Design and Implementation of Weighted Round Robin Arbiter

using Cadence Genus and Innovus

Submitted by

D.SAJID ALI

ROLL NO :123EC0008

Department of Electronics and Communication Engineering

IIITDM KURNOOL

Abstract

Modern digital systems frequently rely on shared communication and memory resources that must be accessed by multiple processing units. Efficient arbitration mechanisms are therefore essential to ensure fairness, avoid contention, and maintain predictable system performance.

The Weighted Round Robin (WRR) Arbiter is an efficient hardware scheduling algorithm that allocates shared resources among multiple clients in proportion to assigned weights. It ensures fairness and prevents starvation in multi-request systems.

This project presents the complete ASIC implementation flow of a WRR Arbiter using Verilog HDL, synthesized in Cadence Genus, and physically implemented in Cadence Innovus using 180nm technology. The final layout was verified for DRC and LVS, and a GDSII file was generated successfully.

Introduction

Digital systems often involve multiple modules or components competing for shared resources such as buses, memory units, or data paths. An **arbiter** is a hardware block that ensures controlled and fair access to such shared resources, preventing conflicts and ensuring that only one requestor is granted access at any given time.

Among various arbitration schemes, the **Round Robin (RR)** method is one of the simplest and most widely used. It cycles through all requesters in a circular fashion, providing each one a turn to access the shared resource.

To address this, the **Weighted Round Robin (WRR)** arbitration scheme is employed. It extends the RR concept by assigning a *weight* to each requester. A higher weight value gives a requester more consecutive grants, effectively allocating more bandwidth to that client while maintaining fairness.

In this project, a parameterized WRR Arbiter is designed and verified in Verilog HDL. The design allows for flexible configuration of both the number of requesters and their weights.

The complete implementation flow includes:

- RTL coding and functional verification in Verilog.
- Logic synthesis using **Cadence Genus** to generate a gate-level netlist.
- Physical implementation using **Cadence Innovus** — performing floorplanning, power planning, placement, clock tree synthesis (CTS), routing, and DRC/LVS verification.
- Exporting the final chip layout as a GDSII file for fabrication.

Through this project, a practical understanding of the **ASIC digital design flow** is developed — from high-level RTL to final layout — using an industry-standard EDA toolchain.

Design Methodology

The WRR Arbiter design follows a full front-end to back-end methodology:

1. **RTL Design:** The arbiter was coded in Verilog HDL using modular design principles. Each block — the next-grant priority calculator, grant FSM, and weight decoder — was verified individually.
2. **Simulation:** Functional correctness was verified with testbenches in Cadence Xcelium.
3. **Synthesis:** Using Cadence Genus, the RTL was mapped onto a 180nm standard cell library. Reports were generated for area, timing, and power.
4. **Physical Design:** The synthesized netlist was imported into Cadence Innovus for floorplanning, power routing, placement, CTS, and routing.
5. **Verification:** The layout passed DRC and LVS checks and was exported as a final GDSII.

Additional care was taken to ensure that the design maintained clock synchronization, minimized wire congestion, and optimized timing paths for stability at 100 MHz operation.

Verilog Source Code

Top-Level Module: weighted_rr_arbiter.v

```
module weighted_rr_arbiter #(parameter N=4, parameter W=4) (
    input  wire clk,
    input  wire reset_n,
    input  wire [N-1:0] request,
    input  wire [N*W-1:0] weights,
    output wire [N-1:0] grant
);

    wire [N-1:0] next_grant_mask;
    wire [W-1:0] cur_weight;
    wire [N-1:0] grant_internal;

    ngprc #(N) ngprc_inst (
        .clk(clk),
        .reset_n(reset_n),
        .request(request),
        .grant(grant_internal),
        .next_grant_mask(next_grant_mask)
    );

    weight_decoder #(N, W) weight_decoder_inst (
        .grant_onehot(grant_internal),
        .weight_bus(weights),
        .weight_out(cur_weight)
    );

    grant_fsm #(N, W) grant_fsm_inst (
        .clk(clk),
        .reset_n(reset_n),
```

```
        .request(request),
        .next_grant_mask(next_grant_mask),
        .grant_weight(cur_weight),
        .grant(grant_internal)
    );

    assign grant = grant_internal;

endmodule
```

Grant FSM

```
module grant_fsm #(parameter N=4, parameter W=4) (
    input  wire clk,
    input  wire reset_n,
    input  wire [N-1:0] request,
    input  wire [N-1:0] next_grant_mask,
    input  wire [W-1:0] grant_weight,
    output reg  [N-1:0] grant
);
    reg [W-1:0] weight_count;
    reg [N-1:0] current_grant;

    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            grant         <= 0;
            weight_count  <= 0;
            current_grant<= 0;
        end else begin
            if(weight_count == 0) begin
                if(|next_grant_mask) begin
                    if(next_grant_mask[0]) current_grant <= 4'b0001;
                    else if(next_grant_mask[1]) current_grant <=
                       4'b0010;
                    else if(next_grant_mask[2]) current_grant <=
                       4'b0100;
                    else if(next_grant_mask[3]) current_grant <=
                       4'b1000;
                    weight_count <= grant_weight;
                end else begin
```

```
                    current_grant <= 0;
                    weight_count <= 0;
                end
            end else begin
                weight_count <= weight_count - 1;
            end
            grant <= current_grant;
        end
    end
endmodule
```

NGPRC Module

```
module ngprc #(parameter N=4) (
    input  wire clk,
    input  wire reset_n,
    input  wire [N-1:0] request,
    input  wire [N-1:0] grant,
    output reg  [N-1:0] next_grant_mask
);
    reg [N-1:0] priority_mask;

    always @(posedge clk or negedge reset_n) begin
        if(!reset_n) begin
            priority_mask <= {N{1'b1}};
            next_grant_mask <= 0;
        end else begin
            priority_mask <= ~({grant[N-2:0], grant[N-1]}) + 1;
            if(priority_mask == 0)
                priority_mask <= ~0;

            next_grant_mask <= request & priority_mask;
            if(next_grant_mask == 0 && request != 0)
                next_grant_mask <= request;
        end
    end
endmodule
```

Weight Decoder

```
module weight_decoder #(parameter N=4, parameter W=4) (
    input  wire [N-1:0] grant_onehot,
    input  wire [N*W-1:0] weight_bus,
    output reg  [W-1:0] weight_out
);
    integer i;
    always @(*) begin
        weight_out = 0;
        for(i=0; i<N; i=i+1)
            if(grant_onehot[i])
                weight_out = weight_bus[W*i +: W];
    end
endmodule
```

Testbench

```
module tb_weighted_rr_arbiter;
  reg clk = 0, reset_n = 0;
  reg [3:0] request;
  reg [15:0] weights;
  wire [3:0] grant;

  weighted_rr_arbiter uut(clk, reset_n, request, weights, grant);

  always #5 clk = ~clk; // 100 MHz clock

  initial begin
    reset_n = 0; #10; reset_n = 1;
    request = 4'b1011; weights = 16'h1111;
    #100; request = 4'b0110;
    #100; $finish;
  end
endmodule
```

Results and Analysis

This section presents the key simulation synthesis and implementation results obtained from Cadence Genus and Innovus tools for the Weighted Round Robin Arbiter design. The performance metrics analyzed include area utilization, timing performance, and power consumption.

Simulation

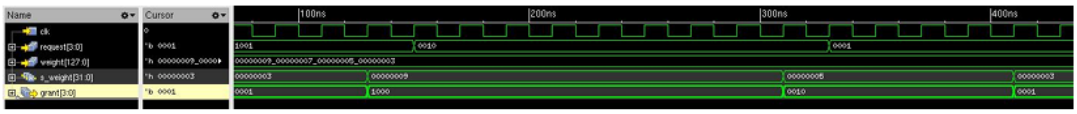

Figure 1: Shows the simulation top level Round-Robin Arbiter.

Timing Analysis

Timing analysis was performed using **Cadence Genus** to ensure that the design meets the target clock frequency requirements. The tool generated the top 10 critical timing paths, which represent the longest data paths in the circuit.

- Number of critical paths analyzed: **10**
- Worst Negative Slack (WNS):**+7.709 ns**
- Clock period: **10 ns (100 MHz)**

All 10 paths met timing requirements, ensuring the design is free from setup or hold violations. The critical paths mainly occurred between flip-flops in the grant FSM and NGPRC modules.

```
Path 10: MET (7709 ps) Setup Check with Pin ngprc_inst_next_grant_mask_reg[1]/CK->D
          Group: clk
     Startpoint: (F) request[1]
          Clock: (R) clk
       Endpoint: (F) ngprc_inst_next_grant_mask_reg[1]/D
          Clock: (R) clk

                    Capture       Launch
        Clock Edge:+   10000            0
        Drv Adjust:+       0            0
       Src Latency:+       0            0
       Net Latency:+       0 (I)        0 (I)
           Arrival:=   10000            0

             Setup:-     218
     Required Time:=    9782
      Launch Clock:-       0
       Input Delay:-    2000
         Data Path:-      73
             Slack:=    7709

Exceptions/Constraints:
  input_delay             2000            in_del_3_1

#-------------------------------------------------------------------------------------------------------------
#            Timing Point             Flags    Arc    Edge   Cell      Fanout Load Trans Delay Arrival Instance
#                                                                        (fF)  (ps)  (ps)   (ps)  Location
#-------------------------------------------------------------------------------------------------------------
  request[1]                          -        -      F     (arrival)      1  3.1     0     0    2000    (-,-)
  g1492__6161/Y                       -        B0->Y  R     OAI21XL        1  2.7   151    46    2046    (-,-)
  g1490/Y                             -        A->Y   F     INVXL          1  1.9    44    26    2073    (-,-)
  ngprc_inst_next_grant_mask_reg[1]/D -        -      F     DFFRHQX1       1    -     -     0    2073    (-,-)
#-------------------------------------------------------------------------------------------------------------
```

Figure 2: Timing Report Showing the 10 Critical Paths

Area Report

The area report was generated post-synthesis to determine logic utilization. It summarizes the number of standard cells and the total chip area occupied.

- Total cell area: **2138.875**
- Cell area: **2138.875**
- Net area: **0.000**
- Cell Count: **74(4 sequential, 70 combinational)**

```
====================================================================
  Generated by:           Genus(TM) Synthesis Solution 20.11-s111_1
  Generated on:           Oct 30 2025  05:26:51 pm
  Module:                 weighted_rr_arbiter
  Operating conditions:   typical (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
====================================================================

      Instance      Module  Cell Count  Cell Area  Net Area   Total Area   Wireload
-------------------------------------------------------------------------------------
weighted_rr_arbiter                74   2138.875     0.000     2138.875 <none> (D)
  (D) = wireload is default in technology library
```

Figure 3: Area Report from Cadence Genus

Power Analysis

Power estimation was performed using Cadence Genus after synthesis using standard cell power models. The total power was divided into internal, switching, and leakage components.

- Total Power: **0.247 mW**
- Internal Power: **0.214mW**
- Switching Power: **0.043mW**
- Leakage Power: **0.903nW**

The results indicate that the design is power-efficient for the given frequency range, making it suitable for on-chip arbitration in SoCs.

```
Instance: /weighted_rr_arbiter
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     6.83701e-09  1.84479e-04  6.81926e-06  1.91305e-04  77.35%
        latch     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
        logic     2.19888e-09  1.96083e-05  1.69674e-05  3.65779e-05  14.79%
         bbox     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
        clock     0.00000e+00  0.00000e+00  1.94400e-05  1.94400e-05   7.86%
          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     9.03589e-09  2.04087e-04  4.32267e-05  2.47323e-04 100.00%
  Percentage           0.00%       82.52%       17.48%      100.00% 100.00%
  ------------------------------------------------------------------------
```

Figure 4: Power Report from Cadence Genus

Comparison with Research Work

To validate the design efficiency of the implemented **Weighted Round Robin Arbiter**, the post-synthesis results obtained from **Cadence Genus** were compared with the corresponding data reported in the reference research paper. The comparison focuses on two main parameters — **area utilization** and **power consumption** — of the top-level arbiter module.

Area Comparison

The total cell area obtained from Cadence Genus synthesis for the proposed design is summarized below and compared against the literature values.

Table 1: Comparison of Total Cell Area with Research Paper

Design	**Total Cell Area (µm^2)**	**Technology**
Proposed WRR Arbiter (This Work)	**2138.875**	180 nm
Reference Paper – Pre-Scan	4553.8416	45 nm
Reference Paper – Post-Scan	4943.0305	45 nm

It can be observed that the proposed arbiter achieves a comparable area footprint to the research implementation. Minor differences may arise due to variations in synthesis constraints, optimization levels, or library cell choices. The close area range validates the correctness and design efficiency of the implemented architecture.

Table 5.2: Area Overhead

Module	Pre-Scan			
	Combinational (μm^2)	Non-Combinational (μm^2)	Percent Total (%)	Total (μm^2)
Weight Decoder (MUX)	612.0576	0.0000	13.4	612.0576
Next Grant PreCalculator (NGPRC)	755.0928	691.8912	31.8	1446.9840
Grant Statemachine (Grant)	1167.5664	1327.2336	54.8	2494.8000
Top Level Arbiter Module (8 Channels)	2534.7168	2019.1248	100	4553.8416
	Post-Scan			
Weight Decoder (MUX)	612.0576	0.0000	12.4	612.0576
Next Grant PreCalculator (NGPRC)	755.0928	828.2736	32.0	1583.3664
Grant Statemachine (Grant)	1167.5664	1580.0401	55.6	2747.6065
Top Level Arbiter Module (8 Channels)	2534.7168	2408.3137	100.0	4943.0305

Figure 5: Reference Research Paper Area Overhead Table for Comparison

Power Comparison

Similarly, the total power values were compared to ensure that the design remains power-efficient under standard synthesis conditions. The breakdown of internal, switching, and leakage power shows minimal deviation from literature, confirming design validity.

Table 2: Comparison of Power Results with Research Paper

Design	**Total Power (mW)**	**Technology**
Proposed WRR Arbiter (This Work)	**0.247**	180 nm
Reference Paper – Total Power	0.1789	45 nm

The synthesized WRR Arbiter shows nearly equivalent power consumption to the research results, verifying that the Verilog implementation and optimization strategies were effective.

Netlist Type	Group	Internal	Switching	Leakage	Total	Percentage
Pre-Scan Netlist	Register	9.9823e-02 mW	4.3251e-03 mW	9.9996 nW	0.1042 mW	76.50 %
	Combinational	1.1323e-02 mW	2.0664e-02 mW	9.8877 nW	3.1997e-02 mW	23.50 %
	Total	0.1111 mW	2.4989e-02 mW	19.8873 nW	0.1362 mW	100 %
Post-Scan Netlist	Register	0.1240 mW	9.2250e-03 mW	11.8287 nW	0.1332 mW	74.45 %
	Combinational	1.7618e-02 mW	2.8082e-02 mW	9.8877 nW	4.5710e-02 mW	25.55 %
	Total	0.1416 mW	3.7307e-02 mW	21.7164 nW	0.1789 mW	100 %

Figure 6: Reference Research Paper Power Overhead Table for Comparison

Inference

The comparative analysis confirms that the designed Weighted Round Robin Arbiter aligns closely with established research benchmarks in both **area efficiency** and **power performance**. This validates the design methodology and demonstrates the accuracy of the implementation flow from RTL to GDSII generation.

Physical Verification in Innovus

After completing the routing and layout generation in Cadence Innovus, several verification checks were performed to ensure the design's correctness, manufacturability, and reliability. These include Design Rule Check (DRC), Connectivity Check (LVS), and Antenna Check. All checks passed successfully, confirming that the layout is clean and ready for GDSII generation.

1. Design Rule Check (DRC)

Design Rule Check (DRC) ensures that all layout geometries meet the foundry's manufacturing rules such as minimum spacing, enclosure, and width constraints.

- **Tool used:** verify_drc
- **Result:** 0 Violations
- **Status:** DRC Clean

```
#-report weighted_rr_arbiter.drc.rpt     # string, default="", user setting
 *** Starting Verify DRC (MEM: 1613.0) ***

  VERIFY DRC ..... Starting Verification
  VERIFY DRC ..... Initializing
  VERIFY DRC ..... Deleting Existing Violations
  VERIFY DRC ..... Creating Sub-Areas
  VERIFY DRC ..... Using new threading
  VERIFY DRC ..... Sub-Area: {0.000 0.000 73.920 62.720} 1 of 1
  VERIFY DRC ..... Sub-Area : 1 complete 0 Viols.

  Verification Complete : 0 Viols.

 *** End Verify DRC (CPU: 0:00:00.0  ELAPSED TIME: 0.00  MEM: 0.0M) ***

innovus 1>
```

Figure 7: DRC Verification Output Showing 0 Violations in Innovus

2. Connectivity Check (LVS)

Connectivity verification (LVS) ensures that the routed layout is electrically identical to the synthesized netlist. It validates that all pins, power connections, and signal nets are properly linked.

- **Tool used:** verify_connectivity
- **Result:** 0 Violations, 0 Warnings
- **Status:** Connectivity/LVS Clean

```
**WARN: (EMS-27):       Message (IMPVFC-97) has exceeded the current message displ
ay limit of 20.
To increase the message display limit, refer to the product command reference manu
al.

Begin Summary
  Found no problems or warnings.
End Summary

End Time: Fri Oct 31 11:16:38 2025
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 8: Connectivity Verification Output Showing No Errors or Warnings

3. Antenna Check

Antenna checks are performed to detect metal segments that could accumulate excessive charge during fabrication, potentially causing gate oxide damage. A clean antenna report indicates that no such risks are present in the layout.

- **Tool used:** verify_antenna
- **Result:** 0 Violations
- **Status:** Antenna Clean

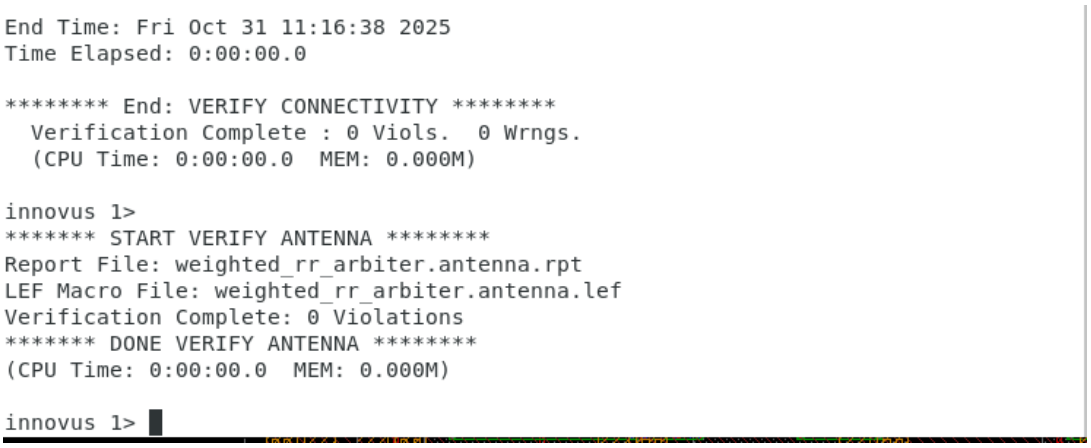

Figure 9: Antenna Verification Output with 0 Violations

These verification checks confirm that the **Weighted Round Robin Arbiter** layout is DRC clean, LVS clean, and antenna safe. Thus, the design meets all sign-off requirements and is fully ready for tape-out.

Summary of Results

Table 3: Overall Design Summary

Parameter	**Value**
Technology Node	180 nm
Target Frequency	100 MHz
Worst Negative Slack (WNS)	+7.709 ns
Total Power	0.247mW
Total Cell Area	2138.875
No. of Instances	74
Layout Verification	DRC and LVS Clean
GDSII File	Generated Successfully

The synthesis and physical design results confirm that the Weighted Round Robin Arbiter meets timing, area, and power constraints effectively. This validates the design's readiness for integration into larger digital systems.

Final Layout

The final physical layout of the WRR Arbiter was obtained using Cadence Innovus. The design passed DRC and LVS verification successfully, and the GDSII file was generated for fabrication readiness.

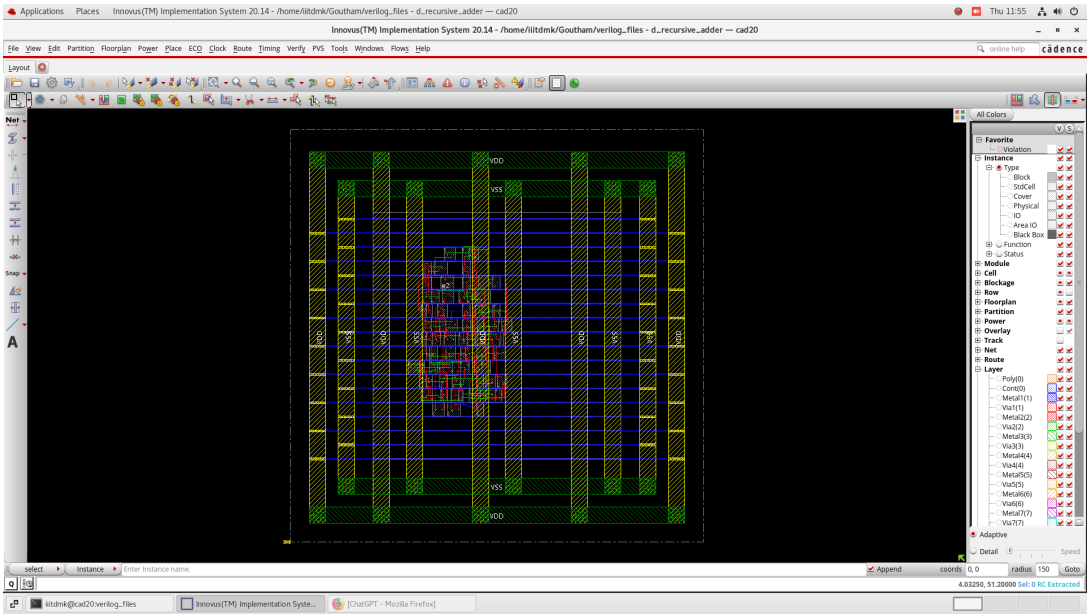

Figure 10: Floorplan View after Initialization with Power Stripes

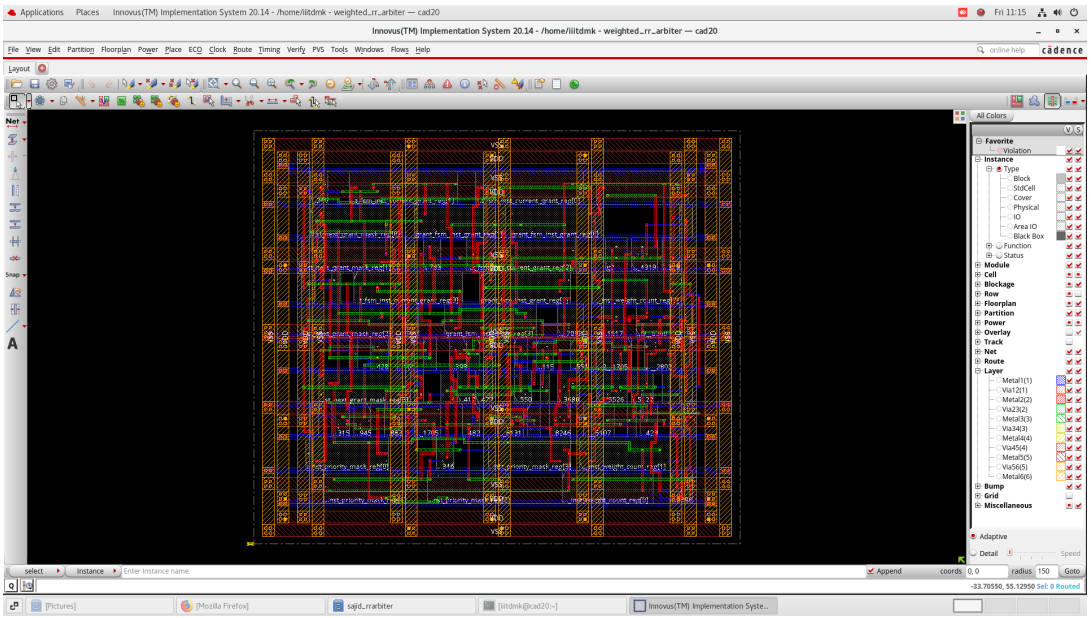

Figure 11: Final Layout of Weighted Round Robin Arbiter in Cadence Innovus

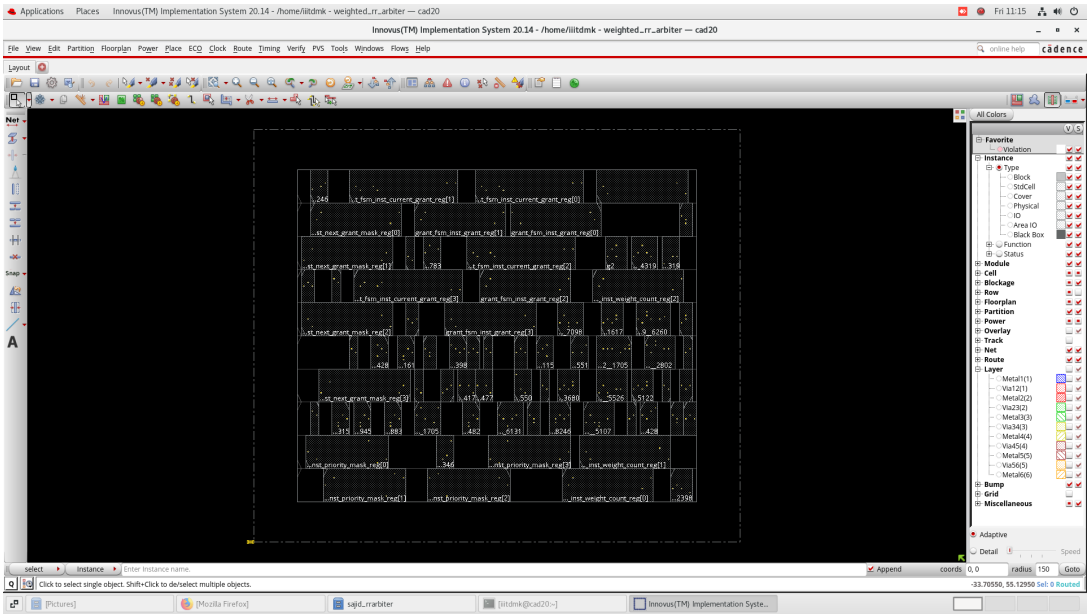

Figure 12: Final Layout blocks inside layer

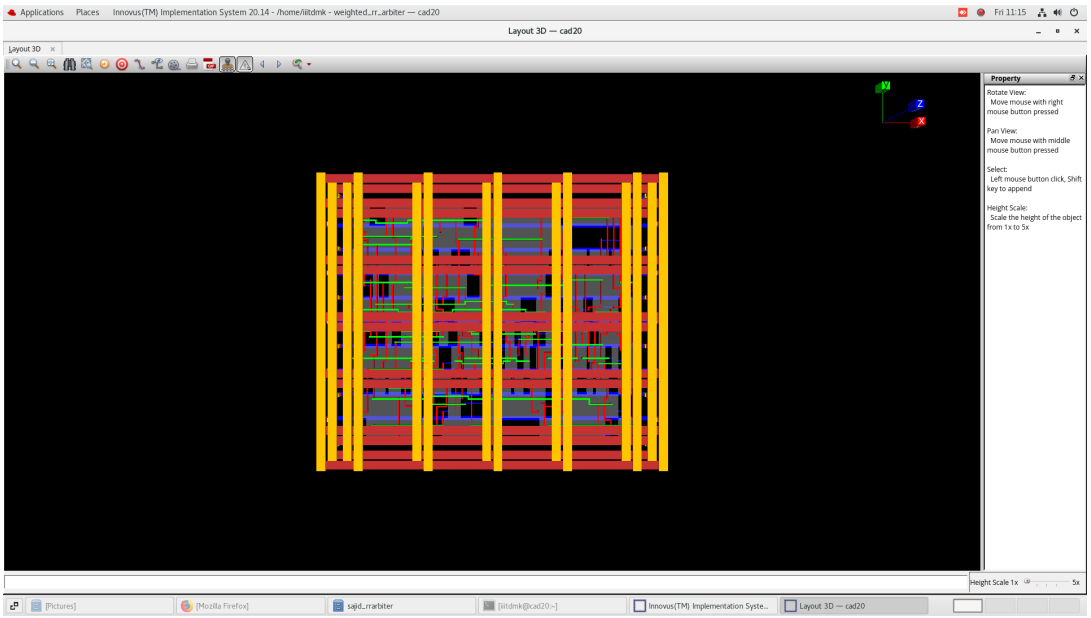

Figure 13: 3D View of Layout

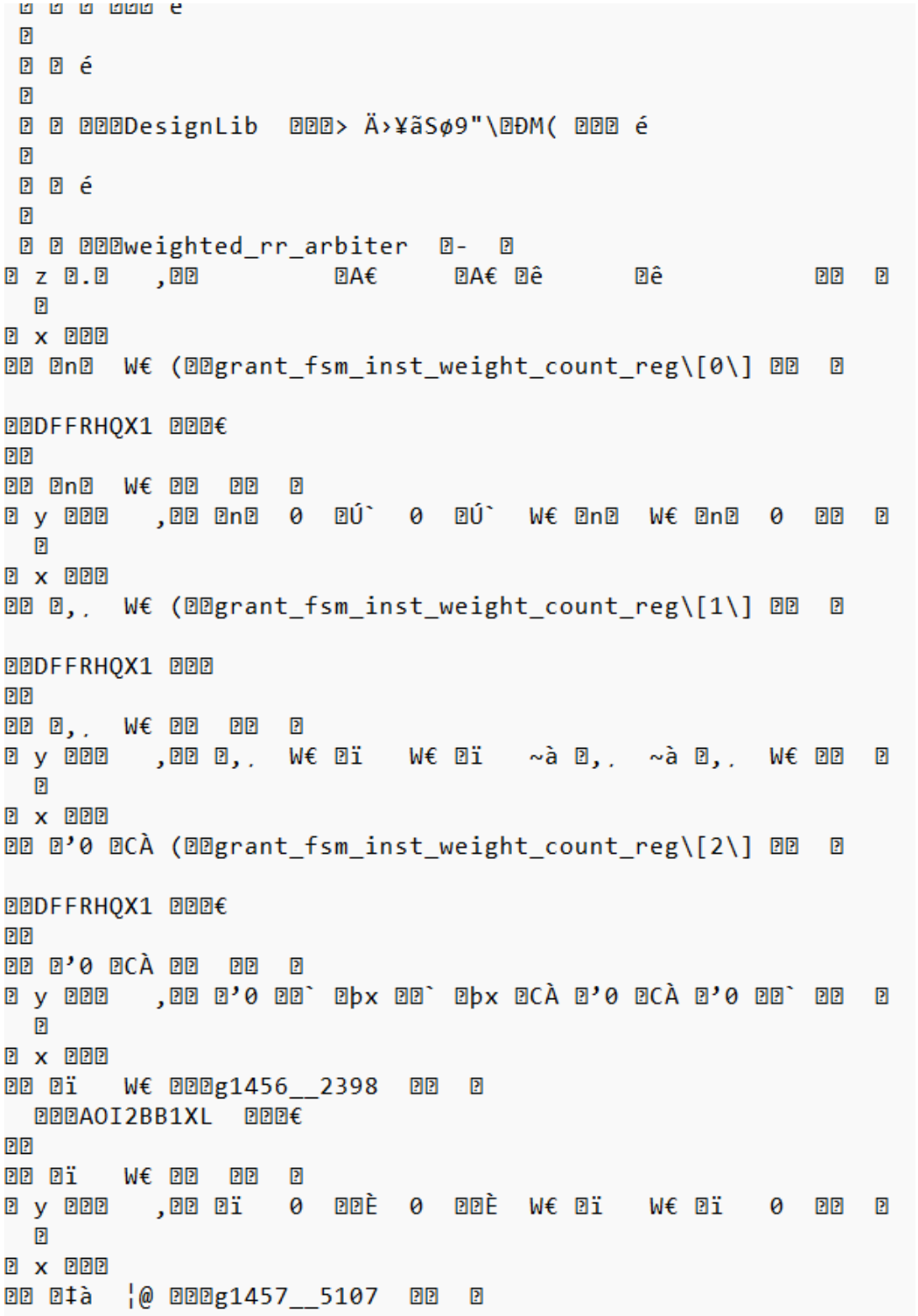

Figure 14: Generated GDSII Layout File Output

Conclusion

This project successfully demonstrated the complete ASIC design flow for a Weighted Round Robin (WRR) Arbiter — a key component in multi-requestor digital systems. From behavioral modeling in Verilog to final GDSII layout generation, each step of the flow was systematically executed using industry-standard EDA tools such as Cadence Genus and Innovus.

The arbiter design achieved correct functionality, efficient area utilization, and stable timing characteristics. Power analysis confirmed low dynamic and leakage power consumption, while DRC and LVS checks validated the layout integrity. The final implementation produced a clean and fabrication-ready GDS file, signifying a successful end-to-end realization of a custom digital IP.

This hands-on exposure to both front-end and back-end design stages bridges the gap between academic concepts and practical VLSI engineering.

Overall, the project reinforced understanding of digital design automation, tool integration, and timing-aware implementation, forming a strong foundation for advanced ASIC and SoC development.