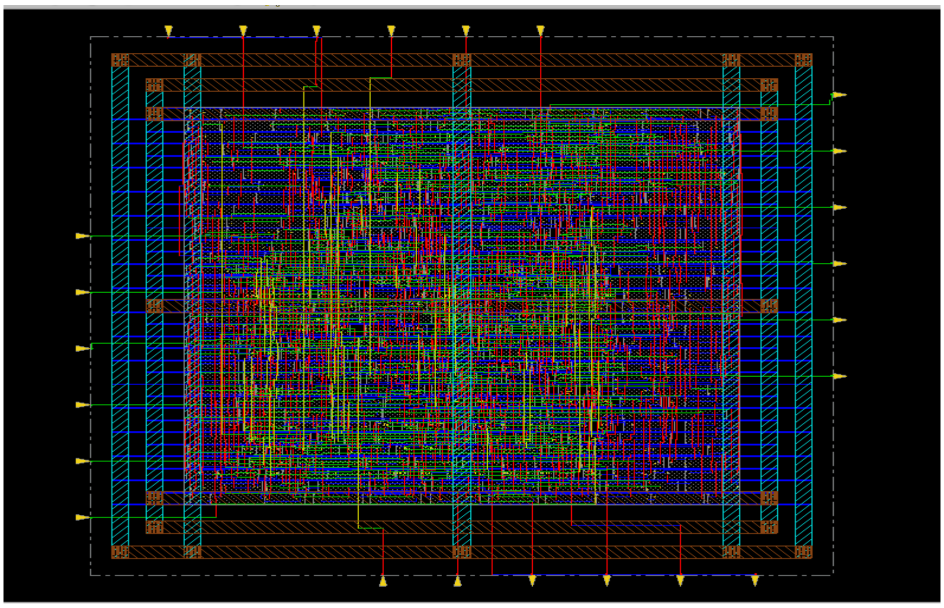
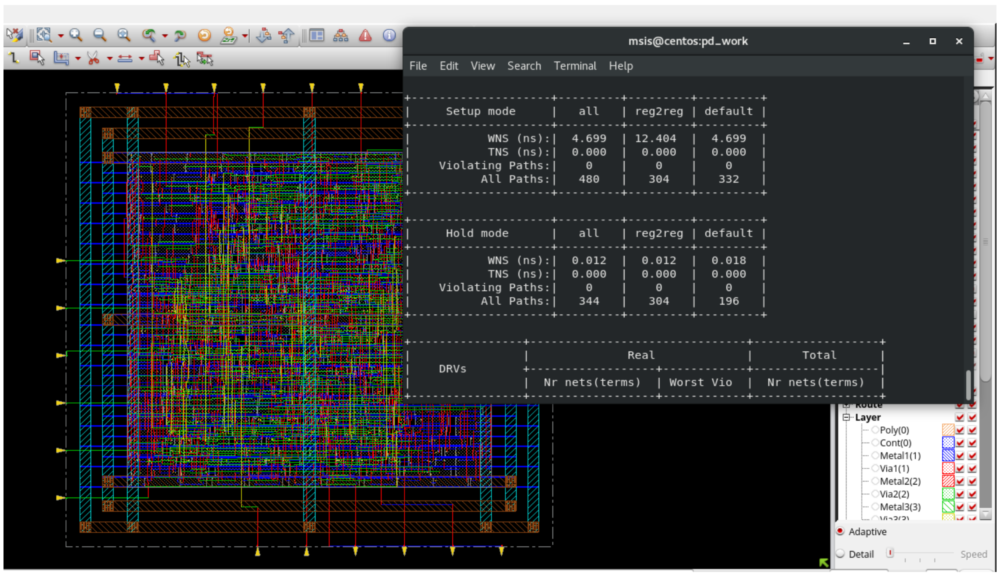


Figure 11:Routing

7.6 Post Route Optimization

In every step we can do the setup and hold time analysis by report timing in the tool. Below snapshot shows the design and timing report after optimization.

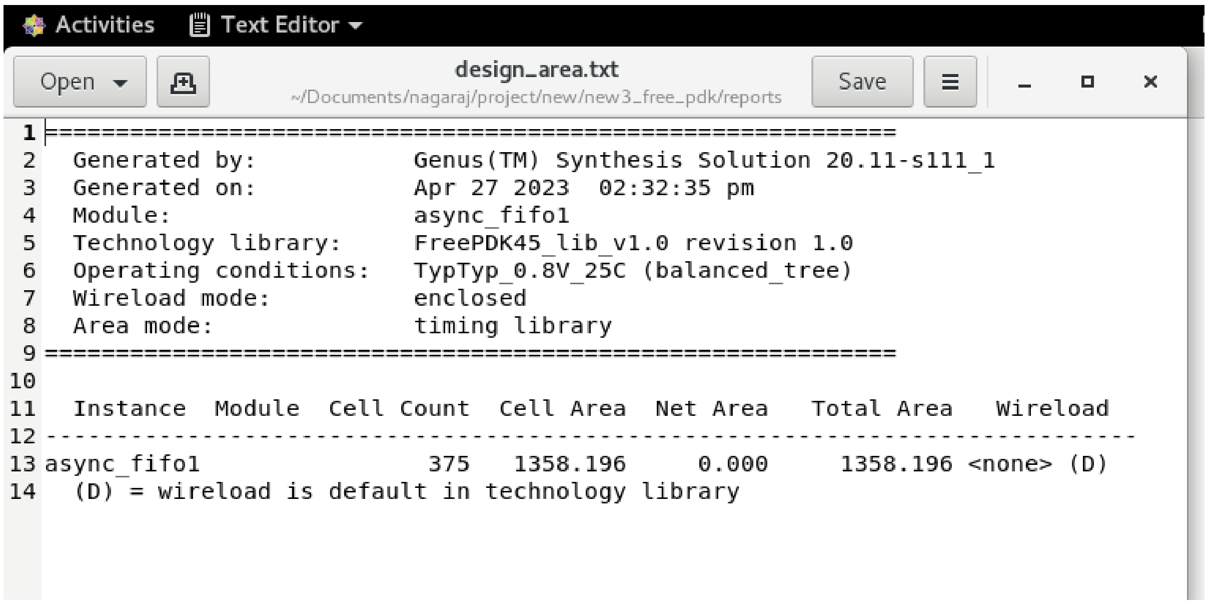


```
+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  4.699  | 12.404  |  4.699  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   480   |   304   |   332   |
+--------------------+---------+---------+---------+

+--------------------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.012  |  0.012  |  0.018  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   344   |   304   |   196   |
+--------------------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |             Real              |       Total      |
|    DRVs        +------------------+------------+------------------|
|                | Nr nets(terms)   | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
```

Figure 12:Timing Report

7.7 Reports

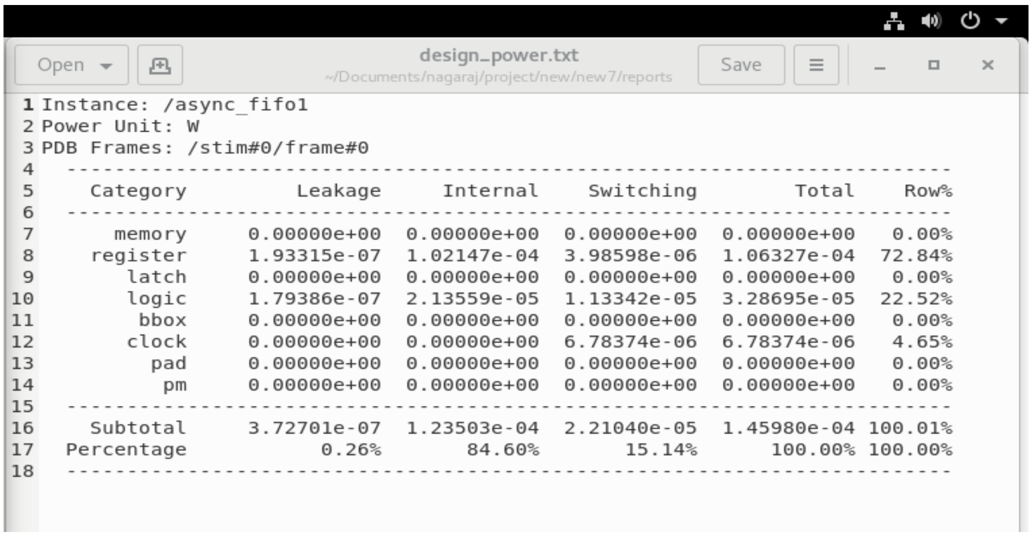
7.7.1 Area report



```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 20.11-s111_1
  Generated on:           Apr 27 2023  02:32:35 pm
  Module:                 async_fifo1
  Technology library:     FreePDK45_lib_v1.0 revision 1.0
  Operating conditions:   TypTyp_0.8V_25C (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

  Instance  Module  Cell Count  Cell Area  Net Area   Total Area   Wireload
-----------------------------------------------------------------------------
async_fifo1                375   1358.196     0.000     1358.196 <none> (D)
  (D) = wireload is default in technology library
```

Figure 13: Area Report

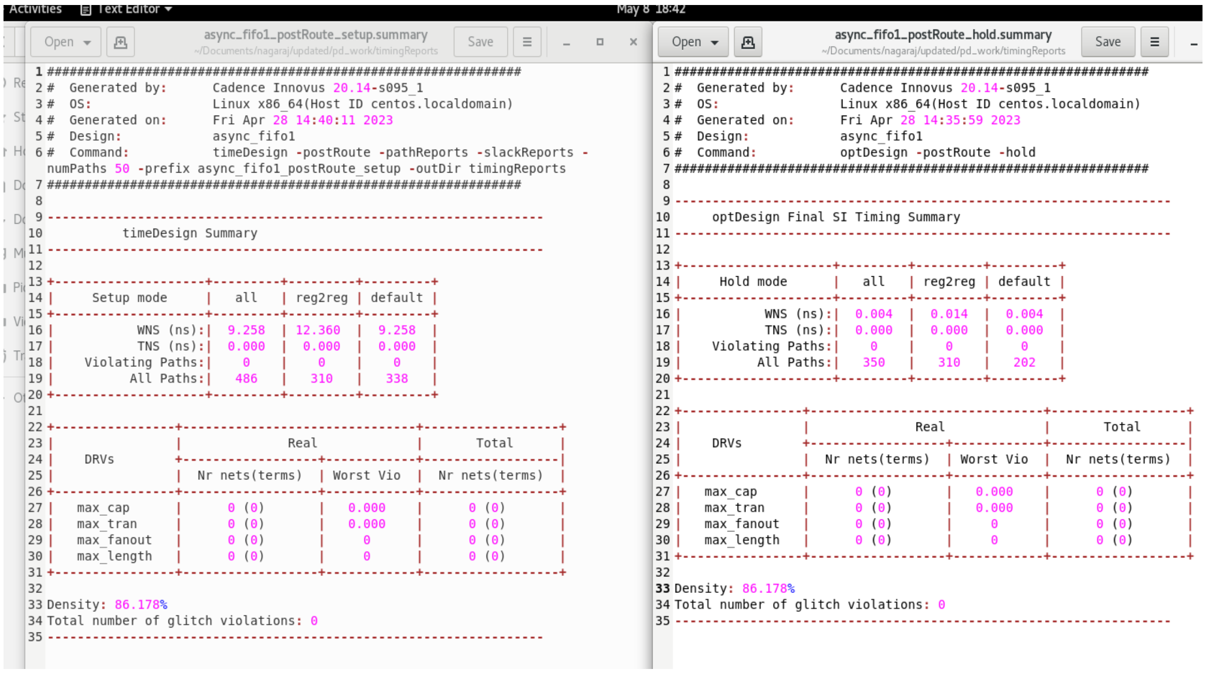
7.7.2 Power report



```
Instance: /async_fifo1
Power Unit: W
PDB Frames: /stim#0/frame#0

  -------------------------------------------------------------------------
     Category         Leakage       Internal      Switching         Total     Row%
  -------------------------------------------------------------------------
       memory     0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00    0.00%
     register     1.93315e-07   1.02147e-04   3.98598e-06   1.06327e-04   72.84%
        latch     0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00    0.00%
        logic     1.79386e-07   2.13559e-05   1.13342e-05   3.28695e-05   22.52%
         bbox     0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00    0.00%
        clock     0.00000e+00   0.00000e+00   6.78374e-06   6.78374e-06    4.65%
          pad     0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00    0.00%
           pm     0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00    0.00%
  -------------------------------------------------------------------------
     Subtotal     3.72701e-07   1.23503e-04   2.21040e-05   1.45980e-04  100.01%
   Percentage           0.26%        84.60%        15.14%       100.00%  100.00%
  -------------------------------------------------------------------------
```

Figure 14: Power Report

7.7.3 Timing report


Activities
Text Editor
May 8 18:42
Open
Save
async_fifo1_postRoute_setup.summary
~/Documents/nagaraj/updated/pd_work/timingReports
async_fifo1_postRoute_hold.summary
~/Documents/nagaraj/updated/pd_work/timingReports

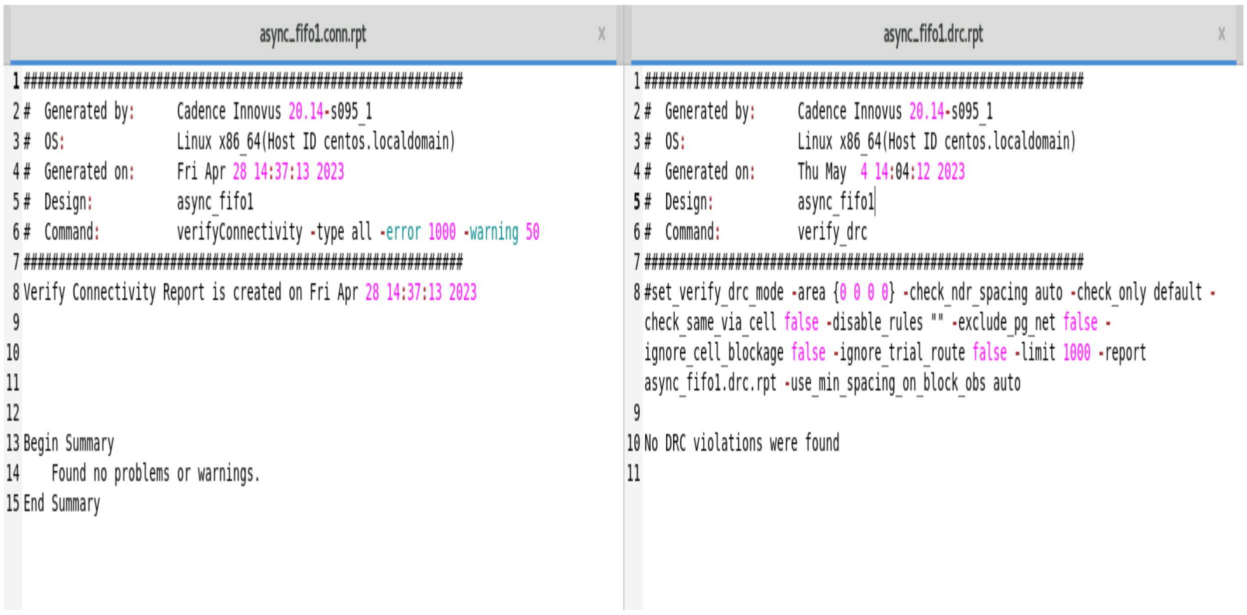
```
###############################################################
#  Generated by:      Cadence Innovus 20.14-s095_1
#  OS:                Linux x86_64(Host ID centos.localdomain)
#  Generated on:      Fri Apr 28 14:40:11 2023
#  Design:            async_fifo1
#  Command:           timeDesign -postRoute -pathReports -slackReports -
numPaths 50 -prefix async_fifo1_postRoute_setup -outDir timingReports
###############################################################

------------------------------------------------------------------
         timeDesign Summary
------------------------------------------------------------------

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  9.258  | 12.360  |  9.258  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   486   |   310   |   338   |
+--------------------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              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: 86.178%
Total number of glitch violations: 0
------------------------------------------------------------------
```

```
###############################################################
#  Generated by:      Cadence Innovus 20.14-s095_1
#  OS:                Linux x86_64(Host ID centos.localdomain)
#  Generated on:      Fri Apr 28 14:35:59 2023
#  Design:            async_fifo1
#  Command:           optDesign -postRoute -hold
###############################################################

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

+--------------------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.004  |  0.014  |  0.004  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   350   |   310   |   202   |
+--------------------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              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: 86.178%
Total number of glitch violations: 0
------------------------------------------------------------------
```

Figure 15:Timing Report

7.8 Design Rule Check (DRC)

Design Rule Check (DRC) is the process of checking physical layout data against fabrication-specific rules specified by the foundry to ensure successful fabrication. Process specific design rules must be followed when drawing layouts to avoid any manufacturing defects during the fabrication of an IC. Violating a design rule might result in a non-functional circuit or low Yield.



```
############################################################
#  Generated by:      Cadence Innovus 20.14-s095_1
#  OS:                Linux x86_64(Host ID centos.localdomain)
#  Generated on:      Fri Apr 28 14:37:13 2023
#  Design:            async_fifo1
#  Command:           verifyConnectivity -type all -error 1000 -warning 50
############################################################
Verify Connectivity Report is created on Fri Apr 28 14:37:13 2023




Begin Summary
    Found no problems or warnings.
End Summary
```

```
############################################################
#  Generated by:      Cadence Innovus 20.14-s095_1
#  OS:                Linux x86_64(Host ID centos.localdomain)
#  Generated on:      Thu May  4 14:04:12 2023
#  Design:            async_fifo1
#  Command:           verify_drc
############################################################
#set_verify_drc_mode -area {0 0 0 0} -check_ndr_spacing auto -check_only default -
check_same_via_cell false -disable_rules "" -exclude_pg_net false -
ignore_cell_blockage false -ignore_trial_route false -limit 1000 -report
async_fifo1.drc.rpt -use_min_spacing_on_block_obs auto

No DRC violations were found

```

Figure 16:DRC Report

8. Conclusions

- The main idea of our project was to have a working Asynchronous FIFO memory block that functions correctly as per its specification & intent.
- The same was achieved by designing and simulating the Asynchronous FIFO design using CADENCE TOOLS.
- This design was verified for its correct functionality by writing test bench.
- Physical design for the Asynchronous FIFO is done.

References

1) Simulation and Synthesis Techniques for Asynchronous FIFO Design Clifford E. Cummings, Sunburst Design, Inc. cliffc@sunburst-design.com
2) Constraining designs for synthesis and timing analysis, by Sridhar gangadharan and Sanjay churiwala.
3) Physical Design essentials by Khosrow Golshan.