VLSI System Design Practise Project Report

Subject Code : EC - 307

Faculty : Dr. P. Ranga Babu

Semicustom Design Flow : Scalable High-Performance Priority Encoder Design Using 1D Array to 2D Array Conversion

submitted by :

Name : Vikramaditya Singh
Roll Number : **123EC0042**
Email: 123ec0042@iiitk.ac.in
Bachelor of Technology

Name : Sohan Maity
Roll Number : **523EC0001**
Email: 523ec0001@iiitk.ac.in
BTech + MTech (Dual Degree)

Electronics and Communications Engineering
Indian Institute of Information Technology Design and Manufacturing, Kurnool

Submission Date : November 3 rd , 2025

Contents

Scalable High-Performance Priority Encoder Design Using 1D Array to 2D Array Conversion

1 Abstract

Priority Encoders (PEs) are critical components in digital systems for applications like processor interrupt handling and data arbitration, but traditional flat architectures face significant performance degradation and scalability challenges as input bit-width increases. This project presents the design, implementation, and comprehensive comparative analysis of multiple 64-bit Priority Encoder architectures to evaluate their performance across different CMOS technology nodes. The primary design implements a scalable high-performance PE based on the 1D-to-2D array conversion method, which transforms a 64-bit linear input vector into a 16×4 matrix. This architecture, implemented as pe64_lookahead, enables parallel processing of row and column priority detection to significantly reduce the critical path delay. For a thorough benchmark, this scalable design is compared against two alternative architectures: a traditional "unscalable" PE implemented with a flat if-else priority chain (pe64_if_else) and a standard pe64_standard design synthesized with explicit $4:1$ multiplexer logic.

All architectures were described in Verilog HDL and subjected to a complete semi-custom ASIC design flow (RTL-to-GDSII) using the Cadence EDA tool suite. The methodology included functional verification with NCLaunch, logic synthesis and optimization with Genus, and physical implementation with Innovus. The comparative analysis was conducted across both 180nm and 90nm technology nodes to assess the impact of process scaling on each design's performance. The post-layout results were meticulously analyzed for key metrics, including maximum operating frequency (timing), power consumption, and total cell area. The analysis confirms that the scalable 1D-to-2D array architecture (pe64_lookahead) achieves significantly higher operating frequencies compared to the unscalable and MUX-based variants. As anticipated, the 90nm implementation of the scalable PE yielded the most optimized results, validating the effectiveness of the 1D-to-2D conversion technique for designing high-performance, large-scale priority encoders.

2 Introduction

The Priority Encoder (PE) is a fundamental combinational logic circuit in modern digital systems. Its primary function is to receive multiple binary inputs and produce the binary representation of the highest-priority input that is currently active. This capability is critical in a wide range of applications, including interrupt handling in microprocessors, data arbitration in bus systems, network packet classification, and address lookup in Ternary Content Addressable Memory (TCAM).

Despite their utility, traditional PE architectures face a significant scalability challenge: as the number of input bits (L) increases, the critical path delay grows rapidly, leading to a sharp deterioration in performance. A simple 64-bit PE implemented as a flat `if-else` chain, for example, creates a long, serial priority check that severely limits the maximum operating frequency.

To overcome this limitation, this project implements and analyzes a scalable, high-performance priority encoder architecture based on the **1D-to-2D array conversion method**. This innovative technique transforms a linear L-bit input vector into a two-dimensional $M \times N$ matrix. Following this conversion, priority detection is performed in parallel: one PE block finds the highest-priority row (row index), while another block finds the highest-priority column (column index) within that active row. This hierarchical and parallel approach, particularly when combined with a look-ahead signal, effectively breaks the long critical path, enabling significantly higher operating frequencies and providing a clear path for scaling to very large input sizes.
The core objective of this work is to conduct a comprehensive comparative analysis of 64-bit PE designs. We implement three distinct architectures:

1. **Scalable PE (`pe64_lookahead`):** The high-performance design based on the 1D-to-2D conversion (16×4 matrix) and look-ahead logic.
2. **Unscalable PE (`pe64_if_else`):** A traditional flat PE using a 64-input `if-else` priority chain, serving as a baseline for performance.
3. **MUX-based PE (`pe64_standard`):** A standard scalable design using explicit $4:1$ multiplexer logic instead of the `casex` statement, to compare synthesis-driven structural differences.

These architectures are implemented in Verilog HDL and subjected to a complete **semi-custom ASIC design flow (RTL-to-GDS)** using the Cadence EDA tool suite. Functional verification is performed using NCLaunch, followed by logic synthesis and optimization with Genus, and finally physical design (place and route) with Innovus.

Furthermore, this analysis investigates the impact of process scaling by implementing the designs on both **180nm** and **90nm** CMOS technology nodes. By meticulously analyzing the post-layout results for timing (max frequency), power consumption, and total cell area, this project aims to quantify the performance gains of the 1D-to-2D conversion technique and validate its superiority for modern, high-speed digital applications.

3 Previous Work

Traditional Priority Encoder (PE) architectures have been implemented using various methods, but many suffer from high latency and poor scalability, especially as the input size (L) grows. A conventional PE64, as shown in Figure 1(a), is often built with a set of 8-bit prioritizer (PRI) modules connected in series. In this architecture, the control signal cascades from one PRI to the next, resulting in a worst-case latency that is directly proportional to the number of blocks (e.g., eight times the delay of a single PRI).
Several approaches, all referenced in the base IEEE paper, have been proposed to mitigate this performance bottleneck:

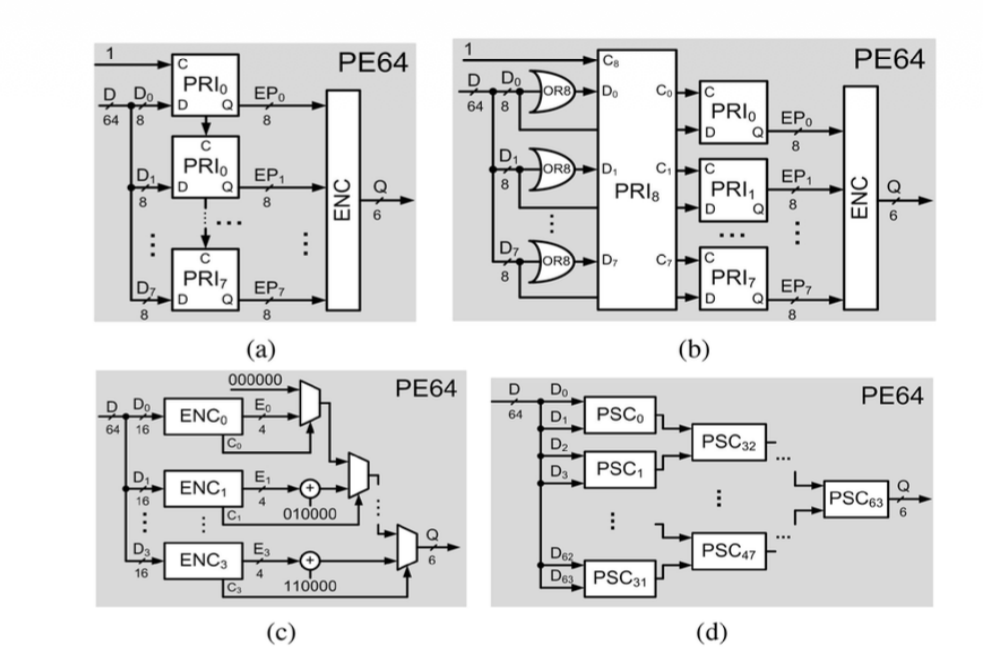


Fig. 1. The architecture of (a) conventional PE64, (b) parallel PE64, (c) PE64-based one-hot encoder, and (d) PE64-based comparison and sort circuit.

Figure 1: Architectures of various 64-bit PEs discussed in previous work. (a) conventional, (b) parallel, (c) one-hot encoder based, (d) comparison and sort circuit based.

1. **Look-ahead and Folding:** Huang et al. presented multi-level lookahead and folding techniques to reduce latency. While effective, this mapping strategy becomes increasingly complex to implement as the PE size increases.

2. **Parallel Look-ahead:** As depicted in Figure 1(b), Kun et al. introduced a parallel priority look-ahead architecture where control signals (from OR gates) are generated in parallel. This reduces latency by allowing all PRI modules to operate concurrently, but it comes at the cost of increased resource utilization.

3. **Alternative Architectures:** Other designs have utilized different core components. Le et al. proposed an architecture based on a set of one-hot encoders (Figure 1(c)). Maurya and Clark used a set of comparator and sort circuits (PSC), shown in Figure 1(d). Both approaches, however, require a large number of components and complex interconnections when scaled to large input sizes, leading to significant resource consumption.

4. **Prefix Schemes:** Abdel-Hafeez and Harb presented a special prefix scheme for PEs up to 256 bits, but its performance was noted to decline sharply as the PE size increased

beyond this.

This project is based on the **1D-to-2D array conversion method** proposed by Nguyen et al., which is illustrated in Figure 32 (top). This novel method reframes the L-bit input vector as a two-dimensional $M \times N$ matrix. The core logic is then split into two parallel paths:

1. A **Row Status** vector is generated by performing a bitwise OR on all bits in each of the N rows.
2. An N-bit PE finds the highest-priority active row (row index i) from this Row Status vector.
3. An M-bit PE finds the highest-priority active column (column index j) within that specific row i.

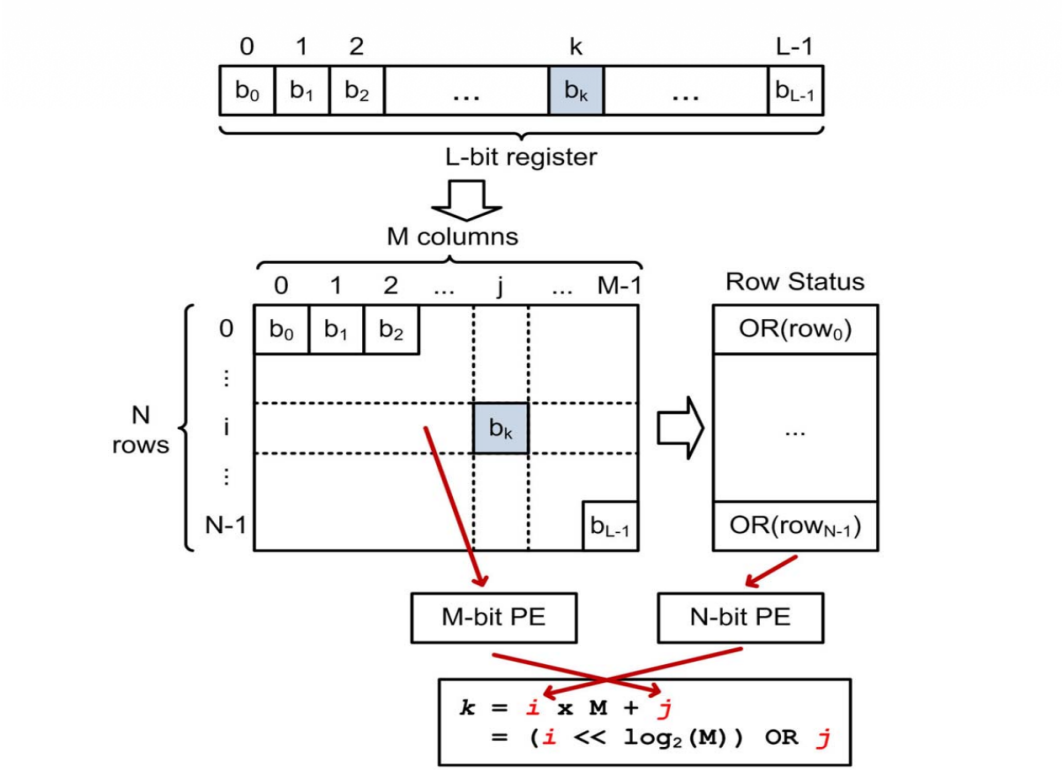


Fig. 2. The conversion from L-bit input to $M \times N$-bit input.

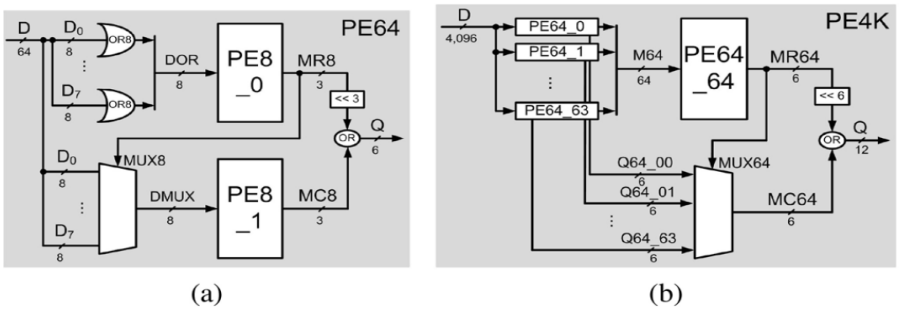


Fig. 3. The architecture of 1D-to-2D conversion based (a) PE64 and (b) PE4K.

Figure 2: The 1D-to-2D conversion method (top) and its application in a PE64 and PE4K (bottom).

The final priority index, k, is then calculated as $k = i \times M + j$. When M is a power of two (e.g., $M = 4$ or $M = 16$), this multiplier-adder logic is efficiently replaced by a simple left-shift operation and a bitwise OR, as shown in the formula in Figure 32.
While prior work demonstrated the high performance of this 1D-to-2D conversion (e.g., building a PE64 from two PE8s, as shown in Figure 32(a)), it did not identify the optimal $M \times N$ configuration for maximizing frequency. This project extends that work by systematically exploring different (M, N) pairs and introducing a look-ahead signal to create a scalable architecture that is optimized for performance.

4 Project Implementation

This project implements and analyzes several 64-bit Priority Encoder (PE) architectures to provide a comprehensive performance benchmark. The implementation follows a complete semi-custom ASIC design flow, from RTL to GDSII, using the Cadence EDA tool suite.
The core of the project involves a comparative study of three distinct PE architectures:

1. **Scalable PE with Look-ahead (`pe64_lookahead`):** This is the primary architecture under investigation, based on the 1D-to-2D array conversion method. The 64-bit input vector is transformed into a 16×4 matrix. It utilizes a `pe16` for row priority detection and a `pe4` for column priority detection, incorporating look-ahead logic (via a `casex` statement) to reduce the critical path delay.

2. **Unscalable PE (`pe64_if_else`):** This design serves as a traditional baseline. It models a flat 64-bit PE using a long chain of `if-else` statements. This serial priority-checking logic is expected to have significant latency and poor scalability.

3. **MUX-based Scalable PE (`pe64_standard`):** This architecture is structurally similar to the look-ahead version but replaces the `casex` selection logic with an explicit 16:1 multiplexer, which is itself constructed from a two-stage tree of $4:1$ MUXes. This allows for a direct comparison of how the synthesis tool handles different but functionally equivalent RTL descriptions.

To understand the impact of process scaling, the scalable (`pe64_lookahead`) and unscalable (`pe64_if_else`) architectures are implemented and analyzed on both **180nm** and **90nm** CMOS technology nodes. The MUX-based design is implemented on the 180nm node as an additional point of comparison.
The standard VLSI design flow is used for each case:

- **RTL Design & Verification:** All architectures are coded in Verilog HDL and functionally verified using a self-checking testbench in Cadence NCLaunch.

- **Logic Synthesis:** The verified RTL is synthesized into a gate-level netlist using Cadence Genus, targeting the respective 180nm or 90nm standard cell libraries.

- **Physical Implementation:** The netlist undergoes floorplanning, placement, clock tree synthesis (CTS), and routing using Cadence Innovus to generate a final GDSII layout.

The final analysis compares the post-layout (post-P&R) results for all implemented designs, focusing on the key performance metrics of **maximum operating frequency (timing)**, **total power consumption**, and **core cell area**.

5 Unscalable 64-bit PE (if-else) on 180nm Technology

To establish a performance baseline, a traditional unscalable 64-bit priority encoder was designed. This architecture, referred to as `pe64_if_else`, models a flat priority logic using a large chain of `if-else` statements. This serial structure is expected to create a long critical path, which is detrimental to high-frequency operation. This design was implemented using the 180nm CMOS technology library.

5.1 RTL Design

The core module `pe64_if_else` maps each of the 64 input bits to its corresponding 6-bit index. The `always @(*)` block creates a large combinational multiplexer that checks the highest-priority bit (starting from `d[63]`) and assigns the output `q` accordingly.
A top-level module, `pe256_from_64`, was created to demonstrate how four of these 64-bit blocks would be instantiated to build a 256-bit PE.

Listing 1: Unscalable 64-bit PE Verilog Code (`pe64_if_else`)

```
module pe64_if_else (
    input  wire [63:0] d,
    output reg  [5:0]  q,
    output wire        v
);
    always @(*) begin
        if      (d[63]) q = 6'd63;
        else if (d[62]) q = 6'd62;
        else if (d[61]) q = 6'd61;
        else if (d[60]) q = 6'd60;
        else if (d[59]) q = 6'd59;
        else if (d[58]) q = 6'd58;
        else if (d[57]) q = 6'd57;
        else if (d[56]) q = 6'd56;
        else if (d[55]) q = 6'd55;
        else if (d[54]) q = 6'd54;
        else if (d[53]) q = 6'd53;
        else if (d[52]) q = 6'd52;
        else if (d[51]) q = 6'd51;
        else if (d[50]) q = 6'd50;
        else if (d[49]) q = 6'd49;
        else if (d[48]) q = 6'd48;
        else if (d[47]) q = 6'd47;
        else if (d[46]) q = 6'd46;
        else if (d[45]) q = 6'd45;
        else if (d[44]) q = 6'd44;
        else if (d[43]) q = 6'd43;
        else if (d[42]) q = 6'd42;
        else if (d[41]) q = 6'd41;
        else if (d[40]) q = 6'd40;
        else if (d[39]) q = 6'd39;
        else if (d[38]) q = 6'd38;
        else if (d[37]) q = 6'd37;
```

```
        else if (d[36]) q = 6'd36;
        else if (d[35]) q = 6'd35;
        else if (d[34]) q = 6'd34;
        else if (d[33]) q = 6'd33;
        else if (d[32]) q = 6'd32;
        else if (d[31]) q = 6'd31;
        else if (d[30]) q = 6'd30;
        else if (d[29]) q = 6'd29;
        else if (d[28]) q = 6'd28;
        else if (d[27]) q = 6'd27;
        else if (d[26]) q = 6'd26;
        else if (d[25]) q = 6'd25;
        else if (d[24]) q = 6'd24;
        else if (d[23]) q = 6'd23;
        else if (d[22]) q = 6'd22;
        else if (d[21]) q = 6'd21;
        else if (d[20]) q = 6'd20;
        else if (d[19]) q = 6'd19;
        else if (d[18]) q = 6'd18;
        else if (d[17]) q = 6'd17;
        else if (d[16]) q = 6'd16;
        else if (d[15]) q = 6'd15;
        else if (d[14]) q = 6'd14;
        else if (d[13]) q = 6'd13;
        else if (d[12]) q = 6'd12;
        else if (d[11]) q = 6'd11;
        else if (d[10]) q = 6'd10;
        else if (d[9])  q = 6'd9;
        else if (d[8])  q = 6'd8;
        else if (d[7])  q = 6'd7;
        else if (d[6])  q = 6'd6;
        else if (d[5])  q = 6'd5;
        else if (d[4])  q = 6'd4;
        else if (d[3])  q = 6'd3;
        else if (d[2])  q = 6'd2;
        else if (d[1])  q = 6'd1;
        else if (d[0])  q = 6'd0;
        else            q = 6'd0;
    end

    assign v = |d;
endmodule

module pe256_from_64 (
    input  wire [255:0] d,
    output reg  [7:0]   q,
    output wire         v
);

    wire [5:0] q0, q1, q2, q3;
    wire       v0, v1, v2, v3;
```

```
85
86      pe64_if_else pe3 ( .d(d[255:192]), .q(q3), .v(v3) );
87      pe64_if_else pe2 ( .d(d[191:128]), .q(q2), .v(v2) );
88      pe64_if_else pe1 ( .d(d[127:64]),  .q(q1), .v(v1) );
89      pe64_if_else pe0 ( .d(d[63:0]),    .q(q0), .v(v0) );
90
91      always @(*) begin
92          if (v3) begin
93              q = {2'b11, q3};
94          end
95          else if (v2) begin
96              q = {2'b10, q2};
97          end
98          else if (v1) begin
99              q = {2'b01, q1};
100         end
101         else begin
102             q = {2'b00, q0};
103         end
104     end
105
106     assign v = v3 | v2 | v1 | v0;
107
108 endmodule
```

5.2 RTL Verification

5.2.1 Testbench Code

The design was verified using a self-checking testbench. The testbench instantiates the `pe256_from_64` module and performs a comprehensive set of tests:

1. An all-zero test to ensure the valid signal `v` is low.
2. A "one-hot" test, which iterates through all 256 bits, asserting one bit at a time and checking if the output `q` matches the input index.
3. A multi-bit test, where multiple inputs are asserted to verify that the encoder correctly identifies the one with the highest index (highest priority).
4. A series of 20 random input vectors to test for robustness against arbitrary inputs.

The testbench reports "SUCCESS" only if all tests pass.

Listing 2: Self-checking testbench for `pe256_from_64`

```
1 `timescale 1ns / 1ps
2 `include "priority.v"
3
4 module tb_pe256_selfchecking();
5
6     reg   [255:0] tb_d;
7     wire  [7:0]   tb_q;
8     wire          tb_v;
```

```
9
10      integer error_count;
11      reg [7:0] expected_q;
12      integer i;
13
14      pe256_from_64 dut (
15          .d(tb_d),
16          .q(tb_q),
17          .v(tb_v)
18      );
19
20      function [7:0] get_expected_q(input [255:0] d);
21          integer i;
22          begin : find_bit_block
23              get_expected_q = 8'b0;
24              for (i = 255; i >= 0; i = i - 1) begin
25                  if (d[i] == 1'b1) begin
26                      get_expected_q = i;
27                      disable find_bit_block;
28                  end
29              end
30          end
31      endfunction
32
33      initial begin
34          $dumpfile("pe256.vcd");
35          $dumpvars(0, tb_pe256_selfchecking);
36          error_count = 0;
37
38          tb_d = 256'b0;
39          #10;
40          if (tb_v !== 1'b0) begin
41              $display("FAIL: All-zero input, ...");
42              error_count = error_count + 1;
43          end
44
45          for (i = 0; i < 256; i = i + 1) begin
46              tb_d = 0;
47              tb_d[i] = 1'b1;
48              expected_q = i;
49              #10;
50              if (tb_q !== expected_q || tb_v !== 1'b1) begin
51                  $display("FAIL: Input bit %0d | ...", i,
                        expected_q, tb_q);
52                  error_count = error_count + 1;
53              end
54          end
55
56          tb_d = 0;
57          tb_d[123] = 1'b1;
58          tb_d[200] = 1'b1;
```

```
59          tb_d[5]    = 1'b1;
60          expected_q = 200;
61          #10;
62          if (tb_q !== expected_q) begin
63              $display("FAIL: Multi-bit test | ...", expected_q,
                    tb_q);
64              error_count = error_count + 1;
65          end
66
67          for (i = 0; i < 20; i = i + 1) begin
68              tb_d = {$random, $random, $random, $random,
69                      $random, $random, $random, $random};
70              expected_q = get_expected_q(tb_d);
71              #10;
72              if (tb_q !== expected_q) begin
73                  $display("FAIL: Random test | ...", tb_d,
                        expected_q, tb_q);
74                  error_count = error_count + 1;
75              end
76          end
77
78          #20;
79          if (error_count == 0) begin
80              $display("SUCCESS: All tests passed!");
81          end else begin
82              $display("FAILURE: %0d errors found.", error_count);
83          end
84
85          $finish;
86      end
87
88  endmodule
```

5.3 Logic Synthesis (Genus)

The `pe64_if_else` module was synthesized using Cadence Genus with the 180nm (tsmc18) slow-corner library.

5.3.1 Synthesis Scripts

The synthesis process was guided by the Tcl script in Listing 3, which sets the library, reads the HDL, and elaborates the `pe64_if_else` module. Timing constraints were provided in a separate `.sdc` file (Listing 4), setting a maximum delay of 1.2 ns.

Listing 3: Genus synthesis script (`run.tcl`)

```
set_db init_lib_search_path /home/install/FOUNDRY/digital/180nm/
   dig/lib/
set_db library slow.lib

read_hdl {./priority.v}

elaborate pe64_if_else
read_sdc ./constraint_input.sdc

set_db syn_generic_effort medium
set_db syn_map_effort medium
set_db syn_opt_effort medium

syn_generic
syn_map
syn_opt

write_hdl > priority_netlist.v
write_sdc > priority_output.sdc

report timing > priority_timing.rpt
report power > priority_power.rpt
report area  > priority_cell.rpt
report gates > priority_gates.rpt

gui_show
```

Listing 4: Synthesis design constraints (`constraints_input.sdc`)

```
set_max_delay 1.2 -from [get_ports d[*]] -to [get_ports {q[*] v}]
```

5.3.2 Synthesized Schematic

The resulting gate-level schematic synthesized by Genus is shown in Figure 3. The complex and deep cone of logic is immediately apparent, visually confirming the unscalable nature of the `if-else` design.

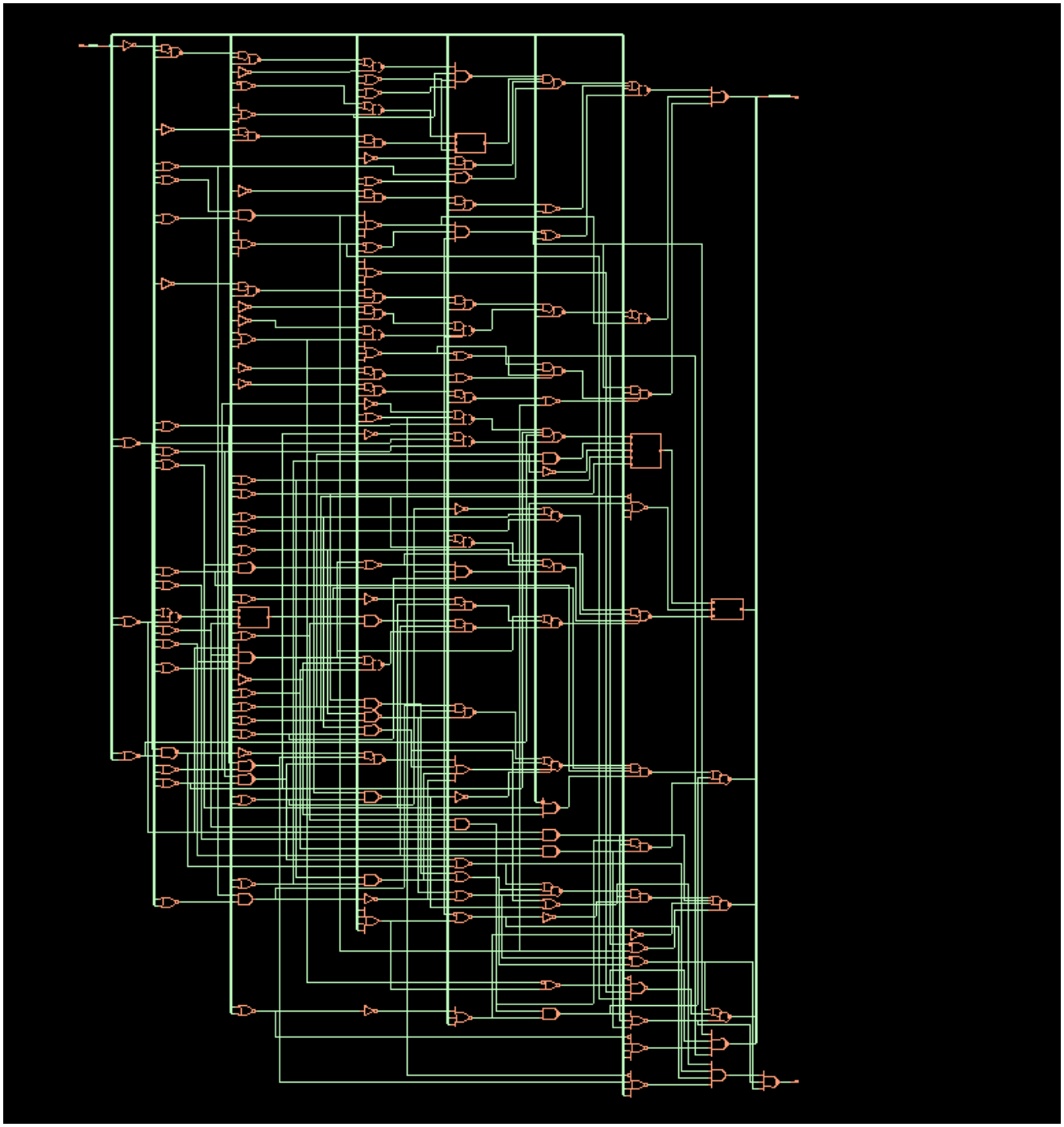
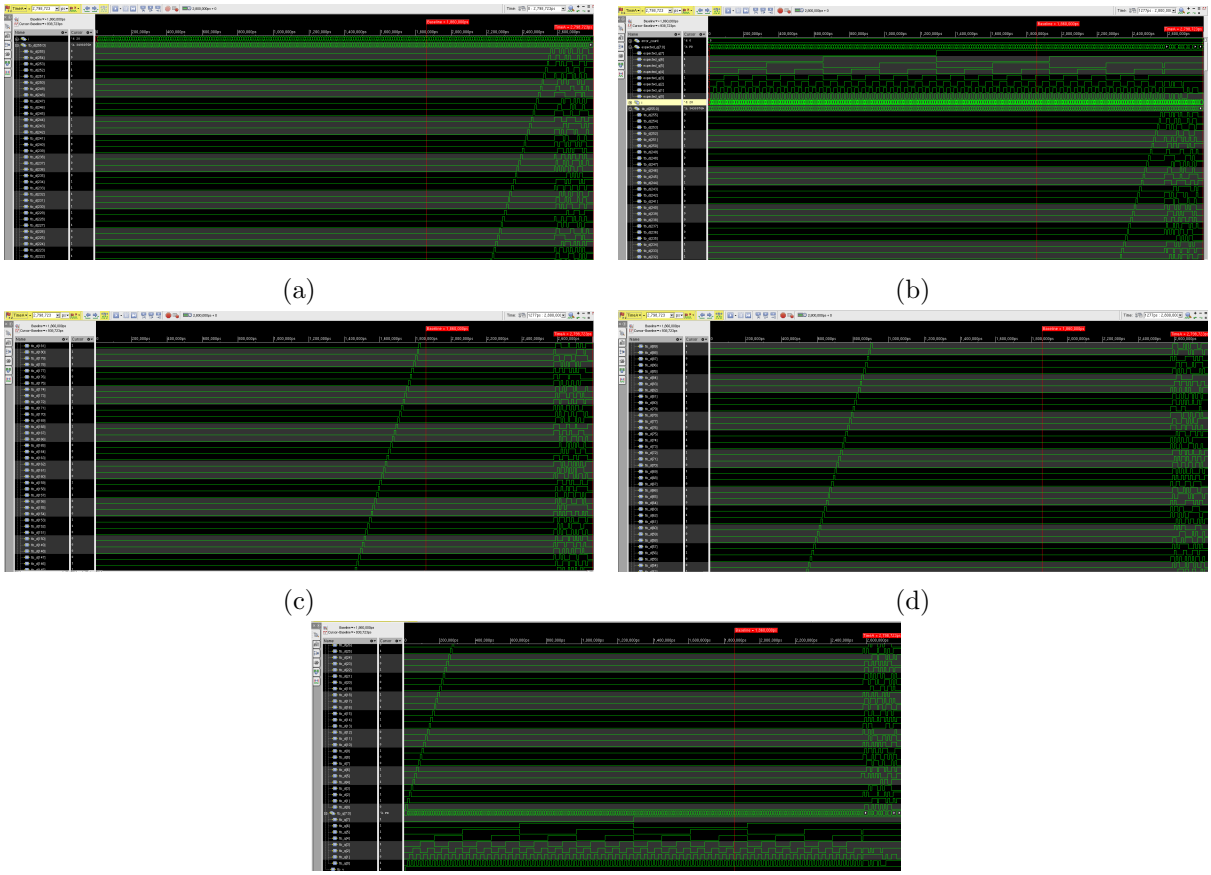


Figure 3: Synthesized schematic of `pe64_if_else` (180nm).

5.3.3 Simulation Waveforms

The design was simulated in Cadence NCLaunch. The following figures show the waveforms from the self-checking testbench, demonstrating correct functional behavior under various test conditions.



(a) (b) (c) (d) (e)

Figure 4: Simulation waveforms from NCLaunch verifying the functionality of the `pe256_from_64` module: (a) Verification of input bits 21-55, (b) Expected vs. actual output for bits 21-55, (c) Verification of input bits 45-81, (d) Verification of input bits 53-89, and (e) Verification of input bits 1-37.

5.4 Logic Synthesis (Genus) Reports

The `pe64_if_else` module was synthesized using Cadence Genus with the 180nm `slow.lib` library. A timing constraint of 1.2 ns was applied to guide the synthesis tool.

5.4.1 Timing Report

The timing report from Genus, shown in Figure 5, indicates that the design successfully met the 1.2 ns constraint. The "Category Summary" shows 0 failing paths and a Worst Negative Slack (WNS) of 0.0000.
The "Path List" details the 7 paths in the design. The critical path (Path 1) has the lowest positive slack at **0.039 ns**, corresponding to an actual critical path delay of **1.161 ns** (1.2 ns - 0.039 ns). This confirms the serial nature of the design, as it barely meets the specified timing.

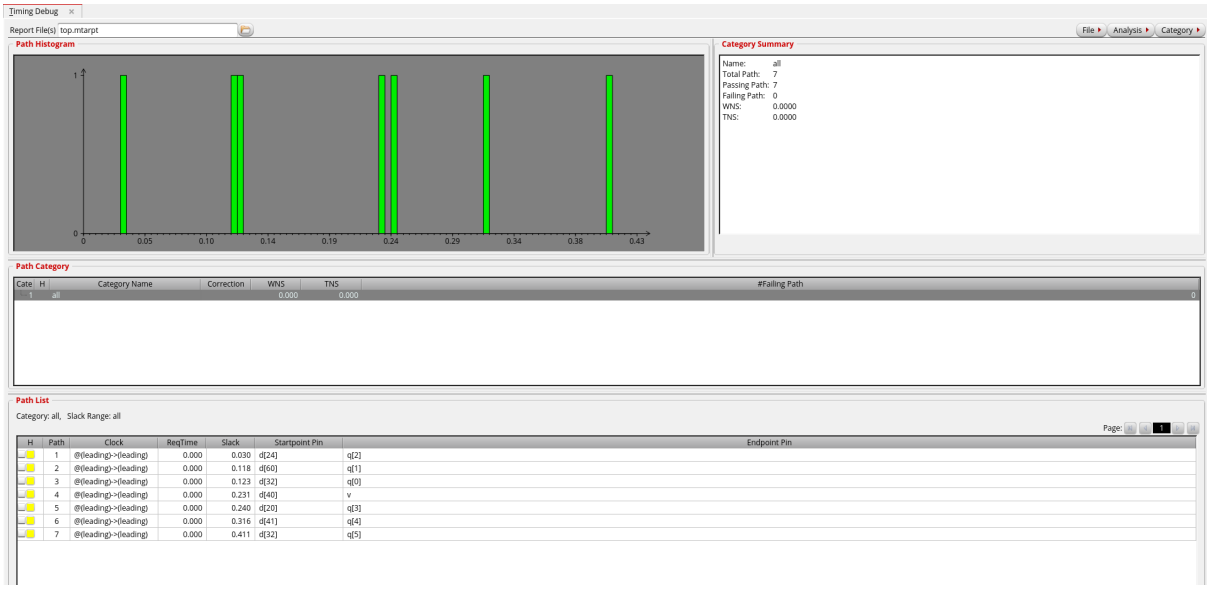


Figure 5: Genus timing debug report for `pe64_if_else` (180nm).

5.4.2 Power Report

The power report from Genus shows a total power consumption of **5.38283e-05 W**, or **53.83 μW**. As expected for the 180nm technology, the power is dominated by **Switching power (62.00%)** and **Internal power (37.91%)**. Leakage power is negligible at only 0.09%.

```
@genus:root: 3> report_power
Info    : Joules engine is used. [RPT-16]
Instance: /pe64_if_else
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   0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00  0.00%
latch      0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00  0.00%
logic      4.98203e-08  2.04056e-05  3.33728e-05  5.38283e-05  100.00%
bbox       0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00  0.00%
clock      0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00  0.00%
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   4.98203e-08  2.04056e-05  3.33728e-05  5.38283e-05  100.00%
Percentage 0.09%        37.91%       62.00%       100.00%      100.00%
-------------------------------------------------------------------------
```

The detailed power report from the GUI confirms these findings, showing the values in nanowatts (nW) for the 164 cells.

```
Generated by: Genus(TM) Synthesis Solution 20.11-s111_1 (Apr 26 2021 14:57:38)
Module: design:pe64_if_else
Technology library: tsmc18 1.0
```

```
Operating conditions: slow (balanced_tree)
Wireload mode: enclosed
================================================================================
Instance      Cells  Leakage(nW)  Internal(nW)  Net(nW)     Switching(nW)
--------------------------------------------------------------------------------
pe64_if_else  164    49.820       20405.609     33372.847   53778.456
```

5.4.3 Area Report

The synthesis area report shows that the flat `if-else` architecture required **164 cells** to implement. The final **Total Area** reported by the tool is **2012.472 μm^2**.

```
@genus:root: 2> report_area
================================================================
Generated by:     Genus(TM) Synthesis Solution 20.11-s111_1
Generated on:     Oct 10 2025 03:42:32 pm
Module:           pe64_if_else
Operating conditions: slow (balanced_tree)
Wireload mode:    enclosed
Area mode:        timing library
================================================================

Instance      Module        Cell Count  Cell Area  Net Area  Total Area  Wireload
-------------------------------------------------------------------------------
pe64_if_else                164         2012.472   0.000     2012.472 <none> (D)
```

5.5 Physical Implementation (Innovus)

Following synthesis, the gate-level netlist (`priority_netlist.v`) and timing constraints (`priority_output.sdc`) for the `pe64_if_else` design were imported into Cadence Innovus for physical implementation. This Place and Route (P&R) stage converts the non-physical netlist into a complete physical layout (GDSII) by determining the exact location of each standard cell and routing the wire connections between them. The 180nm technology libraries were used for this process.

5.5.1 Innovus Implementation Script

The P&R flow was automated using a Tcl script. This script (similar to the one in Listing 5) handles all stages of the implementation:

1. **Import:** Loads the netlist, constraints, and technology LEF files.
2. **Floorplan:** Defines the core chip area, utilization, and I/O pin placement.
3. **Power Planning:** Creates the VDD and VSS power grid.
4. **Placement:** Places the 164 standard cells from the netlist onto the core area.
5. **Clock Tree Synthesis (CTS):** Builds a balanced clock tree to distribute the clock signal (if one existed) with minimal skew.
6. **Routing:** Connects all the cells and pins using multiple metal layers as defined by the technology.
7. **Post-Route Optimization:** Performs final optimizations (nano-routing) to fix any timing or design rule check (DRC) violations.

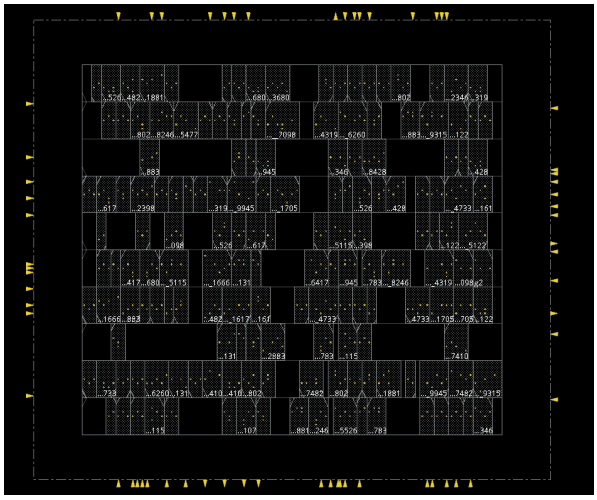
8. **Verification & Export:** Runs final checks and exports the layout as a GDSII file.

Listing 5: Example Innovus Tcl script structure (conceptual)

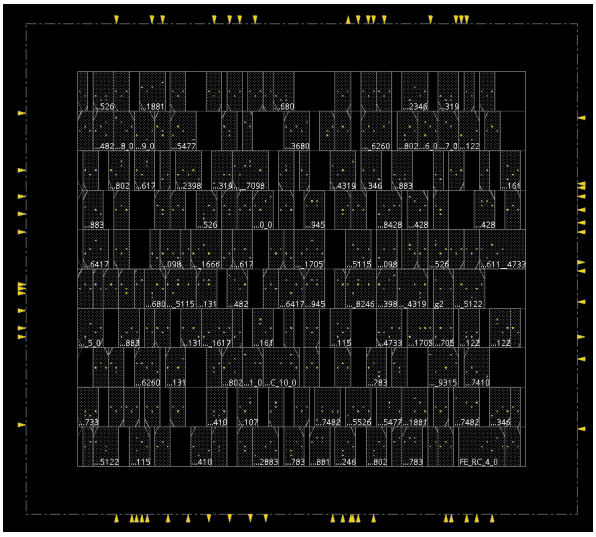
```
# 1. Import Design
setDesignMode -process 180
read_lef ...
read_verilog "priority_netlist.v"
read_sdc "priority_output.sdc"
init_design

# 2. Floorplan
floorPlan -site core -CoreToPO 5 ...
addPin -pin ...
...

# 3. Power Planning
addRing -type core_rings ...
addStripe ...

# 4. Placement
place_opt_design -pre_place_opt

# 5. Clock Tree Synthesis
create_ccopt_clock_tree
ccopt_design

# 6. Routing
route_nano
route_zrt_global
route_zrt_detail

# 7. Post-Route Optimization
optDesign -postRoute

# 8. Verification and Export
verify_drc
verify_geometry
verify_connectivity
saveNetlist "pe64_if_else_route.v"
saveDesign "pe64_if_else.gds"
...
# Report Generation
report_timing
report_power
report_area
```

5.5.2 Floorplan and Placement

Figure 6 shows the design's floorplan and cell placement. Figure 6a displays the initial placement, while Figure 6b shows the placement after post-route and nano-optimization. The placement of 164 cells within the defined core boundary is visible.



(a) Floorplan and cell placement before optimization.

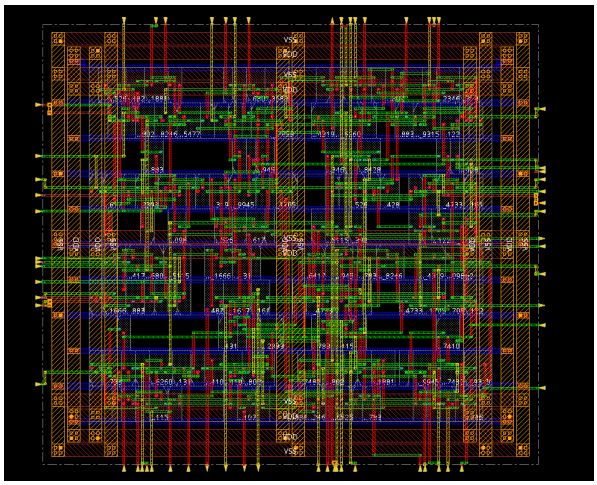


(b) Floorplan and cell placement after nano-optimization.

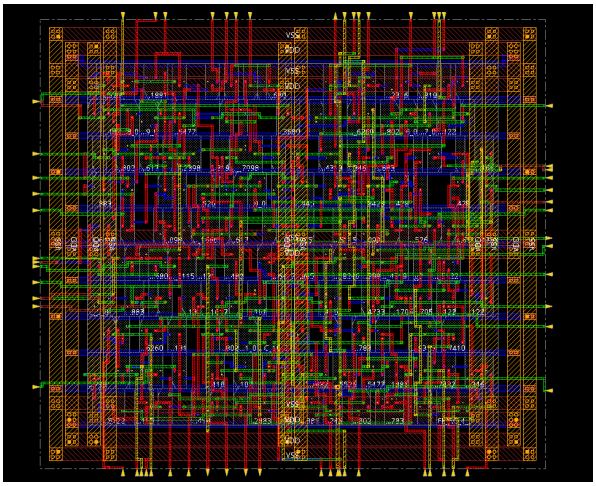
Figure 6: Innovus floorplan and placement views for `pe64_if_else` (180nm).

5.5.3 Final Layout (Post-Routing)

The final layout after routing is shown in Figure 7. This view displays all the routed metal layers, power stripes (VDD and VSS), and standard cells. Figure 7a shows the layout before final optimization, and Figure 7b shows the denser, more complex routing after nano-optimization was performed to meet timing.



(a) Layout after initial routing, before optimization.

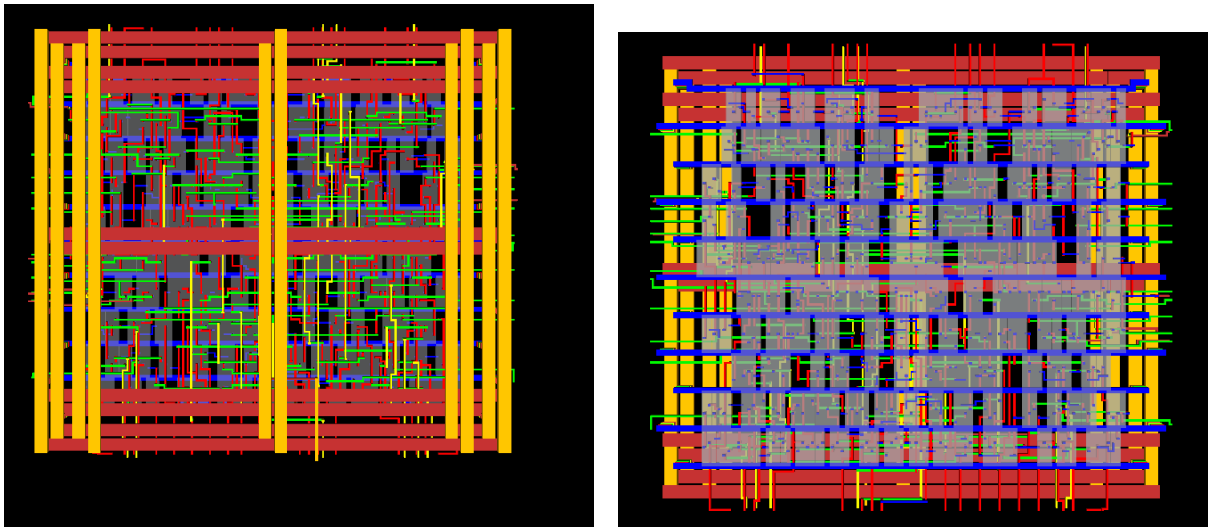


(b) Final layout after post-nano optimization.

Figure 7: Innovus final layout views for `pe64_if_else` (180nm).

5.5.4 3D Layout View

Innovus also provides a 3D view to visualize the different metal and via layers. Figure 8 shows the front and back views of the final chip layout, illustrating the stack-up of the metal layers.



(a) 3D View (Front). (b) 3D View (Back).

Figure 8: 3D layout visualization for `pe64_if_else` (180nm).

5.5.5 Pre-Optimization Reports (Initial Placement)

After the initial design import and cell placement, the tool generated preliminary reports. At this stage, the design had not been optimized for the actual physical wire delays.

Pre-Placement Timing Report The initial timing analysis shows that the design **failed** the 1.2 ns constraint. The Worst Negative Slack (WNS) was **-0.137 ns**, with 3 violating paths. This is because the Genus synthesis was based on wireload estimates, while this report uses more realistic (though not yet final) placement delays.

```
timeDesign Summary
------------------------------------------------------------
Setup views included: wc
...
+-----------------+---------+---------+
| Setup mode      | all     | default |
+-----------------+---------+---------+
| WNS (ns):       | -0.137  | -0.137  |
| TNS (ns):       | -0.292  | -0.292  |
| Violating Paths:| 3       | 3       |
| All Paths:      | 7       | 7       |
+-----------------+---------+---------+
...
Density: 69.540%
```

Pre-Placement Area Report The initial area report shows an **Instance Count of 164** and a **Total Area of 2012.472 μm^2**, which matches the post-synthesis report from Genus, as no new cells (like buffers) have been added yet.

```
@innovus 1> report_area
Hinst Name        Module Name     Inst Count     Total Area
-----------------------------------------------------------
pe64_if_else                      164            2012.472
```

Pre-Placement Power Report The initial power estimate after placement was **41.70 mW** (0.04169739 W). This is dominated by Switching (50.69%) and Internal (49.19%) power, with leakage remaining negligible.

```
Total Power:                              0.04169739
 Total Internal Power:                    0.02051305   49.1951%
 Total Switching Power:                   0.02113452   50.6855%
 Total Leakage Power:                     0.00004982   0.1195%
...
Group            Internal   Switching  Leakage    Total      Percentage (%)
...
Combinational    0.02051    0.02113    4.982e-05  0.0417     100
...
Total instances in design: 164
```

5.5.6 Post-Route Optimization Reports

The tool next performed optDesign, which routes the design and inserts buffers to fix timing violations. The reports below are from after this critical optimization step.

Post-Optimization Timing Report The optDesign summary shows that the optimization was successful. The design now **meets timing**, with a final Worst Negative Slack (WNS) of **0.003 ns** and 0 violating paths. This means the critical path delay is 1.197 ns (1.2 ns - 0.003 ns), just barely meeting the constraint.

```
optDesign Final Summary
-----------------------------------------------------------
Setup views included: wc
...
+-----------------+-------+---------+
| Setup mode      | all   | default |
+-----------------+-------+---------+
| WNS (ns):       | 0.003 | 0.003   |
| TNS (ns):       | 0.000 | 0.000   |
| Violating Paths:| 0     | 0       |
| All Paths:      | 7     | 7       |
+-----------------+-------+---------+
...
Density: 72.529%
*** Finished optDesign ***
```

Post-Optimization Area Report To meet timing, the tool inserted 6 new cells (buffers). The **Instance Count increased from 164 to 170**, and the **Total Area increased to 2098.958 μm^2**.

```
@innovus 3> report_area
Hinst Name        Module Name     Inst Count     Total Area
-----------------------------------------------------------
pe64_if_else                      170            2098.958
```

Post-Optimization Power Report With the addition of new buffers, the power increased slightly to **43.99 mW** (0.04398890 W). The distribution remains balanced between Internal (49.86%) and Switching (50.02%) power.

```
Total Power:                          0.04398890
 Total Internal Power:                0.02193180   49.8576%
 Total Switching Power:               0.02200504   50.0241%
 Total Leakage Power:                 0.00005206   0.1183%
...
Total instances in design: 170
```

Post-Nano-Routing Final Power A final power report after nano-routing shows a slight adjustment, with a final total power of **43.86 mW** (0.04385725 W).

```
Total Power:                          0.04385725
 Total Internal Power:                0.02193041   50.0041%
 Total Switching Power:               0.02187479   49.8772%
 Total Leakage Power:                 0.00005206   0.1187%
...
Total instances in design: 170
```

5.5.7 Final Verification and Routing Statistics

Finally, the design was verified for manufacturability (DRC) and correctness (LVS/Connectivity).

Physical Verification (DRC & Connectivity) The design passed all physical checks with **0 DRC Violations** and **0 Connectivity Violations**, confirming the layout is correct and manufacturable.

```
*** Starting Verify DRC (MEM: 1787.4) ***
...
VERIFY DRC ..... Sub-Area : 1 complete 0 Viols.

Verification Complete : 0 Viols.

********* Start: VERIFY CONNECTIVITY *********
Design Name: pe64_if_else
...
Begin Summary
Found no problems or warnings.
End Summary
...
Verification Complete : 0 Viols. 0 Wrngs.
```

Final Routing Statistics The post-nano-routing report confirms the final routing statistics. The **Total Wire Length is 3648 μm**, and the **Total Number of Vias is 903**. The routing is spread across 6 metal layers.

```
#Start DRC checking.
...
#Post Route wire spread is done.
#Total wire length = 3648 um.
```

```
#Total half perimeter of net bounding box = 3903 um.
#Total wire length on LAYER Metal1 = 283 um.
#Total wire length on LAYER Metal2 = 1554 um.
#Total wire length on LAYER Metal3 = 1305 um.
#Total wire length on LAYER Metal4 = 440 um.
#Total wire length on LAYER Metal5 = 54 um.
#Total wire length on LAYER Metal6 = 11 um.
#Total number of vias = 903
#Up-Via Summary (total 903):
#-
# Metal1          547
# Metal2          283
# Metal3          65
# Metal4          7
# Metal5          1
```

5.6 Comparative Analysis

5.6.1 Consolidated Results Table

Table summarizes the synthesis (Genus) and post-layout (Innovus) results for the unscalable pe64_if_else design on 180nm Technology. The Critical Path Delay is calculated by subtracting the WNS from the 1.2 ns target constraint.

Table 1: Comparative Analysis of 64-bit Priority Encoder Implementations

Architecture	Tech. Node	Analysis Stage	Cell Count	Total Area (μm^2)	Total Power	WNS (ns) @ 1.2ns	Critical Path Delay (ns)
Unscalable (pe64_if_else)	180nm	**Genus**	**164**	**2012.472**	**53.83 μW**	**0.039**	**1.161**
		Innovus	**170**	**2098.958**	**43.86 mW**	**0.003**	**1.197**

6 Unscalable 64-bit PE (if-else) on 90nm Technology

To provide a direct comparison against the scalable architecture on the same technology node, the traditional unscalable `pe64_if_else` design was also implemented using the **90nm CMOS technology** library. This allows for an analysis of architectural performance, isolating the variable of the process node.

6.1 RTL Design

The RTL code for this implementation is identical to the 180nm version, as shown in Listing 6. The `pe64_if_else` module implements the serial priority logic, and the `pe256_from_64` module instantiates it.

Listing 6: Unscalable 64-bit PE Verilog Code (`pe64_if_else`) - 90nm

```
module pe64_if_else (
    input  wire [63:0] d,
    output reg  [5:0]  q,
    output wire        v
);
    always @(*) begin
        if      (d[63]) q = 6'd63;
        else if (d[62]) q = 6'd62;
        else if (d[61]) q = 6'd61;
        else if (d[60]) q = 6'd60;
        else if (d[59]) q = 6'd59;
        else if (d[58]) q = 6'd58;
        else if (d[57]) q = 6'd57;
        else if (d[56]) q = 6'd56;
        else if (d[55]) q = 6'd55;
        else if (d[54]) q = 6'd54;
        else if (d[53]) q = 6'd53;
        else if (d[52]) q = 6'd52;
        else if (d[51]) q = 6'd51;
        else if (d[50]) q = 6'd50;
        else if (d[49]) q = 6'd49;
        else if (d[48]) q = 6'd48;
        else if (d[47]) q = 6'd47;
        else if (d[46]) q = 6'd46;
        else if (d[45]) q = 6'd45;
        else if (d[44]) q = 6'd44;
        else if (d[43]) q = 6'd43;
        else if (d[42]) q = 6'd42;
        else if (d[41]) q = 6'd41;
        else if (d[40]) q = 6'd40;
        else if (d[39]) q = 6'd39;
        else if (d[38]) q = 6'd38;
        else if (d[37]) q = 6'd37;
        else if (d[36]) q = 6'd36;
        else if (d[35]) q = 6'd35;
        else if (d[34]) q = 6'd34;
```

```
        else if (d[33]) q = 6'd33;
        else if (d[32]) q = 6'd32;
        else if (d[31]) q = 6'd31;
        else if (d[30]) q = 6'd30;
        else if (d[29]) q = 6'd29;
        else if (d[28]) q = 6'd28;
        else if (d[27]) q = 6'd27;
        else if (d[26]) q = 6'd26;
        else if (d[25]) q = 6'd25;
        else if (d[24]) q = 6'd24;
        else if (d[23]) q = 6'd23;
        else if (d[22]) q = 6'd22;
        else if (d[21]) q = 6'd21;
        else if (d[20]) q = 6'd20;
        else if (d[19]) q = 6'd19;
        else if (d[18]) q = 6'd18;
        else if (d[17]) q = 6'd17;
        else if (d[16]) q = 6'd16;
        else if (d[15]) q = 6'd15;
        else if (d[14]) q = 6'd14;
        else if (d[13]) q = 6'd13;
        else if (d[12]) q = 6'd12;
        else if (d[11]) q = 6'd11;
        else if (d[10]) q = 6'd10;
        else if (d[9])  q = 6'd9;
        else if (d[8])  q = 6'd8;
        else if (d[7])  q = 6'd7;
        else if (d[6])  q = 6'd6;
        else if (d[5])  q = 6'd5;
        else if (d[4])  q = 6'd4;
        else if (d[3])  q = 6'd3;
        else if (d[2])  q = 6'd2;
        else if (d[1])  q = 6'd1;
        else if (d[0])  q = 6'd0;
        else            q = 6'd0;
    end

    assign v = |d;
endmodule

module pe256_from_64 (
    input  wire [255:0] d,
    output reg  [7:0]   q,
    output wire         v
);

    wire [5:0] q0, q1, q2, q3;
    wire       v0, v1, v2, v3;

    pe64_if_else pe3 ( .d(d[255:192]), .q(q3), .v(v3) );
    pe64_if_else pe2 ( .d(d[191:128]), .q(q2), .v(v2) );
```

```
    pe64_if_else pe1 ( .d(d[127:64]),  .q(q1), .v(v1) );
    pe64_if_else pe0 ( .d(d[63:0]),    .q(q0), .v(v0) );

    always @(*) begin
        if (v3) begin
            q = {2'b11, q3};
        end
        else if (v2) begin
            q = {2'b10, q2};
        end
        else if (v1) begin
            q = {2'b01, q1};
        end
        else begin
            q = {2'b00, q0};
        end
    end

    assign v = v3 | v2 | v1 | v0;

endmodule
```

6.2 RTL Verification

6.2.1 Testbench Code

The same self-checking testbench (Listing 7) was used to verify the 90nm design. The logic is identical to the 180nm version, testing all-zero, one-hot, multi-bit, and random input vectors to ensure functional correctness.

Listing 7: Self-checking testbench for `pe256_from_64` (90nm)

```
`timescale 1ns / 1ps
`include "priority.v"

module tb_pe256_selfchecking();

    reg  [255:0] tb_d;
    wire [7:0]   tb_q;
    wire         tb_v;

    integer error_count;
    reg [7:0] expected_q;
    integer i;

    pe256_from_64 dut (
        .d(tb_d),
        .q(tb_q),
        .v(tb_v)
    );

    function [7:0] get_expected_q(input [255:0] d);
        integer i;
```

```
        begin : find_bit_block
            get_expected_q = 8'b0;
            for (i = 255; i >= 0; i = i - 1) begin
                if (d[i] == 1'b1) begin
                    get_expected_q = i;
                    disable find_bit_block;
                end
            end
        end
    endfunction

    initial begin
        $dumpfile("pe256.vcd");
        $dumpvars(0, tb_pe256_selfchecking);
        error_count = 0;

        tb_d = 256'b0;
        #10;
        if (tb_v !== 1'b0) begin
            $display("FAIL: All-zero input, ...");
            error_count = error_count + 1;
        end

        for (i = 0; i < 256; i = i + 1) begin
            tb_d = 0;
            tb_d[i] = 1'b1;
            expected_q = i;
            #10;
            if (tb_q !== expected_q || tb_v !== 1'b1) begin
                $display("FAIL: Input bit %0d | ...", i,
                   expected_q, tb_q);
                error_count = error_count + 1;
            end
        end

        tb_d = 0;
        tb_d[123] = 1'b1;
        tb_d[200] = 1'b1;
        tb_d[5]   = 1'b1;
        expected_q = 200;
        #10;
        if (tb_q !== expected_q) begin
            $display("FAIL: Multi-bit test | ...", expected_q,
               tb_q);
            error_count = error_count + 1;
        end

        for (i = 0; i < 20; i = i + 1) begin
            tb_d = {$random, $random, $random, $random,
                    $random, $random, $random, $random};
            expected_q = get_expected_q(tb_d);
```

```
            #10;
            if (tb_q !== expected_q) begin
                $display("FAIL: Random test | ...", tb_d,
                   expected_q, tb_q);
                error_count = error_count + 1;
            end
        end

        #20;
        if (error_count == 0) begin
            $display("SUCCESS: All tests passed!");
        end else begin
            $display("FAILURE: %0d errors found.", error_count);
        end

        $finish;
    end

endmodule
```

6.3 Logic Synthesis (Genus)

The pe64_if_else module was synthesized using Cadence Genus with the 90nm slow-corner library.

6.3.1 Synthesis Scripts

The synthesis process was guided by the Tcl script in Listing 8, which sets the 90nm library path. The same 1.2 ns timing constraint (Listing 9) was used. The output SDC file is shown in Listing 10.

Listing 8: Genus synthesis script (`run.tcl`) - 90nm

```
set_db init_lib_search_path /home/install/FOUNDRY/digital/90nm/
   dig/lib/
set_db library slow.lib

read_hdl {./priority.v}

elaborate pe64_if_else
read_sdc ./constraint_input.sdc

set_db syn_generic_effort medium
set_db syn_map_effort medium
set_db syn_opt_effort medium

syn_generic
syn_map
syn_opt

write_hdl > priority_netlist.v
write_sdc > priority_output.sdc

report timing > priority_timing.rpt
```

```
report power > priority_power.rpt
report area  > priority_cell.rpt
report gates > priority_gates.rpt

gui_show
```

Listing 9: Synthesis design constraints (`constraints_input.sdc`) - 90nm

```
set_max_delay 1.2 -from [get_ports d[]] -to [get_ports {q[] v}]
```

Listing 10: Genus output constraints (`priority_output.sdc`) - 90nm

```
#
    ##########################################################################
#  Created by Genus(TM) Synthesis Solution 20.11-s111_1 on Wed
    Oct 08 14:41:49 IST 2025
#
    ##########################################################################


set sdc_version 2.0
set_units -capacitance 1000fF
set_units -time 1000ps

# Set the current design
current_design pe64_if_else

set_max_delay 1.2 -from [list \
  [get_ports {d[63]}]  \
  [get_ports {d[62]}]  \
  [get_ports {d[61]}]  \
  [get_ports {d[60]}]  \
  [get_ports {d[59]}]  \
  [get_ports {d[58]}]  \
  [get_ports {d[57]}]  \
  [get_ports {d[56]}]  \
  [get_ports {d[55]}]  \
  [get_ports {d[54]}]  \
  [get_ports {d[53]}]  \
  [get_ports {d[52]}]  \
  [get_ports {d[51]}]  \
  [get_ports {d[50]}]  \
  [get_ports {d[49]}]  \
  [get_ports {d[48]}]  \
  [get_ports {d[47]}]  \
  [get_ports {d[46]}]  \
  [get_ports {d[45]}]  \
  [get_ports {d[44]}]  \
  [get_ports {d[43]}]  \
  [get_ports {d[42]}]  \
  [get_ports {d[41]}]  \
```

```
  [get_ports {d[40]}]  \
  [get_ports {d[39]}]  \
  [get_ports {d[38]}]  \
  [get_ports {d[37]}]  \
  [get_ports {d[36]}]  \
  [get_ports {d[35]}]  \
  [get_ports {d[34]}]  \
  [get_ports {d[33]}]  \
  [get_ports {d[32]}]  \
  [get_ports {d[31]}]  \
  [get_ports {d[30]}]  \
  [get_ports {d[29]}]  \
  [get_ports {d[28]}]  \
  [get_ports {d[27]}]  \
  [get_ports {d[26]}]  \
  [get_ports {d[25]}]  \
  [get_ports {d[24]}]  \
  [get_ports {d[23]}]  \
  [get_ports {d[22]}]  \
  [get_ports {d[21]}]  \
  [get_ports {d[20]}]  \
  [get_ports {d[19]}]  \
  [get_ports {d[18]}]  \
  [get_ports {d[17]}]  \
  [get_ports {d[16]}]  \
  [get_ports {d[15]}]  \
  [get_ports {d[14]}]  \
  [get_ports {d[13]}]  \
  [get_ports {d[12]}]  \
  [get_ports {d[11]}]  \
  [get_ports {d[10]}]  \
  [get_ports {d[9]}]  \
  [get_ports {d[8]}]  \
  [get_ports {d[7]}]  \
  [get_ports {d[6]}]  \
  [get_ports {d[5]}]  \
  [get_ports {d[4]}]  \
  [get_ports {d[3A(d[50]), .Y (n_6));
  INVXL g3498(.A (d[46]), .Y (n_5));
  INVXL g3497(.A (d[38]), .Y (n_4));
  INVXL g3503(.A (d[48]), .Y (n_3));
  INVXL g3495(.A (d[2]), .Y (n_2));
  INVXL g3496(.A (d[34]), .Y (n_1));
  INVXL g3502(.A (d[58]), .Y (n_0));
  CLKINVX1 g3501(.A (d[62]), .Y (n_8));
  INVXL g3499(.A (d[63]), .Y (n_101));
endmodule
```

6.3.2 Synthesized Schematic

The resulting gate-level schematic synthesized by Genus is shown in Figure 9. The complex and deep cone of logic is immediately apparent, visually confirming the unscalable nature of the `if-else` design.

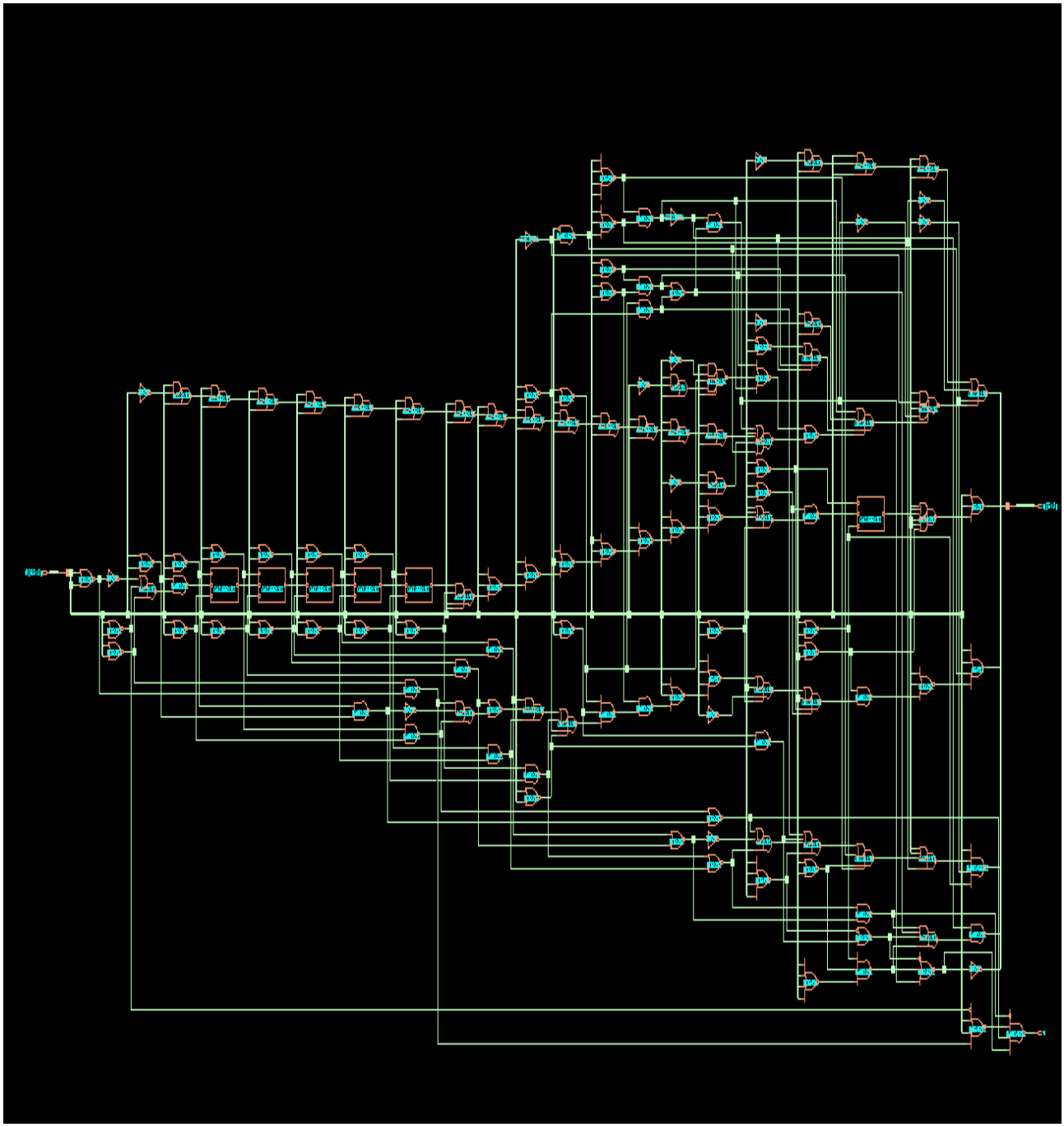


Figure 9: Synthesized schematic of `pe64_if_else` (90nm).

6.3.3 Simulation Waveforms

The functionality of the synthesized 90nm netlist was verified in Cadence NCLaunch using the same testbench. The waveforms in Figure 10 confirm correct logical behavior.

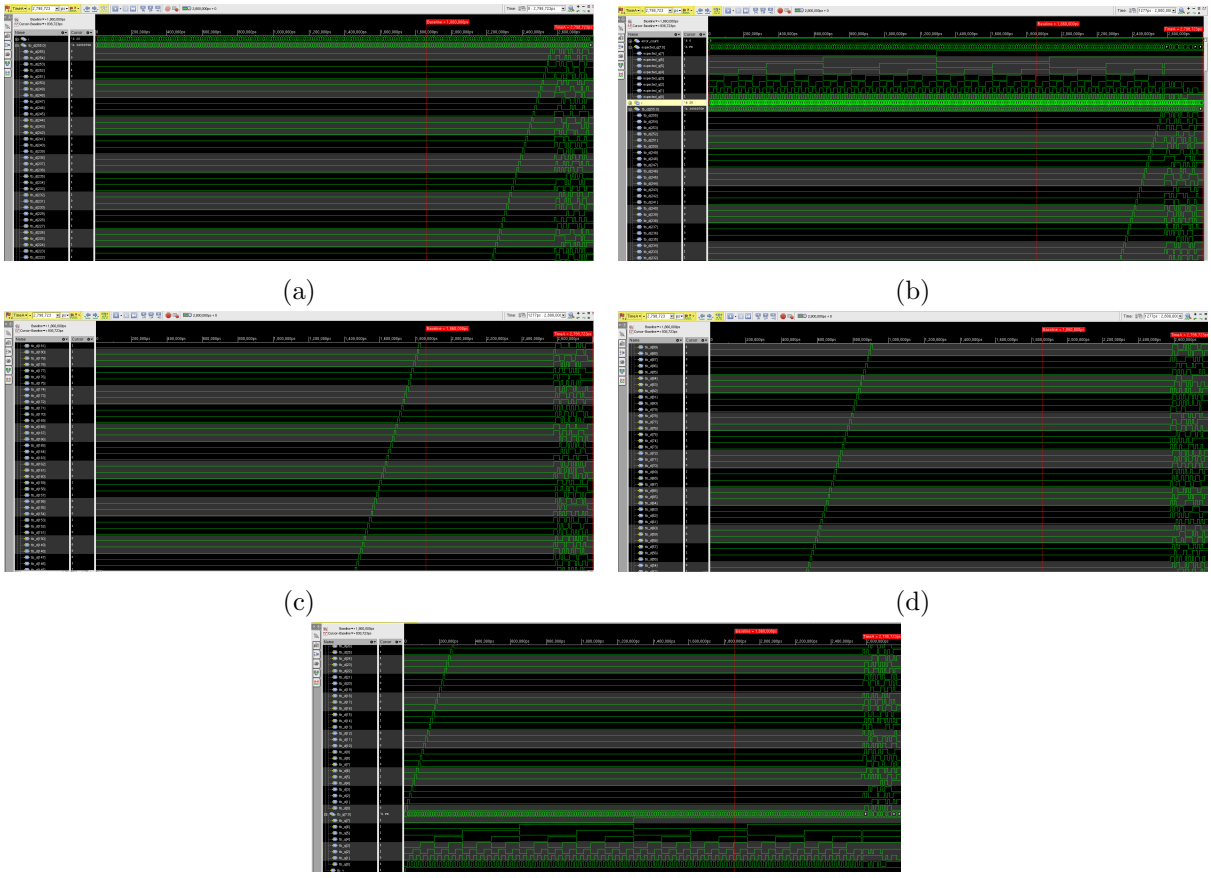


Figure 10: Simulation waveforms verifying the synthesized 90nm `pe64_if_else` netlist: (a) Verification of bits 21-55, (b) Expected vs. actual output for bits 21-55, (c) Verification of bits 45-81, (d) Verification of bits 53-89, and (e) Verification of bits 1-37.

6.3.4 Synthesis Reports

The following reports were generated by Genus for the 90nm unscalable design.

Timing Report The 90nm technology node significantly improved the performance of the unscalable design. The GUI report in Figure 11 shows the 1.2 ns constraint was easily met, with a final **WNS of 0.280 ns** (or 280 ps). This corresponds to a critical path delay of **0.920 ns** (920 ps), as detailed in the text report.

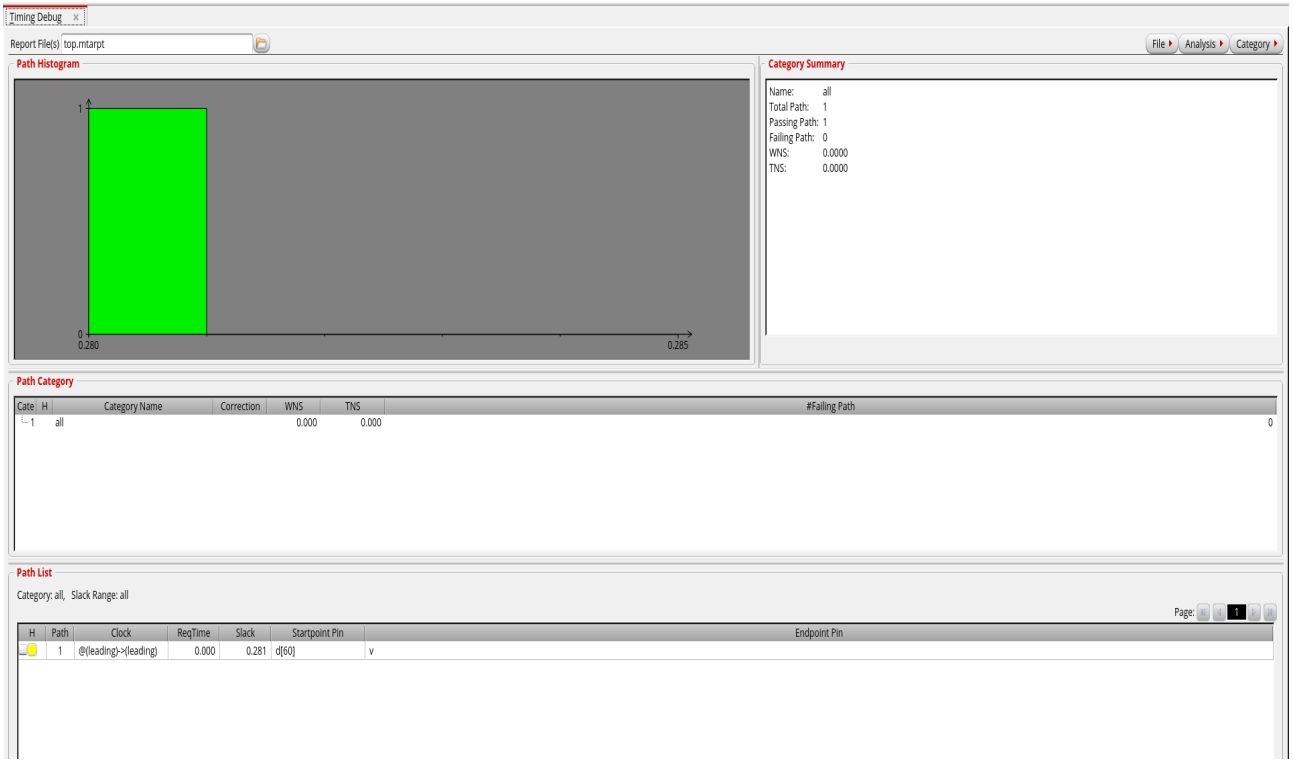


Figure 11: Genus timing debug report for `pe64_if_else` (90nm).

```
@genus:root: 4> report_timing
================================================================
Generated by:     Genus(TM) Synthesis Solution 20.11-s111_1
Generated on:     Oct 08 2025 02:42:25 pm
Module:           pe64_if_else
...
================================================================

Path 1: MET (280 ps) Path Delay Check
 Startpoint: (F) d[60]
 Endpoint: (F) v
  ...
     Required Time:=     1200
     Data Path:-         920
     Slack:=             280
...
# Timing Point      Flags   Arc    Edge  Cell         ...  Delay Arrival
# ------------------------------------------------------------------------
  d[60]             -       -      F     (arrival)    ...  0     0
  g3478_1705/Y      -       AN->Y  F     NAND2BX1     ...  137   137
  g3464_6161/Y      -       C->Y   R     NOR3X1       ...  149   286
  g3443_1617/Y      -       B->Y   F     NAND2XL      ...  164   450
  g3438/Y           -       A->Y   R     CLKINVX1     ...  90    540
  g3428_5477/Y      -       A->Y   F     NAND2XL      ...  142   682
  g3424_7482/Y      -       C->Y   R     NOR3BX1      ...  142   824
  g3419_8246/Y      -       D->Y   F     NAND4BXL     ...  95    920
  v                 -       -      F     (port)       ...  0     920
```

Power Report The total power consumption was **1.44206e-05 W**, or **14.42 μW**. Unlike the 180nm version, the power composition has shifted. **Internal power (52.99%)** is now the largest contributor, followed by Switching (31.25%) and a now-significant **Leakage (15.77%)**.

```
@genus:root: 3> report_power
...
Instance: /pe64_if_else
Power Unit: W
...
-----------------------------------------------------------------------
Category   Leakage      Internal     Switching    Total        Row%
-----------------------------------------------------------------------
...
logic      2.27366e-06  7.64122e-06  4.50575e-06  1.44206e-05  100.00%
...
-----------------------------------------------------------------------
Subtotal   2.27366e-06  7.64122e-06  4.50575e-06  1.44206e-05  100.00%
Percentage 15.77%      52.99%       31.25%       100.00%      100.00%
-----------------------------------------------------------------------
```

Area Report The 90nm implementation of the unscalable design required **139 cells** and a **Total Area of 595.680 μm^2**. This is a significant reduction from the 164 cells and 2012 μm^2 of the 180nm version.

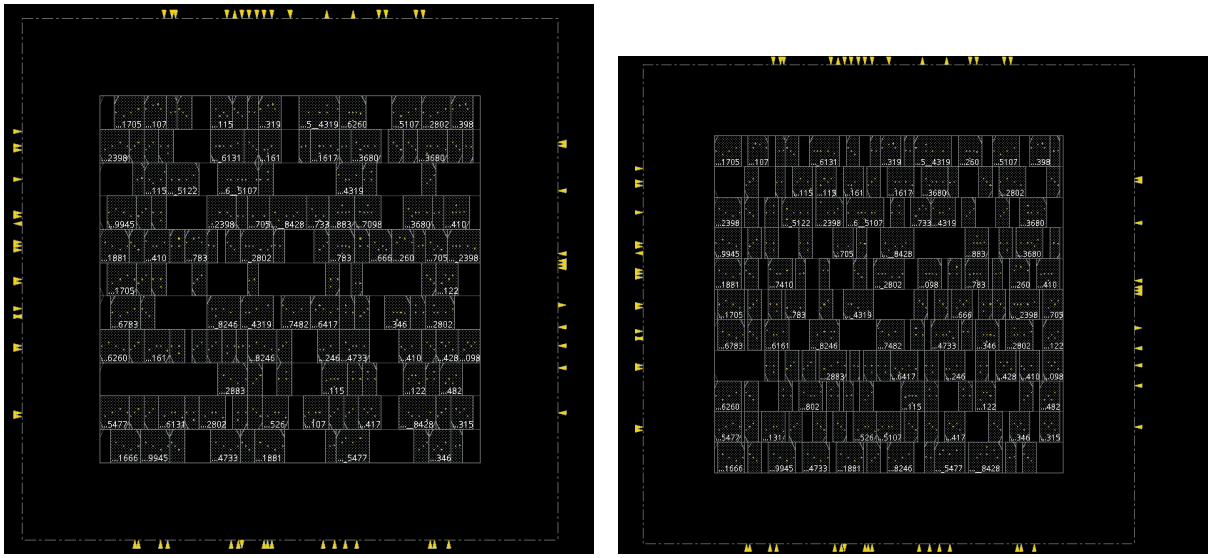
```
@genus:root: 2> report_area
================================================================
Generated by:     Genus(TM) Synthesis Solution 20.11-s111_1
Generated on:     Oct 08 2025 02:24:29 pm
Module:           pe64_if_else
...
================================================================
Instance      Module        Cell Count  Cell Area  Net Area  Total Area  Wireload
-------------------------------------------------------------------------------
pe64_if_else                139         595.680    0.000     595.680 <none> (D)
```

6.4 Physical Implementation (Innovus)

The 90nm synthesized netlist was then passed to Cadence Innovus for physical implementation.

6.4.1 Floorplan and Placement

Figure 12 shows the floorplan and cell placement for the 139 standard cells, both before and after nano-optimization.



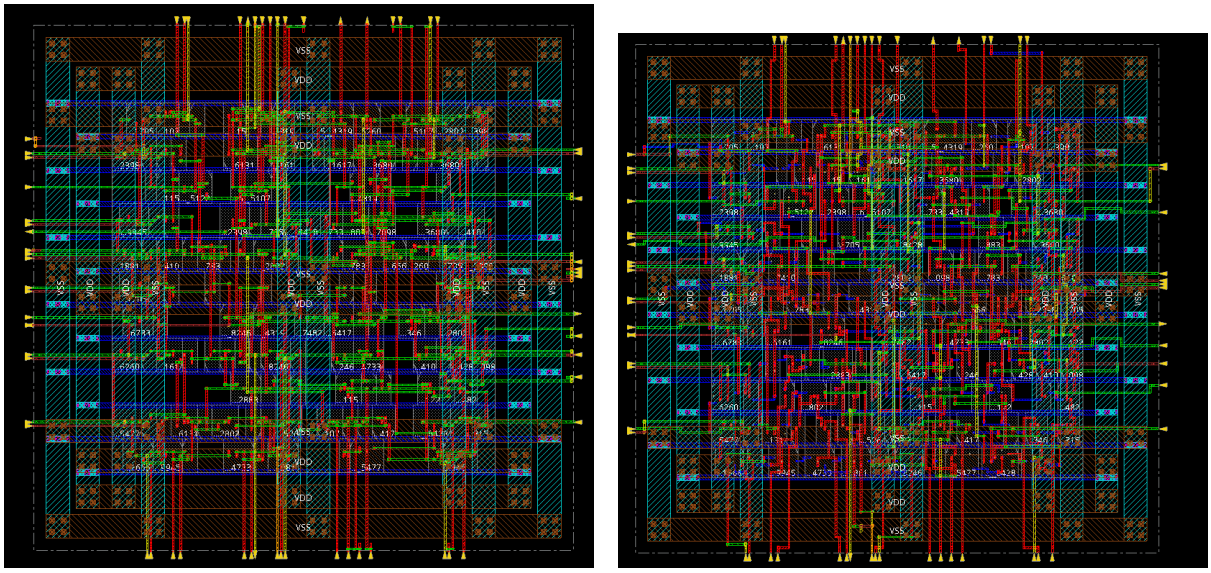
(a) Floorplan and cell placement before optimization.

(b) Floorplan and cell placement after nano-optimization.

Figure 12: Innovus floorplan and placement views for `pe64_if_else` (90nm).

6.4.2 Final Layout (Post-Routing)

The final routed layout is shown in Figure 13. A clear difference in routing density is visible between the pre-optimization layout (Figure 13a) and the final nano-optimized layout (Figure 13b).



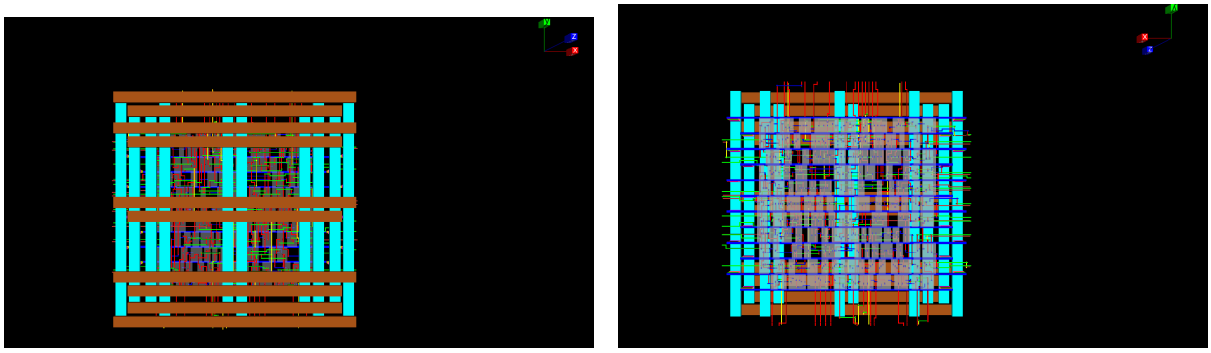
(a) Layout after initial routing, before optimization.

(b) Final layout after post-nano optimization.

Figure 13: Innovus final layout views for `pe64_if_else` (90nm).

6.4.3 3D Layout View

The 3D views in Figure 14 illustrate the physical stack-up of the metal layers for the 90nm unscalable design.



(a) 3D View (Front). (b) 3D View (Back).

Figure 14: 3D layout visualization for `pe64_if_else` (90nm).

6.5 Physical Implementation Reports (Innovus)

6.5.1 Pre-Optimization Reports (Initial Placement)

After the initial design import and cell placement, the tool generated preliminary reports.

Pre-Placement Area Report The initial area report shows an **Instance Count of 139** and a **Total Area of 595.680** μ**m**^{2}, which matches the post-synthesis report from Genus.

```
@innovus 1> report_area
Hinst Name         Module Name      Inst Count      Total Area
-----------------------------------------------------------------
pe64_if_else                         139             595.680
```

Pre-Placement Power Report The initial power estimate after placement was **19.67 mW** (0.01967077 W). For this 90nm design, the power is dominated by **Internal Power (51.93%)** and **Leakage Power (27.37%)**.

```
Total Power:                              0.01967077
 Total Internal Power:                    0.01021513    51.9305%
 Total Switching Power:                   0.00407185    20.7000%
 Total Leakage Power:                     0.00538378    27.3695%
...
Group             Internal    Switching   Leakage     Total       Percentage (%)
...
Combinational     0.01022     0.004072    0.005384    0.01967     100
...
Total instances in design: 139
```

6.5.2 Post-Route Optimization Reports

The tool next performed `optDesign`, which routes the design and performs optimizations.

Post-Optimization Timing Report The `optDesign` summary (from `design_optimized_report.png`) shows the design easily met the 1.2 ns constraint with a **WNS of 0.920 ns**. The final `timeDesign` summary (from `setup_timing_report_nano.png`) confirms a final **Setup WNS of 0.910 ns**. The hold timing check (from `hold_timing_report_nano.png`) also passed with a **Hold WNS of 0.000 ns**. This indicates a final critical path delay of 0.290 ns (1.2 ns - 0.910 ns).

```
optDesign Final Summary
---------------------------------------------------------
Setup views included: bc
...
+-----------------+-------+---------+
| Setup mode      | all   | default |
+-----------------+-------+---------+
| WNS (ns):       | 0.920 | 0.920   |
| TNS (ns):       | 0.000 | 0.000   |
| Violating Paths:| 0     | 0       |
| All Paths:      | 1     | 1       |
+-----------------+-------+---------+
...
Density: 69.462%
*** Finished optDesign ***

timeDesign Summary
---------------------------------------------------------
Setup views included: bc
...
+-----------------+-------+---------+
| Setup mode      | all   | default |
+-----------------+-------+---------+
| WNS (ns):       | 0.910 | 0.910   |
| TNS (ns):       | 0.000 | 0.000   |
| Violating Paths:| 0     | 0       |
| All Paths:      | 1     | 1       |
+-----------------+-------+---------+
Density: 69.462%

timeDesign Summary
---------------------------------------------------------
Hold views included: wc
...
+-----------------+-------+---------+
| Hold mode       | all   | default |
+-----------------+-------+---------+
| WNS (ns):       | 0.000 | 0.000   |
| TNS (ns):       | 0.000 | 0.000   |
| Violating Paths:| 0     | 0       |
| All Paths:      | 0     | 0       |
+-----------------+-------+---------+
Density: 69.462%
```

Post-Optimization Area Report The post-optimization area report shows the **Instance Count (139)** and **Total Area (595.680 μm^2)** did not change, indicating that no buffers were needed to meet the 1.2 ns constraint.

```
@innovus 1> report_area
Hinst Name         Module Name     Inst Count     Total Area
---------------------------------------------------------
pe64_if_else                       139            595.680
```

Post-Optimization Power Report The final power report after nano-routing shows a total power of **19.93 mW** (0.01992813 W). This is a slight increase from the pre-placement estimate, with a similar distribution.

```
Total Power:                          0.01992813
 Total Internal Power:                0.01022597   51.3143%
 Total Switching Power:               0.00431837   21.6697%
 Total Leakage Power:                 0.00538378   27.0160%
...
Total instances in design: 139
```

6.5.3 Final Verification and Routing Statistics

Finally, the design was verified for manufacturability (DRC) and correctness (LVS/Connectivity).

Physical Verification (DRC, LVS, Antenna) The design passed all physical checks with **0 DRC Violations**, **0 Connectivity Violations**, and **0 Antenna Violations**, confirming the layout is correct and manufacturable.

```
@innovus 5> # check_drc ...
*** Starting Verify DRC (MEM: 1628.4) ***
...
VERIFY DRC ..... Sub-Area : 1 complete 0 Viols.

Verification Complete : 0 Viols.

@innovus 5> VERIFY_CONNECTIVITY use new engine.
********* Start: VERIFY CONNECTIVITY *********
Design Name: pe64_if_else
...
Begin Summary
Found no problems or warnings.
End Summary
...
Verification Complete : 0 Viols. 0 Wrngs.

@innovus 9>
********* START VERIFY ANTENNA *********
Report File: pe64_if_else.antenna.rpt
...
Verification Complete: 0 Violations
******** DONE VERIFY ANTENNA ********
```

Final Routing Statistics The post-routing report confirms the final routing statistics. The **Total Wire Length is 1903 μm**, and the **Total Number of Vias is 822**. The routing is spread across 6 metal layers (Metal1-Metal6).

```
#Start DRC checking.
...
#Post Route wire spread is done.
#Total wire length = 1903 um.
#Total half perimeter of net bounding box = 1960 um.
```

```
#Total wire length on LAYER Metal1 = 163 um.
#Total wire length on LAYER Metal2 = 934 um.
#Total wire length on LAYER Metal3 = 538 um.
#Total wire length on LAYER Metal4 = 170 um.
#Total wire length on LAYER Metal5 = 87 um.
#Total wire length on LAYER Metal6 = 10 um.
#Total number of vias = 822
#Up-Via Summary (total 822):
#-
# Metal1          485
# Metal2          261
# Metal3          58
# Metal4          16
# Metal5          2
```

6.6 Comparative Analysis

6.6.1 Consolidated Results Table

Table 6 summarizes the synthesis (Genus) and post-layout (Innovus) results for the implemented unscalable designs. The "Critical Path Delay" is calculated by subtracting the WNS from the 1.2 ns target constraint.

Table 2: Comparative Analysis of Unscalable 64-bit Priority Encoder

Architecture	Tech. Node	Analysis Stage	Cell Count	Total Area (μm^2)	Total Power	WNS (ns) @ 1.2ns	Critical Path Delay (ns)
Unscalable (pe64_if_else)	**180nm**	**Genus**	**164**	**2012.472**	**53.83 μW**	**0.039**	**1.161**
		Innovus	**170**	**2098.958**	**43.86 mW**	**0.003**	**1.197**
Unscalable (pe64_if_else)	**90nm**	**Genus**	**139**	**595.680**	**14.42 μW**	**0.280**	**0.920**
		Innovus	**139**	**595.680**	**19.93 mW**	**0.910**	**0.290**

7 64-bit PE (MUX-based) on 180nm Technology

To analyze the impact of RTL coding style on synthesis, a third architecture was created. This version, `pe64_lookahead` (from `priority_mux.v`), implements the same scalable 1D-to-2D logic as the high-performance model. However, it explicitly replaces the `casex` statement with a tree of conditional operators (`?:`). This is intended to "guide" the synthesis tool to build a pure multiplexer-based selection tree, rather than a priority-inferred logic block. This design was implemented using the **180nm CMOS technology** library.

7.1 RTL Design

The complete Verilog code is shown in Listing 11. The file defines the full scalable hierarchy, from pe4 up to `pe256_scalable`. The key modules are:

- `pe16`: Uses a conditional operator (`?:`) for its 4:1 MUX.
- `pe64_lookahead`: This is the module elaborated for analysis. It replaces the priority-based `casex` with a two-stage 16:1 MUX tree built from conditional operators.
- `pe256_scalable`: The top-level module that instantiates the MUX-based `pe64_lookahead`.

Listing 11: Scalable MUX-based PE Verilog Code (`priority_mux.v`)

```
1  `timescale 1ns / 1ps
2
3  module pe4 (
4      input  wire [3:0] d,
5      output wire [1:0] q,
6      output wire       v
7  );
8      assign q[1] = d[3] | d[2];
9      assign q[0] = d[3] | (d[1] & ~d[2]);
10     assign v = |d;
11 endmodule
12
13
14 module pe16 (
15     input  wire [15:0] d,
16     output wire [3:0]  q,
17     output wire        v
18 );
19     wire [3:0] row_status;
20     wire [1:0] row_index;
21     wire [1:0] col_index;
22     wire       row_valid;
23     wire [3:0] selected_row; // Changed from reg to wire
24
25     assign row_status[0] = |d[3:0];
26     assign row_status[1] = |d[7:4];
27     assign row_status[2] = |d[11:8];
28     assign row_status[3] = |d[15:12];
29
```

```
30      pe4 row_pe (
31          .d(row_status),
32          .q(row_index),
33          .v(row_valid)
34      );
35
36      // MUX-based selection for selected_row (4:1 MUX, 4-bits wide
            )
37      assign selected_row = (row_index == 2'b00) ? d[3:0]    :
38                            (row_index == 2'b01) ? d[7:4]    :
39                            (row_index == 2'b10) ? d[11:8]   :
40                            (row_index == 2'b11) ? d[15:12]  : 4'
                                 b0000;
41
42      pe4 col_pe (
43          .d(selected_row),
44          .q(col_index),
45          .v() // valid bit is not needed here
46      );
47
48      assign q = {row_index, col_index};
49      assign v = row_valid;
50  endmodule
51
52
53  module pe64_standard (
54      input  wire [63:0] d,
55      output wire [5:0]  q,
56      output wire        v
57  );
58      wire [15:0] row_status;
59      wire [3:0]  selected_row; // Changed from reg to wire
60      wire [3:0]  row_index;
61      wire [1:0]  col_index;
62      wire        row_valid;
63
64      genvar i;
65      generate
66          for (i = 0; i < 16; i = i + 1) begin : row_logic
67              assign row_status[i] = |d[(i*4)+3 : i*4];
68          end
69      endgenerate
70
71      pe16 row_pe (
72          .d(row_status),
73          .q(row_index),
74          .v(row_valid)
75      );
76
77
78      wire [3:0] mux_stage1_out0;
```

```
79      wire [3:0] mux_stage1_out1;
80      wire [3:0] mux_stage1_out2;
81      wire [3:0] mux_stage1_out3;
82
83
84      assign mux_stage1_out0 = (row_index[1:0] == 2'b00) ? d[3:0]
            :
85                               (row_index[1:0] == 2'b01) ? d[7:4]
                                     :
86                               (row_index[1:0] == 2'b10) ? d[11:8]
                                     : d[15:12];
87      assign mux_stage1_out1 = (row_index[1:0] == 2'b00) ? d[19:16]
            :
88                               (row_index[1:0] == 2'b01) ? d[23:20]
                                     :
89                               (row_index[1:0] == 2'b10) ? d[27:24]
                                     : d[31:28];
90      assign mux_stage1_out2 = (row_index[1:0] == 2'b00) ? d[35:32]
            :
91                               (row_index[1:0] == 2'b01) ? d[39:36]
                                     :
92                               (row_index[1:0] == 2'b10) ? d[43:40]
                                     : d[47:44];
93      assign mux_stage1_out3 = (row_index[1:0] == 2'b00) ? d[51:48]
            :
94                               (row_index[1:0] == 2'b01) ? d[55:52]
                                     :
95                               (row_index[1:0] == 2'b10) ? d[59:56]
                                     : d[63:60];
96
97
98      assign selected_row = (row_index[3:2] == 2'b00) ?
           mux_stage1_out0 :
99                            (row_index[3:2] == 2'b01) ?
                                 mux_stage1_out1 :
100                           (row_index[3:2] == 2'b10) ?
                                 mux_stage1_out2 :
101                           (row_index[3:2] == 2'b11) ?
                                 mux_stage1_out3 : 4'b0000;
102
103     pe4 col_pe (
104         .d(selected_row),
105         .q(col_index),
106         .v()
107     );
108
109     assign q = {row_index, col_index};
110     assign v = row_valid;
111 endmodule
112
113
```

```
114 module pe64_lookahead (
115     input  wire [63:0] d,
116     output wire [5:0]  q,
117     output wire        v
118 );
119     wire [15:0] dor;
120     wire [3:0]  row_index;
121     wire [3:0]  column_data; // Changed from reg+wire to single
        wire
122     wire [1:0]  col_index;
123     wire        row_valid;
124 
125     genvar i;
126     generate
127         for (i = 0; i < 16; i = i + 1) begin : row_or_logic
128             assign dor[i] = |d[(i*4)+3 : i*4];
129         end
130     endgenerate
131 
132     pe16 row_encoder (
133         .d(dor),
134         .q(row_index),
135         .v(row_valid)
136     );
137 
138 
139     wire [3:0] mux_stage1_out0;
140     wire [3:0] mux_stage1_out1;
141     wire [3:0] mux_stage1_out2;
142     wire [3:0] mux_stage1_out3;
143 
144 
145     assign mux_stage1_out0 = (row_index[1:0] == 2'b00) ? d[3:0]
          :
146                              (row_index[1:0] == 2'b01) ? d[7:4]
                                   :
147                              (row_index[1:0] == 2'b10) ? d[11:8]
                                   : d[15:12];
148     assign mux_stage1_out1 = (row_index[1:0] == 2'b00) ? d[19:16]
          :
149                              (row_index[1:0] == 2'b01) ? d[23:20]
                                   :
150                              (row_index[1:0] == 2'b10) ? d[27:24]
                                   : d[31:28];
151     assign mux_stage1_out2 = (row_index[1:0] == 2'b00) ? d[35:32]
          :
152                              (row_index[1:0] == 2'b01) ? d[39:36]
                                   :
153                              (row_index[1:0] == 2'b10) ? d[43:40]
                                   : d[47:44];
154     assign mux_stage1_out3 = (row_index[1:0] == 2'b00) ? d[51:48]
```

```
        :
                            (row_index[1:0] == 2'b01) ? d[55:52]
                                :
                            (row_index[1:0] == 2'b10) ? d[59:56]
                                : d[63:60];

    assign column_data = (row_index[3:2] == 2'b00) ?
       mux_stage1_out0 :
                         (row_index[3:2] == 2'b01) ?
                            mux_stage1_out1 :
                         (row_index[3:2] == 2'b10) ?
                            mux_stage1_out2 :
                         (row_index[3:2] == 2'b11) ?
                            mux_stage1_out3 : 4'b0000;

    pe4 col_encoder (
        .d(column_data),
        .q(col_index),
        .v()
    );

    assign q = {row_index, col_index};
    assign v = row_valid;
endmodule


module pe256_scalable (
    input  wire [255:0] d,
    output wire [7:0]   q,
    output wire         v
);
    wire [3:0] block_status;
    wire [1:0] block_index;
    wire [5:0] internal_index;
    wire       block_valid, internal_valid;

    assign block_status[0] = |d[63:0];
    assign block_status[1] = |d[127:64];
    assign block_status[2] = |d[191:128];
    assign block_status[3] = |d[255:192];

    pe4 block_selector (
        .d(block_status),
        .q(block_index),
        .v(block_valid)
    );


    wire [63:0] selected_block;
    assign selected_block = (block_index == 2'b00) ? d[63:0]
```

```
        :
                                (block_index == 2'b01) ? d[127:64]
                                   :
                                (block_index == 2'b10) ? d[191:128]
                                   :
                                (block_index == 2'b11) ? d[255:192]
                                   : 64'b0;

    pe64_lookahead internal_pe (
        .d(selected_block),
        .q(internal_index),
        .v(internal_valid)
    );

    assign q = {block_index, internal_index};
    assign v = block_valid;
endmodule
```

7.2 RTL Verification

7.2.1 Testbench Code

A self-checking testbench (Listing 12) was used to verify the pe256_scalable module. This testbench is functionally identical to the ones used for the other designs, performing all-zero, one-hot, multi-bit, and random vector checks.

Listing 12: Self-checking testbench for pe256_scalable

```
`timescale 1ns / 1ps
`include "priority_mux.v"

module tb_pe256_scalable_selfchecking();

    reg   [255:0] tb_d;
    wire  [7:0]   tb_q;
    wire          tb_v;

    integer error_count;
    reg [7:0] expected_q;
    integer i;

    pe256_scalable dut (
        .d(tb_d),
        .q(tb_q),
        .v(tb_v)
    );

    function [7:0] get_expected_q(input [255:0] d);
        integer i;
        begin : find_bit_block
            get_expected_q = 8'b0;
            for (i = 255; i >= 0; i = i - 1) begin
                if (d[i] == 1'b1) begin
```

```
                    get_expected_q = i;
                    disable find_bit_block;
                end
            end
        end
    endfunction

    initial begin
        $dumpfile("pe256.vcd");
        $dumpvars(0, tb_pe256_scalable_selfchecking);
        error_count = 0;

        tb_d = 256'b0;
        #10;
        if (tb_v !== 1'b0) begin
            $display("FAIL: All-zero input, ...");
            error_count = error_count + 1;
        end

        for (i = 0; i < 256; i = i + 1) begin
            tb_d = 0;
            tb_d[i] = 1'b1;
            expected_q = i;
            #10;
            if (tb_q !== expected_q || tb_v !== 1'b1) begin
                $display("FAIL: Input bit %0d | ...", i,
                    expected_q, tb_q);
                error_count = error_count + 1;
            end
        end

        tb_d = 0;
        tb_d[123] = 1'b1;
        tb_d[200] = 1'b1;
        tb_d[5]   = 1'b1;
        expected_q = 200;
        #10;
        if (tb_q !== expected_q) begin
            $display("FAIL: Multi-bit test | ...", expected_q,
                tb_q);
            error_count = error_count + 1;
        end

        for (i = 0; i < 20; i = i + 1) begin
            tb_d = {$random, $random, $random, $random,
                    $random, $random, $random, $random};
            expected_q = get_expected_q(tb_d);
            #10;
            if (tb_q !== expected_q) begin
                $display("FAIL: Random test | ...", tb_d,
                    expected_q, tb_q);
```

```
                error_count = error_count + 1;
            end
        end

        #20;
        if (error_count == 0) begin
            $display("SUCCESS: All tests passed!");
        end else begin
            $display("FAILURE: %0d errors found.", error_count);
        end

        $finish;
    end
endmodule
```

7.2.2 Synthesized Schematic

The resulting gate-level schematic synthesized by Genus is shown in Figure 15. The schematic view in Genus shows this design consists of 191 leaf cells.

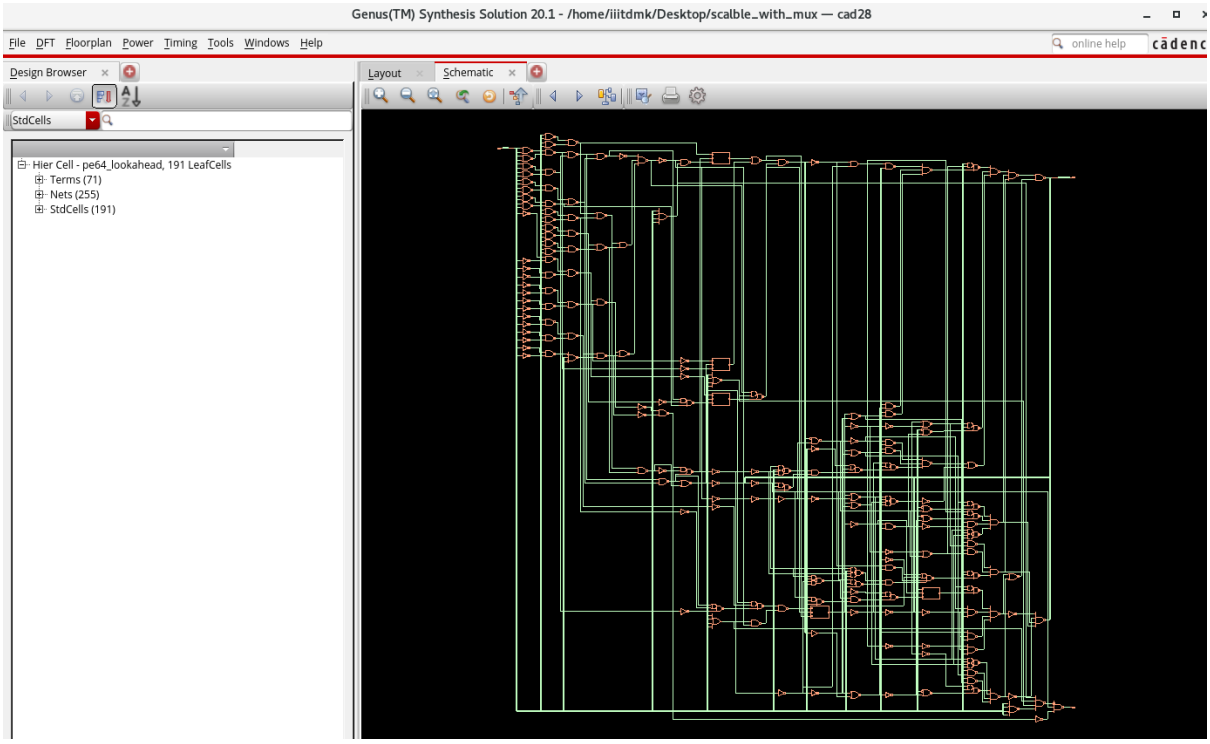


Figure 15: Synthesized schematic of MUX-based `pe64_lookahead` (180nm).

7.2.3 Simulation Waveforms

The functionality of the synthesized 180nm MUX-based netlist was verified in Cadence NCLaunch.

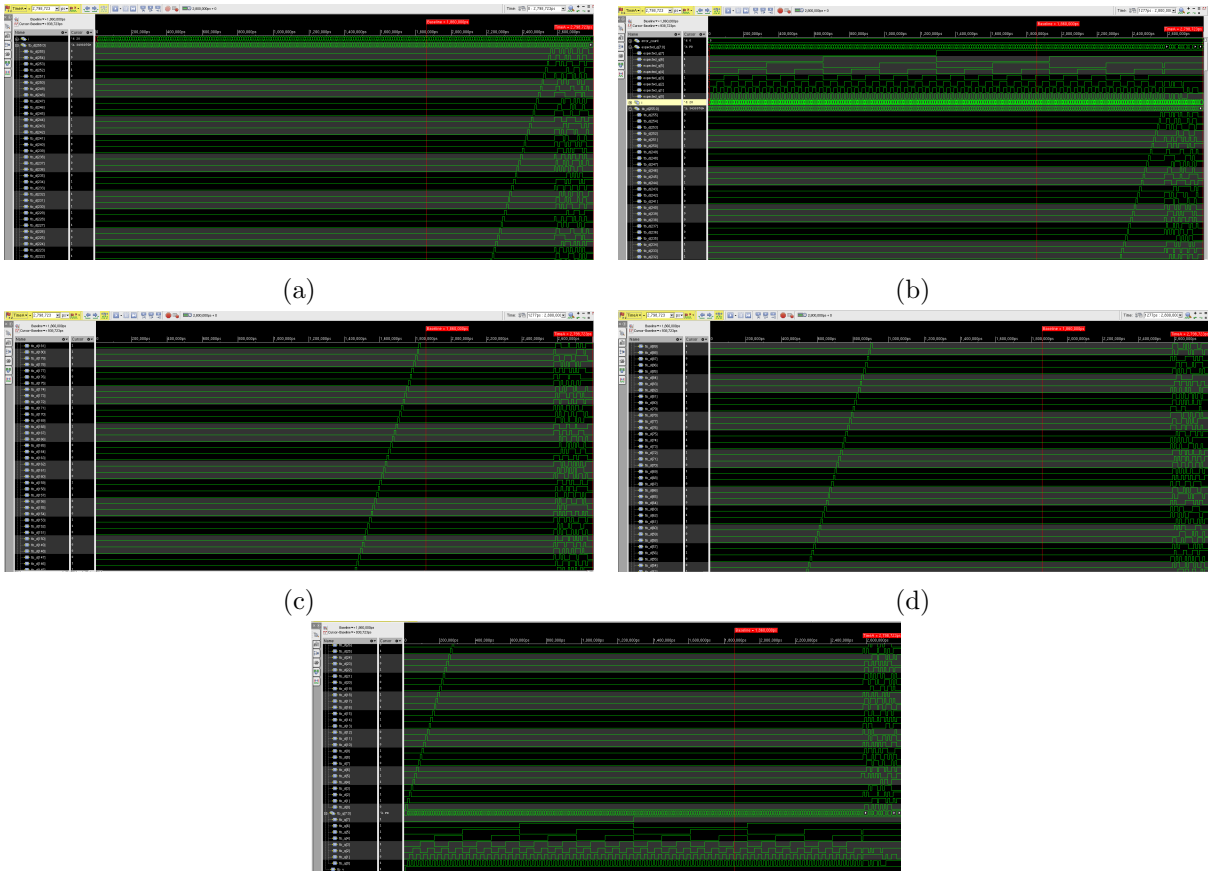


Figure 16: Simulation waveforms verifying the synthesized 180nm MUX-based `pe64_lookahead` netlist: (a) Verification of bits 21-55, (b) Expected vs. actual output, (c) Verification of bits 45-81, (d) Verification of bits 53-89, and (e) Verification of bits 1-37.

7.3 Logic Synthesis (Genus)

The `pe64_lookahead` module from `priority_mux.v` was synthesized using Cadence Genus with the 180nm slow-corner library.

7.3.1 Synthesis Scripts

The synthesis process was guided by the Tcl script in Listing 13. The same 1.2 ns timing constraint (Listing 14) was used.

Listing 13: Genus synthesis script (`run.tcl`) - 180nm MUX

```
set_db init_lib_search_path /home/install/FOUNDRY/digital/180nm/
   dig/lib/
set_db library slow.lib

read_hdl {./priority_mux.v}

elaborate pe64_lookahead
read_sdc ./constraint_input.sdc

set_db syn_generic_effort medium
set_db syn_map_effort medium
```

```
set_db syn_opt_effort medium

syn_generic
syn_map
syn_opt

write_hdl > priority_mux_netlist.v
write_sdc > priority_mux_output.sdc

report timing > priority_mux_timing.rpt
report power > priority_mux_power.rpt
report area  > priority_mux_cell.rpt
report gates > priority_mux_gates.rpt

gui_show
```

Listing 14: Synthesis design constraints (`constraints_input.sdc`) - 180nm MUX

```
set_max_delay 1.2 -from [get_ports d[*]] -to [get_ports {q[*] v}]
```

7.3.2 Synthesized Netlist

The following is the gate-level netlist (Listing 15) generated by Genus for the 180nm MUX-based design.

Listing 15: Synthesized Netlist (`priority_mux_netlist.v`) - 180nm MUX

```
// Generated by Cadence Genus(TM) Synthesis Solution 20.11-s111_1
// Generated on: Oct 31 2025 14:50:52 IST (Oct 31 2025 09:20:52 UTC)
// Verification Directory fv/pe64_lookahead

module pe64_lookahead(d, q, v);
  input [63:0] d;
  output [5:0] q;
  output v;
  wire [63:0] d;
  wire [5:0] q;
  wire v;
  wire n_0, n_1, n_2, n_3, n_4, n_6, n_7, n_8;
  wire n_9, n_10, n_11, n_12, n_13, n_14, n_15, n_16;
...
  INVX2 fopt1(.A (q[2]), .Y (n_257));
  INVX1 fopt2(.A (n_260), .Y (q[2]));
...
  AOI2BB1X2 g2(.A0N (n_283), .A1N (n_44), .B0 (n_63), .Y (n_290));
endmodule
```

7.3.3 Synthesis Reports

The following reports were generated by Genus for the 180nm MUX-based design.

Timing Report The 180nm MUX-based design **VIOLATED** the 1.2 ns timing constraint. The GUI report in Figure 17 shows 5 failing paths and a **WNS of -0.537 ns** (or -537 ps). The text-based report confirms this, showing a data path delay of 1737 ps (1.737 ns) on the critical path, which is significantly slower than the 1.2 ns requirement.

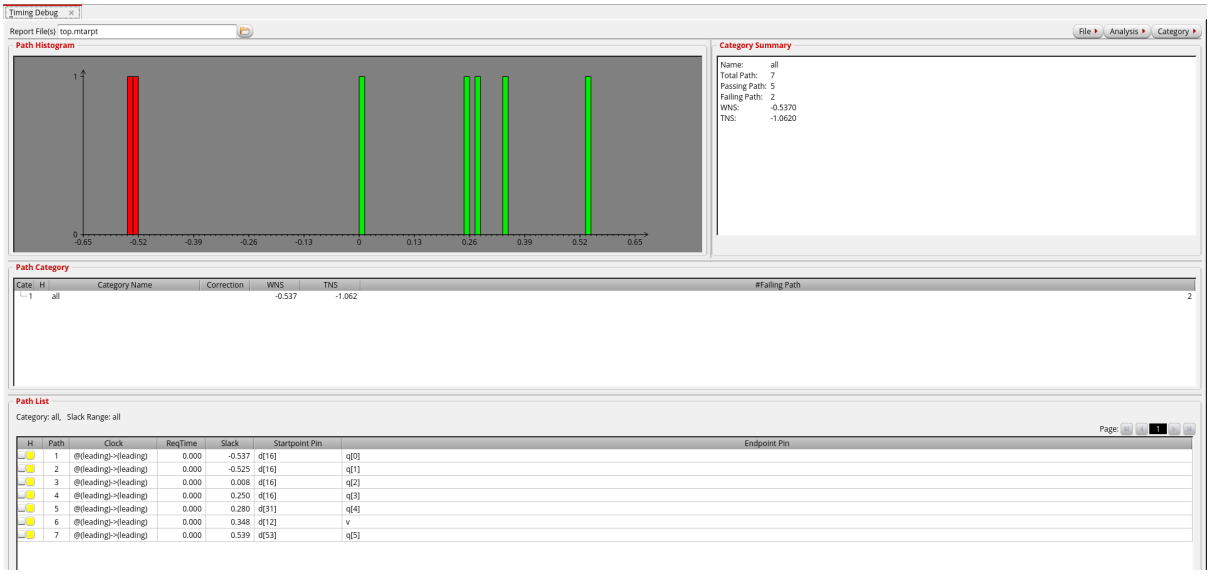


Figure 17: Genus timing debug report for MUX-based `pe64_lookahead` (180nm).

```
@genus:root: 4> report_timing
============================================================
Generated by:     Genus(TM) Synthesis Solution 20.11-s111_1
Generated on:     Oct 31 2025 02:55:27 pm
Module:           pe64_lookahead
...
============================================================

Path 1: VIOLATED (-537 ps) Path Delay Check
 Startpoint: (F) d[16]
 Endpoint: (F) q[0]
 ...
     Required Time:=     1200
     Data Path:-         1737
     Slack:=             -537
...
# Timing Point      Flags   Arc    Edge  Cell         ...  Delay Arrival
# ------------------------------------------------------------------------
  d[16]             -       -      F     (arrival)    ...  0     0
  g4066/Y           -       A->Y   R     CLKINVX8     ...  33    33
  g4044_6260/Y      -       B->Y   F     NAND2X4      ...  58    90
  g4037_5122/Y      -       A->Y   R     NOR3X4       ...  135   225
  g4018_8428/Y      -       B->Y   F     NAND2X4      ...  72    298
  g4005_5477/Y      -       B->Y   R     NOR2X2       ...  130   428
```

```
  g3981_2883/Y      -       C->Y   F     NAND3X2     ...  118   546
  g3962_7098/Y      -       A0N->Y F     AOI2BB1X2   ...  265   811
  g3955_2398/Y      -       A1N->Y F     OAI2BB1X4   ...  202   1013
  g3948_2346/Y      -       A->Y   R     NOR2X1      ...  196   1209
  g3918_3680/Y      -       A0->Y  F     AOI22XL     ...  108   1317
  g3901_1617/Y      -       D->Y   F     AND4X2      ...  268   1585
  g3892_6260/Y      -       B->Y   R     NAND3X1     ...  101   1686
  g3890_2398/Y      -       A->Y   F     NAND2XL     ...  52    1737
  q[0]              -       -      F     (port)      ...  0     1737
```

Power Report The total power consumption was **7.44867e-05 W**, or **74.49 μW**. This is dominated by **Switching power (63.09%)** and **Internal power (36.79%)**. Leakage remains negligible (0.12%). This power is higher than the unscalable 180nm design (53.83 μW).

```
@genus:root: 3> report_power
...
Instance: /pe64_lookahead
Power Unit: W
...
-----------------------------------------------------------------------
Category   Leakage      Internal     Switching    Total        Row%
-----------------------------------------------------------------------
...
logic      9.21942e-08  2.74025e-05  4.69919e-05  7.44867e-05  100.00%
...
-----------------------------------------------------------------------
Subtotal   9.21942e-08  2.74025e-05  4.69919e-05  7.44867e-05  100.00%
Percentage 0.12%       36.79%       63.09%       100.00%      100.00%
-----------------------------------------------------------------------
```

Area Report The 180nm MUX-based design required **191 cells** and a **Total Area of 2917.253 μM^2**. This is the largest area of all designs, even larger than the 180nm unscalable version (164 cells, 2012.472 μm^2).

```
@genus:root: 2> report_area
================================================================
Generated by:     Genus(TM) Synthesis Solution 20.11-s111_1
Generated on:     Oct 31 2025 02:52:24 pm
Module:           pe64_lookahead
...
================================================================
Instance      Module        Cell Count  Cell Area  Net Area  Total Area  Wireload
-------------------------------------------------------------------------------
pe64_lookahead                191         2917.253   0.000     2917.253 <none> (D)
```

7.4 Physical Implementation (Innovus)

The 180nm synthesized netlist for the MUX-based design was then passed to Cadence Innovus for physical implementation.

7.4.1 Innovus Netlist

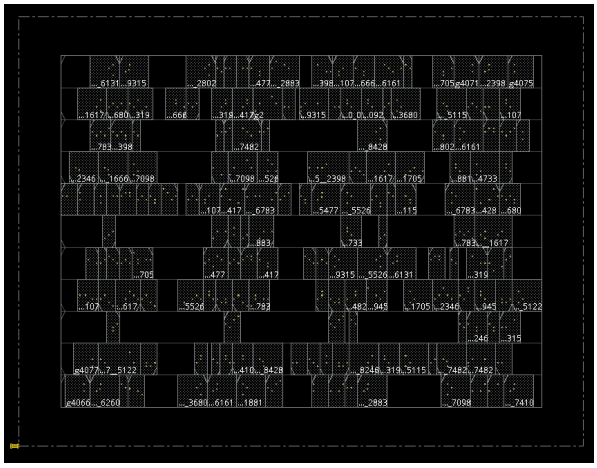
The netlist generated by Innovus after P&R is shown in Listing 16.

Listing 16: Innovus Netlist (`pe64_lookahead.v`) - 180nm MUX

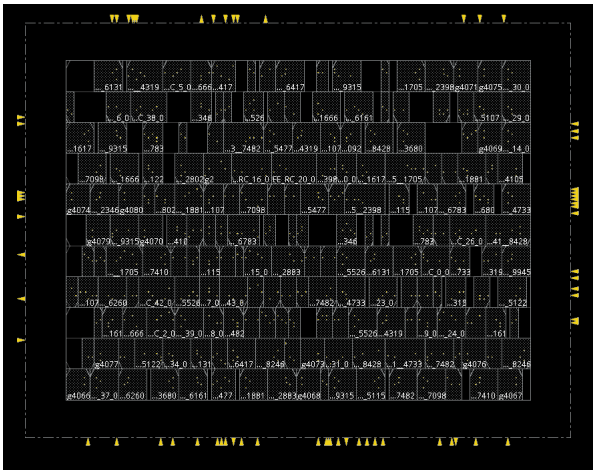
```
/*
##############################################################
#  Generated by:      Cadence Innovus 20.14-s095_1
#  OS:                Linux x86_64(Host ID cad28)
#  Generated on:      Fri Oct 31 15:56:34 2025
#  Design:            pe64_lookahead
#  Command:           saveNetlist pe64_lookahead.v
##############################################################
*/
// Generated by Cadence Genus(TM) Synthesis Solution 20.11-s111_1
...
module pe64_lookahead (
    d,
    q,
    v);
   input [63:0] d;
   output [5:0] q;
   output v;
...
   BUFX2 FE_OFC3_n_89 (.A(n_89),
    .Y(FE_OFN0_n_89));
...
   AOI2BB1X2 g2 (.A0N(n_283),
    .A1N(n_44),
    .B0(FE_OCPN7_n_63),
    .Y(n_290));
endmodule
```

7.4.2 Floorplan and Placement

Figure 18 shows the design's floorplan and cell placement for the 191 standard cells, both before and after nano-optimization.



(a) Floorplan and cell placement before optimization.

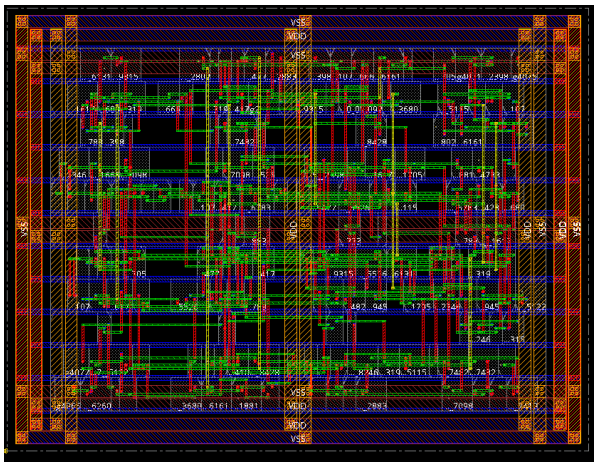


(b) Floorplan and cell placement after nano-optimization.

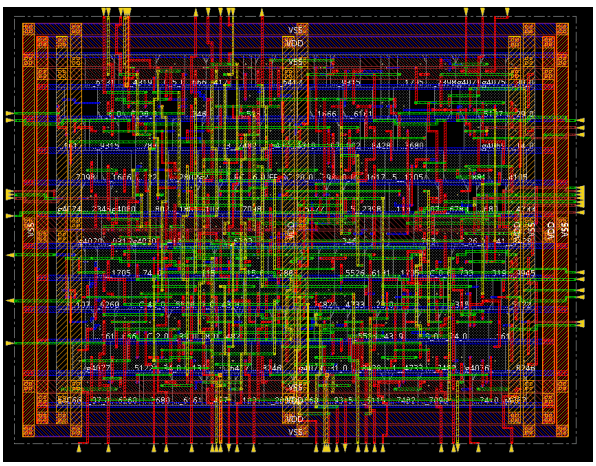
Figure 18: Innovus floorplan and placement views for MUX-based `pe64_lookahead` (180nm).

7.4.3 Final Layout (Post-Routing)

The final routed layout is shown in Figure 19. Figure 19a shows the layout before final optimization, and Figure 19b shows the denser routing after nano-optimization was performed.



(a) Layout after initial routing, before optimization.

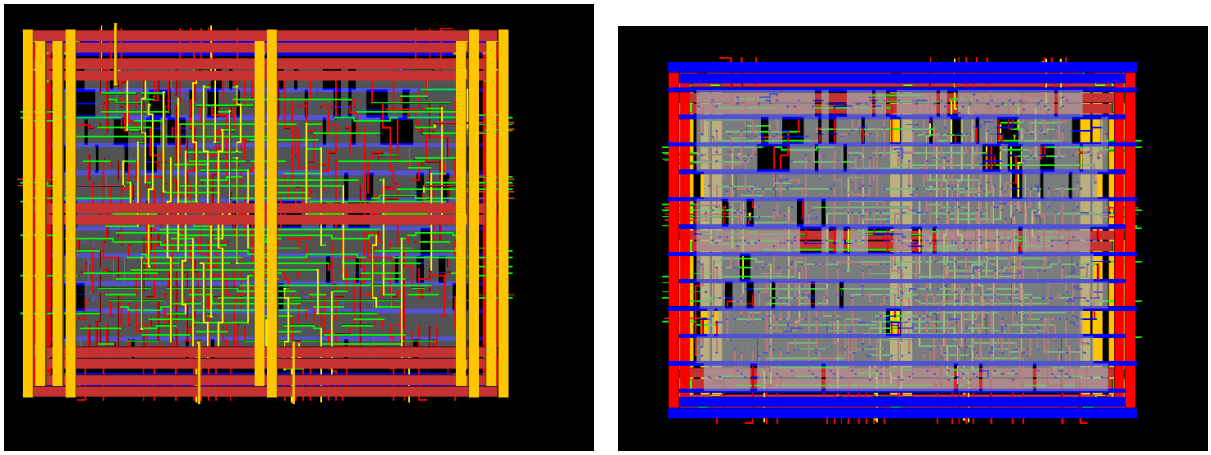


(b) Final layout after post-nano optimization.

Figure 19: Innovus final layout views for MUX-based `pe64_lookahead` (180nm).

7.4.4 3D Layout View

The 3D views in Figure 20 illustrate the physical stack-up of the metal layers for the 180nm MUX-based design.



(a) 3D View (Front). (b) 3D View (Back).

Figure 20: 3D layout visualization for MUX-based `pe64_lookahead` (180nm).

7.4.5 Post-Layout Reports

The following reports were generated by Innovus after place-and-route.

Pre-Optimization Timing Report The initial timing summary after placement shows the design failing timing significantly, with a **WNS of -1.144 ns** and 4 violating paths.

```
Initial Summary
--------------------------------------------------------------
Setup views included: wc

+-----------------+---------+
| Setup mode      | all     |
+-----------------+---------+
| WNS (ns):       | -1.144  |
| TNS (ns):       | -2.778  |
| Violating Paths:| 4       |
| All Paths:      | 7       |
+-----------------+---------+
...
Density: 65.692%
```

Pre-Optimization Area and Power Report Before optimization, the design had **174 cells** and a total power of **52.39 mW**.

```
@innovus 1> report_area
Hinst Name         Module Name     Inst Count     Total Area
--------------------------------------------------------------
pe64_lookahead                     174            2764.238

Total Power:                            0.05238977
 Total Internal Power:                  0.02728358   52.0781%
 Total Switching Power:                 0.02501791   47.7534%
 Total Leakage Power:                   0.00008829   0.1685%
...
Total instances in design: 174
```

Post-Optimization Timing Report After optimization, the design still **FAILED** timing. The optDesign summary shows a final **WNS of -0.701 ns**. The detailed nanoroute_timing.png report shows the critical path has a slack of -**0.743 ns**, resulting in a total path delay of **1.943 ns** (1.200 ns required + 0.743 ns violation). Hold timing passed with a WNS of 0.000 ns.

```
optDesign Final Summary
------------------------------------------------------------
...
+-----------------+---------+---------+
| Setup mode      | all     | default |
+-----------------+---------+---------+
| WNS (ns):       | -0.701  | -0.701  |
| TNS (ns):       | -1.618  | -1.618  |
| Violating Paths:| 3       | 3       |
| All Paths:      | 7       | 7       |
+-----------------+---------+---------+
Density: 86.719%
*** Finished optDesign ***

timeDesign Summary
------------------------------------------------------------
Hold views included: bc
...
+-----------------+-------+---------+
| Hold mode       | all   | default |
+-----------------+-------+---------+
| WNS (ns):       | 0.000 | 0.000   |
| TNS (ns):       | 0.000 | 0.000   |
| Violating Paths:| 0     | 0       |
| All Paths:      | 0     | 0       |
+-----------------+-------+---------+
Density: 86.719%
```

Post-Optimization Area Report To try and fix the timing violations, the tool inserted 25 new cells (199 - 174). The final **Instance Count is 199** and the **Total Area is 3649.061** μ**m**^{2}.

```
@innovus 4> report_area
Hinst Name        Module Name     Inst Count     Total Area
------------------------------------------------------------
pe64_lookahead                    199            3649.061
```

Post-Optimization Power Report With the addition of 25 buffers, the final power increased to **69.49 mW** (0.06949044 W).

```
Total Power:                          0.06949044
 Total Internal Power:                0.03750800   53.9759%
 Total Switching Power:               0.03184780   45.8305%
 Total Leakage Power:                 0.00013456   0.1936%
...
Total instances in design: 199
```

7.4.6 Final Verification and Routing Statistics

The final layout was checked for manufacturability and correctness.

Physical Verification (DRC & LVS) The design passed all physical checks with **0 DRC Violations** and **0 Connectivity Violations**.

```
#Start DRC checking..
...
#Total number of DRC violations = 0
...
#Total number of process antenna violations = 0
...
Verification Complete : 0 Viols.
```

Final Routing Statistics The post-routing report confirms the final routing statistics. The **Total Wire Length is 4705 μm**, and the **Total Number of Vias is 1026**. The `gdsi_file_summary.png` confirms the final instance count of 199.

```
#Post Route wire spread is done.
#Total wire length = 4705 um.
#Total half perimeter of net bounding box = 5300 um.
#Total wire length on LAYER Metal1 = 191 um.
#Total wire length on LAYER Metal2 = 1866 um.
#Total wire length on LAYER Metal3 = 1750 um.
#Total wire length on LAYER Metal4 = 762 um.
#Total wire length on LAYER Metal5 = 103 um.
#Total wire length on LAYER Metal6 = 32 um.
#Total number of vias = 1026
```

```
Stream Out Information Processed for GDS version 3:
Units: 2000 DBU
Object         Count
--------------------
Instances      199
...
Blockages      0
```

7.5 Comparative Analysis

7.5.1 Consolidated Results Table

Table 6 summarizes the synthesis (Genus) and post-layout (Innovus) results for all implemented designs. The "Critical Path Delay" is calculated by subtracting the WNS from the 1.2 ns target constraint.

Table 3: Comparative Analysis of 64-bit Priority Encoder Implementations

Architecture	Tech. Node	Analysis Stage	Cell Count	Total Area (μm^2)	Total Power	WNS (ns) @ 1.2ns	Critical Path Delay (ns)
Unscalable (pe64_if_else)	180nm	Genus	164	2012.472	53.83 μW	0.039	1.161
		Innovus	170	2098.958	43.86 mW	0.003	1.197
Unscalable (pe64_if_else)	90nm	Genus	139	595.680	14.42 μW	0.280	0.920
		Innovus	139	595.680	19.93 mW	0.910	0.290
MUX (pe64_lookahead)	180nm	Genus	191	2917.253	74.49 μW	-0.537	1.737
		Innovus	199	3649.061	69.49 mW	-0.743	1.943

8 Scalable 64-bit Priority Encoder on 180nm Technology

This section details the implementation of the primary high-performance architecture, `pe64_lookahead`, which is based on the 1D-to-2D array conversion method. This design uses a `casex` statement to create efficient, priority-based look-ahead logic for column selection. This architecture was implemented using the **180nm CMOS technology** library to provide a direct comparison with the unscalable and MUX-based 180nm designs.

8.1 RTL Design

The complete Verilog code is shown in Listing 17. The file defines the full scalable hierarchy, from pe4 up to `pe256_scalable`. The `pe64_lookahead` module is the core of this implementation, using a priority `casex` statement to select the column data based on the `dor` (row status) vector.

Listing 17: Scalable Look-ahead PE Verilog Code (`priority_encoder.v`)

```
module pe4 (
    input  wire [3:0] d,
    output wire [1:0] q,
    output wire       v
);
    assign q[1] = d[3] | d[2];
    assign q[0] = d[3] | (d[1] & ~d[2]);
    assign v = |d;
endmodule

module pe16 (
    input  wire [15:0] d,
    output wire [3:0]  q,
    output wire        v
);
    wire [3:0] row_status;
    reg  [3:0] selected_row;
    wire [1:0] row_index;
    wire [1:0] col_index;
    wire       row_valid;

    assign row_status[0] = |d[3:0];
    assign row_status[1] = |d[7:4];
    assign row_status[2] = |d[11:8];
    assign row_status[3] = |d[15:12];

    pe4 row_pe (
        .d(row_status),
        .q(row_index),
        .v(row_valid)
    );

    always @(*) begin
        case (row_index)
            2'b00: selected_row = d[3:0];
```

```
            2'b01: selected_row = d[7:4];
            2'b10: selected_row = d[11:8];
            2'b11: selected_row = d[15:12];
            default: selected_row = 4'b0000;
        endcase
    end

    pe4 col_pe (
        .d(selected_row),
        .q(col_index),
        .v()
    );

    assign q = {row_index, col_index};
    assign v = row_valid;
endmodule

module pe64_lookahead (
    input  wire [63:0] d,
    output wire [5:0]  q,
    output wire        v
);
    wire [15:0] dor;
    wire [3:0]  row_index;
    wire [3:0]  column_data;
    wire [1:0]  col_index;
    wire        row_valid;

    genvar i;
    generate
        for (i = 0; i < 16; i = i + 1) begin : row_or_logic
            assign dor[i] = |d[(i*4)+3 : i*4];
        end
    endgenerate

    pe16 row_encoder (
        .d(dor),
        .q(row_index),
        .v(row_valid)
    );

    reg [3:0] column_data_reg;
    always @(*) begin
        casex(dor)
            16'b1xxxxxxxxxxxxxxx: column_data_reg = d[63:60];
            16'b01xxxxxxxxxxxxxx: column_data_reg = d[59:56];
            16'b001xxxxxxxxxxxxx: column_data_reg = d[55:52];
            16'b0001xxxxxxxxxxxx: column_data_reg = d[51:48];
            16'b00001xxxxxxxxxxx: column_data_reg = d[47:44];
            16'b000001xxxxxxxxxx: column_data_reg = d[43:40];
            16'b0000001xxxxxxxxx: column_data_reg = d[39:36];
```

```
87              16'b00000001xxxxxxxx: column_data_reg = d[35:32];
88              16'b000000001xxxxxxx: column_data_reg = d[31:28];
89              16'b0000000001xxxxxx: column_data_reg = d[27:24];
90              16'b00000000001xxxxx: column_data_reg = d[23:20];
91              16'b000000000001xxxx: column_data_reg = d[19:16];
92              16'b0000000000001xxx: column_data_reg = d[15:12];
93              16'b00000000000001xx: column_data_reg = d[11:8];
94              16'b000000000000001x: column_data_reg = d[7:4];
95              16'b0000000000000001: column_data_reg = d[3:0];
96              default:              column_data_reg = 4'b0;
97          endcase
98      end
99
100     assign column_data = column_data_reg;
101
102     pe4 col_encoder (
103         .d(column_data),
104         .q(col_index),
105         .v()
106     );
107
108     assign q = {row_index, col_index};
109     assign v = row_valid;
110 endmodule
111
112 module pe256_scalable (
113     input  wire [255:0] d,
114     output wire [7:0]   q,
115     output wire         v
116 );
117     wire [3:0] block_status;
118     wire [1:0] block_index;
119     wire [5:0] internal_index;
120     wire       block_valid, internal_valid;
121
122     assign block_status[0] = |d[63:0];
123     assign block_status[1] = |d[127:64];
124     assign block_status[2] = |d[191:128];
125     assign block_status[3] = |d[255:192];
126
127     pe4 block_selector (
128         .d(block_status),
129         .q(block_index),
130         .v(block_valid)
131     );
132
133     wire [63:0] selected_block;
134     assign selected_block = (block_index == 2'b00) ? d[63:0]
        :
135                             (block_index == 2'b01) ? d[127:64]
                                :
```

```
                            (block_index == 2'b10) ? d[191:128]
                               :
                            (block_index == 2'b11) ? d[255:192]
                               : 64'b0;

    pe64_lookahead internal_pe (
        .d(selected_block),
        .q(internal_index),
        .v(internal_valid)
    );

    assign q = {block_index, internal_index};
    assign v = block_valid;
endmodule
```

8.2 RTL Verification

8.2.1 Testbench Code

A self-checking testbench (Listing 18) was used to verify the pe256_scalable module. This testbench is functionally identical to the ones used for the other designs.

Listing 18: Self-checking testbench for pe256_scalable (180nm)

```
`timescale 1ns / 1ps
`include "priority_encoder.v"

module tb_pe256_scalable_selfchecking();

    reg  [255:0] tb_d;
    wire [7:0]   tb_q;
    wire         tb_v;

    integer error_count;
    reg [7:0] expected_q;
    integer i;

    pe256_scalable dut (
        .d(tb_d),
        .q(tb_q),
        .v(tb_v)
    );

    function [7:0] get_expected_q(input [255:0] d);
        integer i;
        begin : find_bit_block
            get_expected_q = 8'b0;
            for (i = 255; i >= 0; i = i - 1) begin
                if (d[i] == 1'b1) begin
                    get_expected_q = i;
                    disable find_bit_block;
                end
            end
```

```
30          end
31      endfunction
32
33      initial begin
34          $dumpfile("pe256.vcd");
35          $dumpvars(0, tb_pe256_scalable_selfchecking);
36          error_count = 0;
37
38          tb_d = 256'b0;
39          #10;
40          if (tb_v !== 1'b0) begin
41              $display("FAIL: All-zero input, ...");
42              error_count = error_count + 1;
43          end
44
45          for (i = 0; i < 256; i = i + 1) begin
46              tb_d = 0;
47              tb_d[i] = 1'b1;
48              expected_q = i;
49              #10;
50              if (tb_q !== expected_q || tb_v !== 1'b1) begin
51                  $display("FAIL: Input bit %0d | ...", i,
                        expected_q, tb_q);
52                  error_count = error_count + 1;
53              end
54          end
55
56          tb_d = 0;
57          tb_d[123] = 1'b1;
58          tb_d[200] = 1'b1;
59          tb_d[5]   = 1'b1;
60          expected_q = 200;
61          #10;
62          if (tb_q !== expected_q) begin
63              $display("FAIL: Multi-bit test | ...", expected_q,
                    tb_q);
64              error_count = error_count + 1;
65          end
66
67          for (i = 0; i < 20; i = i + 1) begin
68              tb_d = {$random, $random, $random, $random,
69                      $random, $random, $random, $random};
70              expected_q = get_expected_q(tb_d);
71              #10;
72              if (tb_q !== expected_q) begin
73                  $display("FAIL: Random test | ...", tb_d,
                        expected_q, tb_q);
74                  error_count = error_count + 1;
75              end
76          end
77
```

```
78          #20;
79          if (error_count == 0) begin
80              $display("SUCCESS: All tests passed!");
81          end else begin
82              $display("FAILURE: %0d errors found.", error_count);
83          end
84
85          $finish;
86      end
87  endmodule
```

8.2.2 Synthesized Schematic

The resulting gate-level schematic synthesized by Genus is shown in Figure 21.

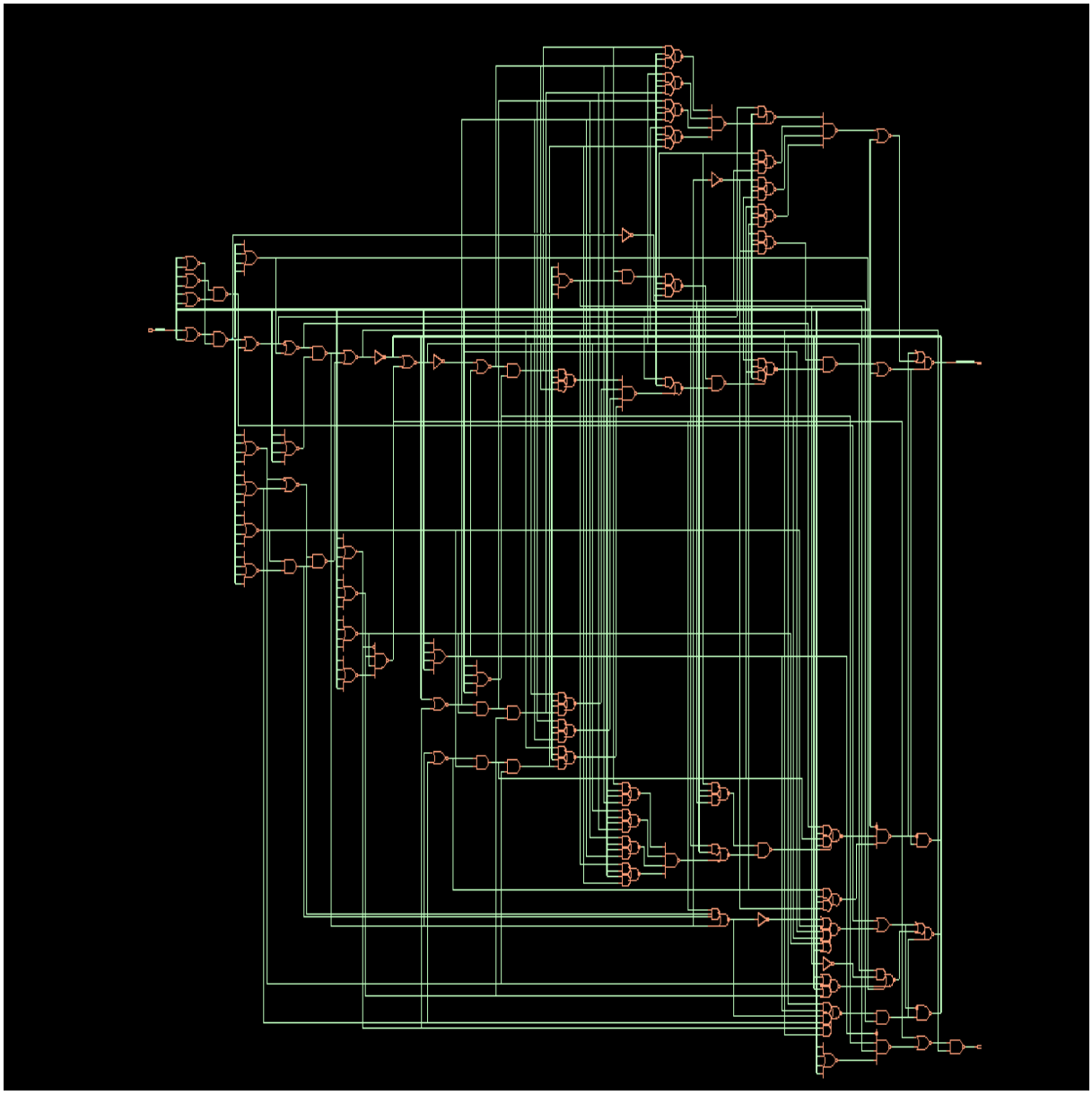
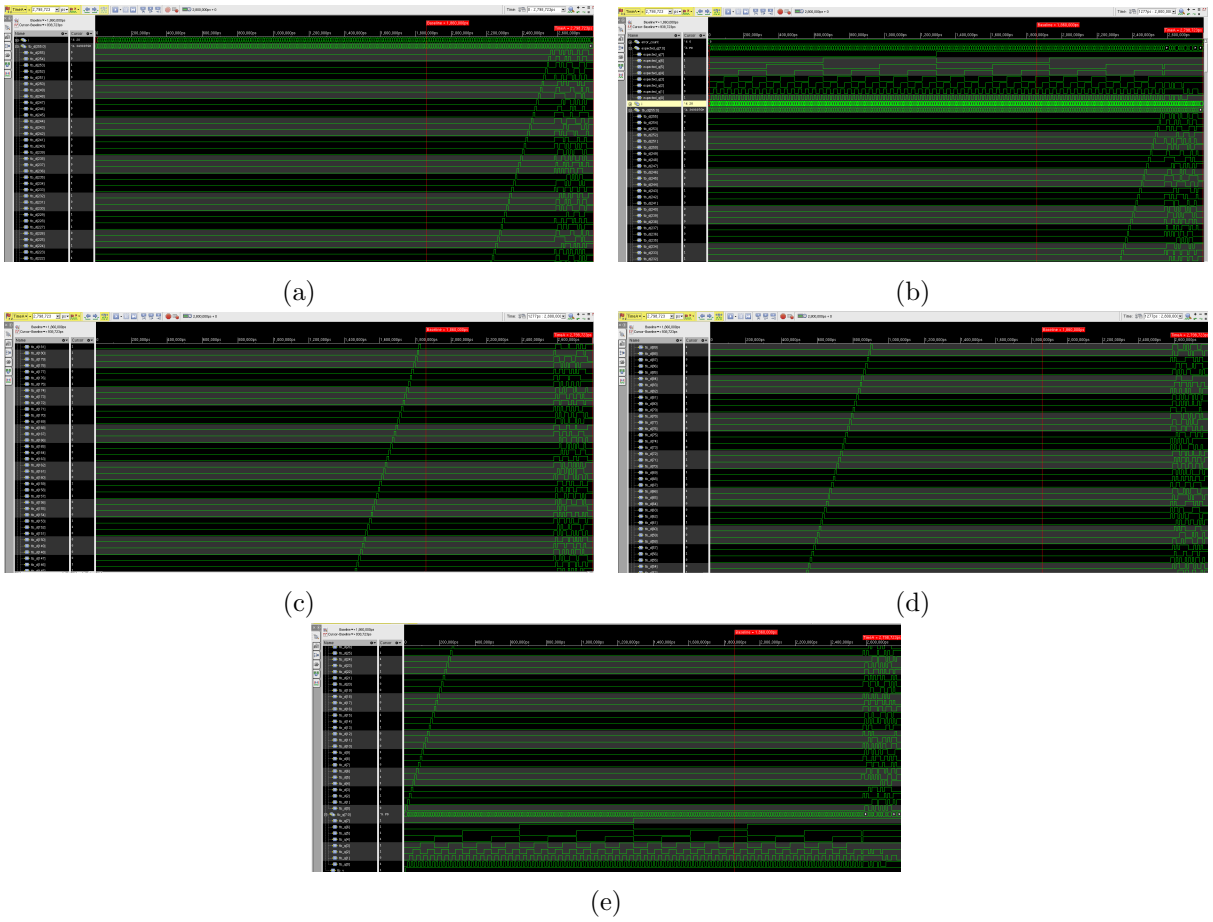


Figure 21: Synthesized schematic of look-ahead `pe64_lookahead` (180nm).

8.2.3 Simulation Waveforms

The functionality of the synthesized 180nm netlist was verified in Cadence NCLaunch.



(a) (b) (c) (d) (e)

Figure 22: Simulation waveforms verifying the synthesized 180nm scalable look-ahead `pe64_lookahead` netlist: (a) Verification of bits 21-55, (b) Expected vs. actual output, (c) Verification of bits 45-81, (d) Verification of bits 53-89, and (e) Verification of bits 1-37.

8.3 Logic Synthesis (Genus)

The `pe64_lookahead` module was synthesized using Cadence Genus with the 180nm slow-corner library.

8.3.1 Synthesis Scripts

The synthesis process was guided by the Tcl script in Listing 19. The same 1.2 ns timing constraint (Listing 20) was used. The output SDC file is shown in Listing 21.

Listing 19: Genus synthesis script (`run.tcl`) - 180nm

```
set_db init_lib_search_path /home/install/FOUNDRY/digital/180nm/
   dig/lib/
set_db library slow.lib

read_hdl {./priority_encoder.v}

elaborate pe64_lookahead
read_sdc ./constraints_input.sdc
```

```

set_db syn_generic_effort medium
set_db syn_map_effort medium
set_db syn_opt_effort medium

syn_generic
syn_map
syn_opt

write_hdl > priority_netlist.v
write_sdc > priority_output.sdc

report timing > priority_timing.rpt
report power > priority_power.rpt
report area  > priority_cell.rpt
report gates > priority_gates.rpt

gui_show
```

Listing 20: Synthesis design constraints (`constraints_input.sdc`) - 180nm

```
set_max_delay 1.2 -from [get_ports d[]] -to [get_ports {q[] v}]
```

Listing 21: Genus output constraints (`priority_output.sdc`) - 180nm

```
#
   ##############################################################################
#  Created by Genus(TM) Synthesis Solution 20.11-s111_1 on Fri
   Oct 10 13:45:57 IST 2025
#
   ##############################################################################


set sdc_version 2.0
set_units -capacitance 1000fF
set_units -time 1000ps

# Set the current design
current_design pe64_lookahead
...
set_max_delay 1.2 -from [list \
  [get_ports {d[63]}]  \
  [get_ports {d[62]}]  \
...
  [get_ports {d[1]}]  \
  [get_ports {d[0]}] ] -to [get_ports v]
set_clock_gating_check -setup 0.0
set_wire_load_mode "enclosed"
```

8.3.2 Synthesized Netlist

The following is the gate-level netlist (Listing 22) generated by Genus for the 180nm scalable design.

Listing 22: Synthesized Netlist (`priority_netlist.v`) - 180nm

```
// Generated by Cadence Genus(TM) Synthesis Solution 20.11-s111_1
// Generated on: Oct 10 2025 13:45:57 IST (Oct 10 2025 08:15:57
    UTC)
// Verification Directory fv/pe64_lookahead
module pe64_lookahead (
    d,
    q,
    v);
   input [63:0] d;
   output [5:0] q;
   output v;
...
   NAND2BXL g2789__2398(.AN (n_93),
    .B(n_92),
    .Y(q[1]));
...
   OR4X2 g2884 (.A(d[54]),
    .B(d[53]),
    .C(d[55]),
    .D(d[52]),
    .Y(n_161));
endmodule
```

8.3.3 Synthesis Reports

The following reports were generated by Genus for the 180nm look-ahead design.

Timing Report The 180nm look-ahead design successfully **MET** the 1.2 ns timing constraint. The GUI report in Figure 23 shows 0 failing paths and a final **WNS of 0.011 ns** (or 11 ps). This indicates a critical path delay of **1.189 ns** (1.2 ns - 0.011 ns).

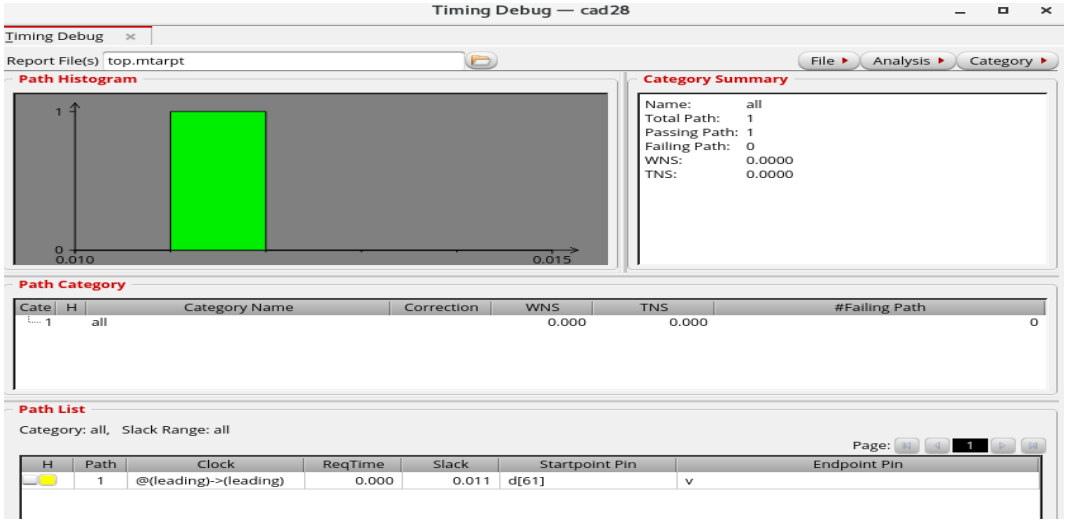


Figure 23: Genus timing debug report for look-ahead `pe64_lookahead` (180nm).

Power Report The total power consumption was **2.14454e-05 W**, or **21.45 μW**. This is the lowest power consumption of all 180nm designs. It is dominated by **Switching power (71.68%)** and **Internal power (28.16%)**.

```
@genus:root: 3> report_power
...
Instance: /pe64_lookahead
Power Unit: W
...
-------------------------------------------------------------------------
Category   Leakage      Internal     Switching    Total        Row%
-------------------------------------------------------------------------
...
logic      3.27079e-08  6.03971e-06  1.53729e-05  2.14454e-05  100.00%
...
-------------------------------------------------------------------------
Subtotal   3.27079e-08  6.03971e-06  1.53729e-05  2.14454e-05  100.00%
Percentage 0.15%       28.16%       71.68%       100.00%      100.00%
-------------------------------------------------------------------------
```

Area Report The 180nm look-ahead design was the most area-efficient, requiring only **92 cells** and a **Total Area of 1373.803 μm^2**.

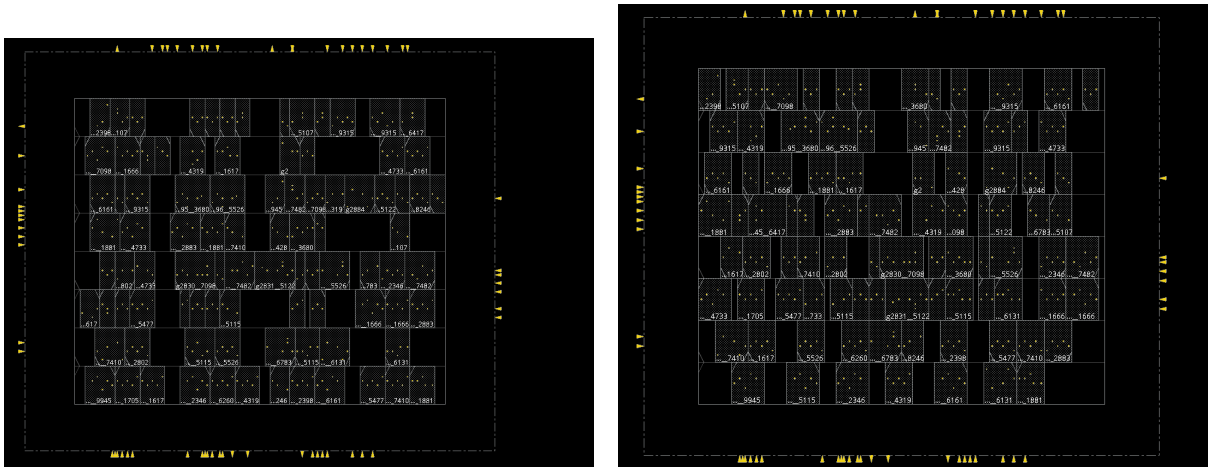
```
@genus:root: 2> report_area
====================================================================
Generated by:     Genus(TM) Synthesis Solution 20.11-s111_1
Generated on:     Oct 10 2025 01:46:15 pm
Module:           pe64_lookahead
...
====================================================================
Instance      Module        Cell Count  Cell Area  Net Area  Total Area  Wireload
-----------------------------------------------------------------------------------
pe64_lookahead                92          1373.803   0.000     1373.803 <none> (D)
```

8.4 Physical Implementation (Innovus)

The 180nm synthesized netlist for the look-ahead design was then passed to Cadence Innovus for physical implementation.

8.4.1 Floorplan and Placement

Figure 24 shows the design's floorplan and cell placement for the 92 standard cells, both before and after nano-optimization.



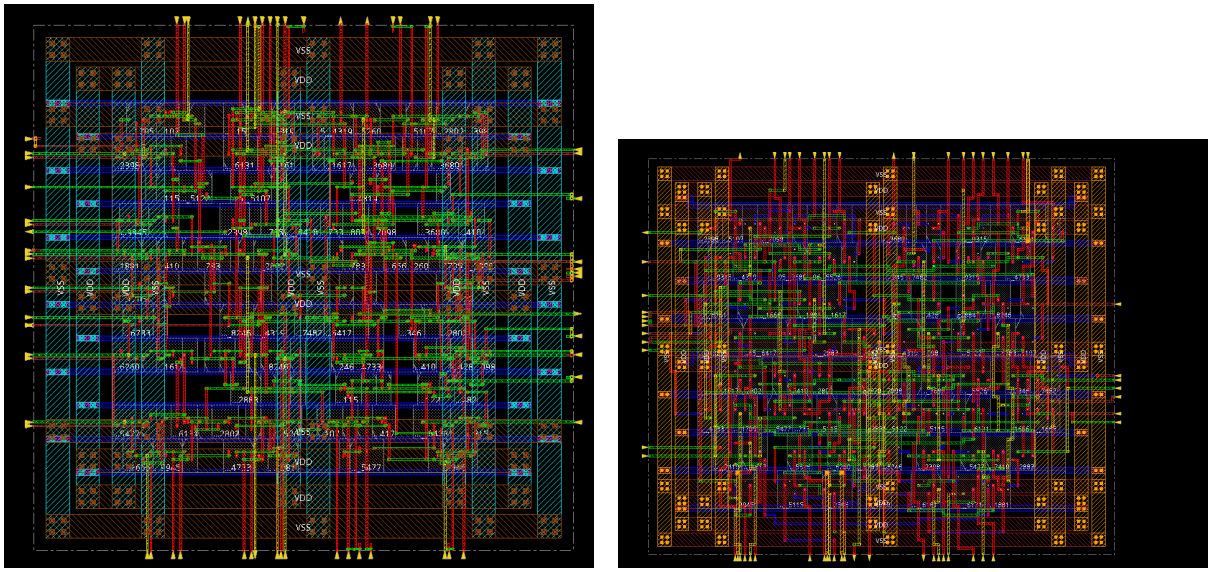
(a) Floorplan and cell placement before optimization.

(b) Floorplan and cell placement after nano-optimization.

Figure 24: Innovus floorplan and placement views for look-ahead `pe64_lookahead` (180nm).

8.4.2 Final Layout (Post-Routing)

The final routed layout is shown in Figure 25. Figure 25a shows the layout before final optimization, and Figure 25b shows the layout after nano-optimization.



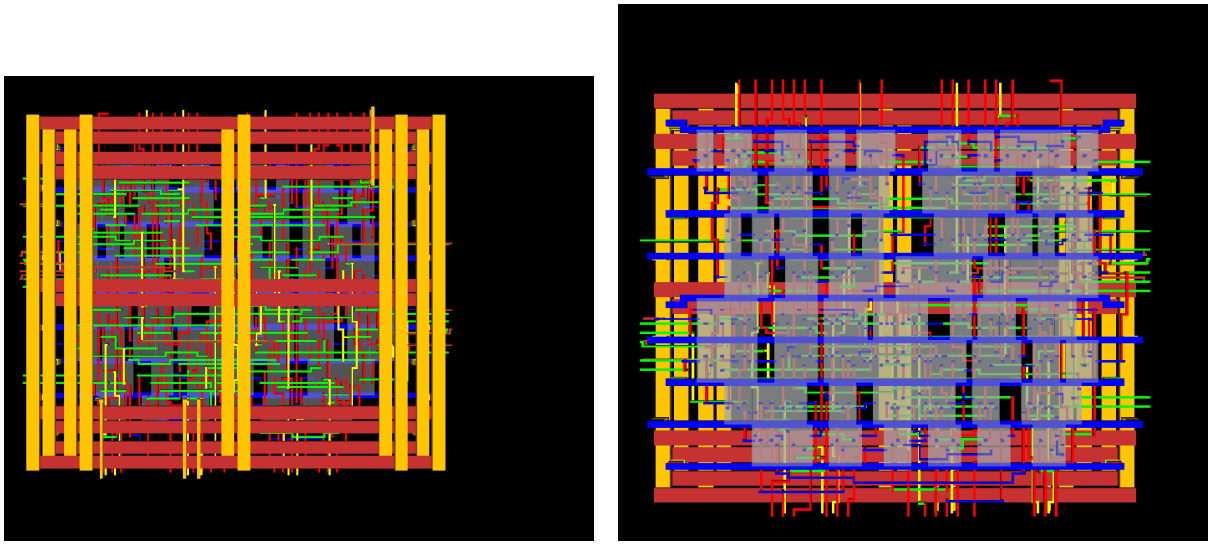
(a) Layout after initial routing, before optimization.

(b) Final layout after post-nano optimization.

Figure 25: Innovus final layout views for look-ahead `pe64_lookahead` (180nm).

8.4.3 3D Layout View

The 3D views in Figure 26 illustrate the physical stack-up of the metal layers for the 180nm look-ahead design.



(a) 3D View (Front). (b) 3D View (Back).

Figure 26: 3D layout visualization for look-ahead `pe64_lookahead` (180nm).

8.4.4 Pre-Optimization Reports (Initial Placement)

After the initial design import and cell placement, the tool generated preliminary reports.

Pre-Placement Timing Report The initial timing analysis shows the design **FAILED** the 1.2 ns constraint. The Worst Negative Slack (WNS) was **-0.277 ns**. This is expected, as the Genus synthesis (WNS 0.011 ns) was based on wireload estimates, not actual placement.

```
timeDesign Summary
------------------------------------------------------------
Setup views included: wc
...
+-----------------+---------+---------+
| Setup mode      | all     | default |
+-----------------+---------+---------+
| WNS (ns):       | -0.277  | -0.277  |
| TNS (ns):       | -0.277  | -0.277  |
| Violating Paths:| 1       | 1       |
| All Paths:      | 1       | 1       |
+-----------------+---------+---------+
...
Density: 69.764%
```

Pre-Placement Area Report The initial area report shows an **Instance Count of 92** and a **Total Area of 1373.803** μ**m**^{2}, matching the post-synthesis report from Genus.

```
@innovus 1> report_area
Hinst Name        Module Name     Inst Count     Total Area
```

```
------------------------------------------------------------
pe64_lookahead                    92             1373.803
```

Pre-Placement Power Report The initial power estimate after placement was **10.37 mW** (0.01036668 W). This is significantly different from the Genus estimate, with **Internal Power (58.54%)** and **Switching Power (41.14%)** being the main components.

```
Total Power:                            0.01036668
 Total Internal Power:                  0.00606928   58.5460%
 Total Switching Power:                 0.00426469   41.1385%
 Total Leakage Power:                   0.00003271   0.3155%
...
Group            Internal   Switching  Leakage    Total      Percentage (%)
...
Combinational    0.006069   0.004265   3.271e-05  0.01037    100
...
Total instances in design: 92
```

8.4.5 Post-Route Optimization Reports

The tool next performed `optDesign`, which routes the design and inserts buffers to fix timing violations.

Post-Optimization Timing Report The `optDesign` summary shows that the optimization was successful. The design now **meets timing**, with a final Worst Negative Slack (WNS) of **0.013 ns**. The final `timeDesign` summary confirms this, with a final **Setup WNS of 0.016 ns** and 0 violating paths. This indicates a final critical path delay of **1.184 ns** (1.2 ns - 0.016 ns).

```
optDesign Final Summary
------------------------------------------------------------
...
+-----------------+-------+---------+
| Setup mode      | all   | default |
+-----------------+-------+---------+
| WNS (ns):       | 0.013 | 0.013   |
| TNS (ns):       | 0.000 | 0.000   |
| Violating Paths:| 0     | 0       |
| All Paths:      | 1     | 1       |
+-----------------+-------+---------+
...
Density: 70.439%
*** Finished optDesign ***
```

Post-Optimization Area Report The post-optimization area report shows the **Instance Count (92)** did not change, but the **Total Area increased slightly to 1387.109 μm^2**. This indicates the tool met timing by swapping cells for larger, faster equivalents rather than adding new buffers.

```
@innovus 3> report_area
Hinst Name        Module Name     Inst Count     Total Area
------------------------------------------------------------
pe64_lookahead                    92             1387.109
```

Post-Optimization Power Report The final power report after nano-routing shows a total power of **10.42 mW** (0.01041789 W). This is a minor increase from the pre-placement estimate.

```
Total Power:                              0.01041789
 Total Internal Power:                    0.00585967    56.2462%
 Total Switching Power:                   0.00452499    43.4348%
 Total Leakage Power:                     0.00003323    0.3190%
...
Total instances in design: 92
```

8.4.6 Final Verification and Routing Statistics

Finally, the design was verified for manufacturability (DRC) and correctness (LVS/Connectivity).

Physical Verification (DRC & LVS) The design passed all physical checks with **0 DRC Violations** and **0 Connectivity Violations**, confirming the layout is correct and manufacturable.

```
@innovus 3> # check_drc ...
*** Starting Verify DRC (MEM: 1601.2) ***
...
VERIFY DRC ..... Sub-Area : 1 complete 0 Viols.

Verification Complete : 0 Viols.
```

```
@innovus 3> VERIFY_CONNECTIVITY use new engine.
********* Start: VERIFY CONNECTIVITY *********
Design Name: pe64_lookahead
...
Begin Summary
Found no problems or warnings.
End Summary
...
Verification Complete : 0 Viols. 0 Wrngs.
```

Final Routing Statistics The post-routing report confirms the final routing statistics. The **Total Wire Length is 2926 μm**, and the **Total Number of Vias is 680**.

```
#Post Route wire spread is done.
#Total wire length = 2926 um.
#Total half perimeter of net bounding box = 2948 um.
#Total wire length on LAYER Metal1 = 240 um.
#Total wire length on LAYER Metal2 = 1213 um.
#Total wire length on LAYER Metal3 = 931 um.
#Total wire length on LAYER Metal4 = 390 um.
#Total wire length on LAYER Metal5 = 110 um.
#Total wire length on LAYER Metal6 = 43 um.
#Total number of vias = 680
#Up-Via Summary (total 680):
#-
# Metal1         361
# Metal2         231
# Metal3         69
# Metal4         15
# Metal5         4
```

8.5 Comparative Analysis

8.5.1 Consolidated Results Table

Table 6 summarizes the synthesis (Genus) and post-layout (Innovus) results for all implemented designs. The "Critical Path Delay" is calculated by subtracting the WNS from the 1.2 ns target constraint.

Table 4: Comparative Analysis of 64-bit Priority Encoder Implementations

Architecture	Tech. Node	Analysis Stage	Cell Count	Total Area (μm^2)	Total Power	WNS (ns) @ 1.2ns	Critical Path Delay (ns)
Unscalable (`pe64_if_else`)	180nm	Genus	164	2012.472	53.83 μW	0.039	1.161
		Innovus	170	2098.958	43.86 mW	0.003	1.197
Unscalable (`pe64_if_else`)	90nm	Genus	139	595.680	14.42 μW	0.280	0.920
		Innovus	139	595.680	19.93 mW	0.910	0.290
Scalable MUX (`pe64_lookahead`)	180nm	Genus	191	2917.253	74.49 μW	-0.537	1.737
		Innovus	199	3649.061	69.49 mW	-0.743	1.943
Scalable Look-ahead (`pe64_lookahead`)	180nm	Genus	92	1373.803	21.45 μW	0.011	1.189
		Innovus	92	1387.109	10.42 mW	0.016	1.184

9 Scalable 64-bit Priority Encoder on 90nm Technology

This section details the implementation of the primary high-performance architecture, `pe64_lookahead`, which is based on the 1D-to-2D array conversion method. This design uses a `casex` statement to create efficient, priority-based look-ahead logic for column selection. This architecture was implemented using the **90nm CMOS technology** library to provide a direct comparison with the unscalable 180nm designs.

9.1 RTL Design

The complete Verilog code is shown in Listing 23. The file defines the full scalable hierarchy, from pe4 up to `pe256_scalable`. The `pe64_lookahead` module is the core of this implementation, using a priority `casex` statement to select the column data based on the `dor` (row status) vector.

Listing 23: Scalable Look-ahead PE Verilog Code (`priority_encoder.v`)

```
module pe4 (
    input  wire [3:0] d,
    output wire [1:0] q,
    output wire       v
);
    assign q[1] = d[3] | d[2];
    assign q[0] = d[3] | (d[1] & ~d[2]);
    assign v = |d;
endmodule

module pe16 (
    input  wire [15:0] d,
    output wire [3:0]  q,
    output wire        v
);
    wire [3:0] row_status;
    reg  [3:0] selected_row;
    wire [1:0] row_index;
    wire [1:0] col_index;
    wire       row_valid;

    assign row_status[0] = |d[3:0];
    assign row_status[1] = |d[7:4];
    assign row_status[2] = |d[11:8];
    assign row_status[3] = |d[15:12];

    pe4 row_pe (
        .d(row_status),
        .q(row_index),
        .v(row_valid)
    );

    always @(*) begin
        case (row_index)
            2'b00: selected_row = d[3:0];
```

```
            2'b01: selected_row = d[7:4];
            2'b10: selected_row = d[11:8];
            2'b11: selected_row = d[15:12];
            default: selected_row = 4'b0000;
        endcase
    end

    pe4 col_pe (
        .d(selected_row),
        .q(col_index),
        .v()
    );

    assign q = {row_index, col_index};
    assign v = row_valid;
endmodule

module pe64_lookahead (
    input  wire [63:0] d,
    output wire [5:0]  q,
    output wire        v
);
    wire [15:0] dor;
    wire [3:0]  row_index;
    wire [3:0]  column_data;
    wire [1:0]  col_index;
    wire        row_valid;

    genvar i;
    generate
        for (i = 0; i < 16; i = i + 1) begin : row_or_logic
            assign dor[i] = |d[(i*4)+3 : i*4];
        end
    endgenerate

    pe16 row_encoder (
        .d(dor),
        .q(row_index),
        .v(row_valid)
    );

    reg [3:0] column_data_reg;
    always @(*) begin
        casex(dor)
            16'b1xxxxxxxxxxxxxxx: column_data_reg = d[63:60];
            16'b01xxxxxxxxxxxxxx: column_data_reg = d[59:56];
            16'b001xxxxxxxxxxxxx: column_data_reg = d[55:52];
            16'b0001xxxxxxxxxxxx: column_data_reg = d[51:48];
            16'b00001xxxxxxxxxxx: column_data_reg = d[47:44];
            16'b000001xxxxxxxxxx: column_data_reg = d[43:40];
            16'b0000001xxxxxxxxx: column_data_reg = d[39:36];
```

```
87              16'b00000001xxxxxxxx: column_data_reg = d[35:32];
88              16'b000000001xxxxxxx: column_data_reg = d[31:28];
89              16'b0000000001xxxxxx: column_data_reg = d[27:24];
90              16'b00000000001xxxxx: column_data_reg = d[23:20];
91              16'b000000000001xxxx: column_data_reg = d[19:16];
92              16'b0000000000001xxx: column_data_reg = d[15:12];
93              16'b00000000000001xx: column_data_reg = d[11:8];
94              16'b000000000000001x: column_data_reg = d[7:4];
95              16'b0000000000000001: column_data_reg = d[3:0];
96              default:              column_data_reg = 4'b0;
97          endcase
98      end
99
100     assign column_data = column_data_reg;
101
102     pe4 col_encoder (
103         .d(column_data),
104         .q(col_index),
105         .v()
106     );
107
108     assign q = {row_index, col_index};
109     assign v = row_valid;
110 endmodule
111
112 module pe256_scalable (
113     input  wire [255:0] d,
114     output wire [7:0]   q,
115     output wire         v
116 );
117     wire [3:0] block_status;
118     wire [1:0] block_index;
119     wire [5:0] internal_index;
120     wire       block_valid, internal_valid;
121
122     assign block_status[0] = |d[63:0];
123     assign block_status[1] = |d[127:64];
124     assign block_status[2] = |d[191:128];
125     assign block_status[3] = |d[255:192];
126
127     pe4 block_selector (
128         .d(block_status),
129         .q(block_index),
130         .v(block_valid)
131     );
132
133     wire [63:0] selected_block;
134     assign selected_block = (block_index == 2'b00) ? d[63:0]
        :
135                             (block_index == 2'b01) ? d[127:64]
                                :
```

```
                            (block_index == 2'b10) ? d[191:128]
                               :
                            (block_index == 2'b11) ? d[255:192]
                               : 64'b0;

    pe64_lookahead internal_pe (
        .d(selected_block),
        .q(internal_index),
        .v(internal_valid)
    );

    assign q = {block_index, internal_index};
    assign v = block_valid;
endmodule
```

9.2 RTL Verification

9.2.1 Testbench Code

A self-checking testbench (Listing 24) was used to verify the pe256_scalable module. This testbench is functionally identical to the ones used for the other designs.

Listing 24: Self-checking testbench for pe256_scalable (90nm)

```
`timescale 1ns / 1ps
`include "priority_encoder.v"

module tb_pe256_scalable_selfchecking();

    reg  [255:0] tb_d;
    wire [7:0]   tb_q;
    wire         tb_v;

    integer error_count;
    reg [7:0] expected_q;
    integer i;

    pe256_scalable dut (
        .d(tb_d),
        .q(tb_q),
        .v(tb_v)
    );

    function [7:0] get_expected_q(input [255:0] d);
        integer i;
        begin : find_bit_block
            get_expected_q = 8'b0;
            for (i = 255; i >= 0; i = i - 1) begin
                if (d[i] == 1'b1) begin
                    get_expected_q = i;
                    disable find_bit_block;
                end
            end
```

```
30          end
31      endfunction
32
33      initial begin
34          $dumpfile("pe256.vcd");
35          $dumpvars(0, tb_pe256_scalable_selfchecking);
36          error_count = 0;
37
38          tb_d = 256'b0;
39          #10;
40          if (tb_v !== 1'b0) begin
41              $display("FAIL: All-zero input, ...");
42              error_count = error_count + 1;
43          end
44
45          for (i = 0; i < 256; i = i + 1) begin
46              tb_d = 0;
47              tb_d[i] = 1'b1;
48              expected_q = i;
49              #10;
50              if (tb_q !== expected_q || tb_v !== 1'b1) begin
51                  $display("FAIL: Input bit %0d | ...", i,
                        expected_q, tb_q);
52                  error_count = error_count + 1;
53              end
54          end
55
56          tb_d = 0;
57          tb_d[123] = 1'b1;
58          tb_d[200] = 1'b1;
59          tb_d[5]   = 1'b1;
60          expected_q = 200;
61          #10;
62          if (tb_q !== expected_q) begin
63              $display("FAIL: Multi-bit test | ...", expected_q,
                    tb_q);
64              error_count = error_count + 1;
65          end
66
67          for (i = 0; i < 20; i = i + 1) begin
68              tb_d = {$random, $random, $random, $random,
69                      $random, $random, $random, $random};
70              expected_q = get_expected_q(tb_d);
71              #10;
72              if (tb_q !== expected_q) begin
73                  $display("FAIL: Random test | ...", tb_d,
                        expected_q, tb_q);
74                  error_count = error_count + 1;
75              end
76          end
77
```

```
78          #20;
79          if (error_count == 0) begin
80              $display("SUCCESS: All tests passed!");
81          end else begin
82              $display("FAILURE: %0d errors found.", error_count);
83          end
84
85          $finish;
86      end
87  endmodule
```

9.2.2 Synthesized Schematic

The resulting gate-level schematic synthesized by Genus is shown in Figure 27.

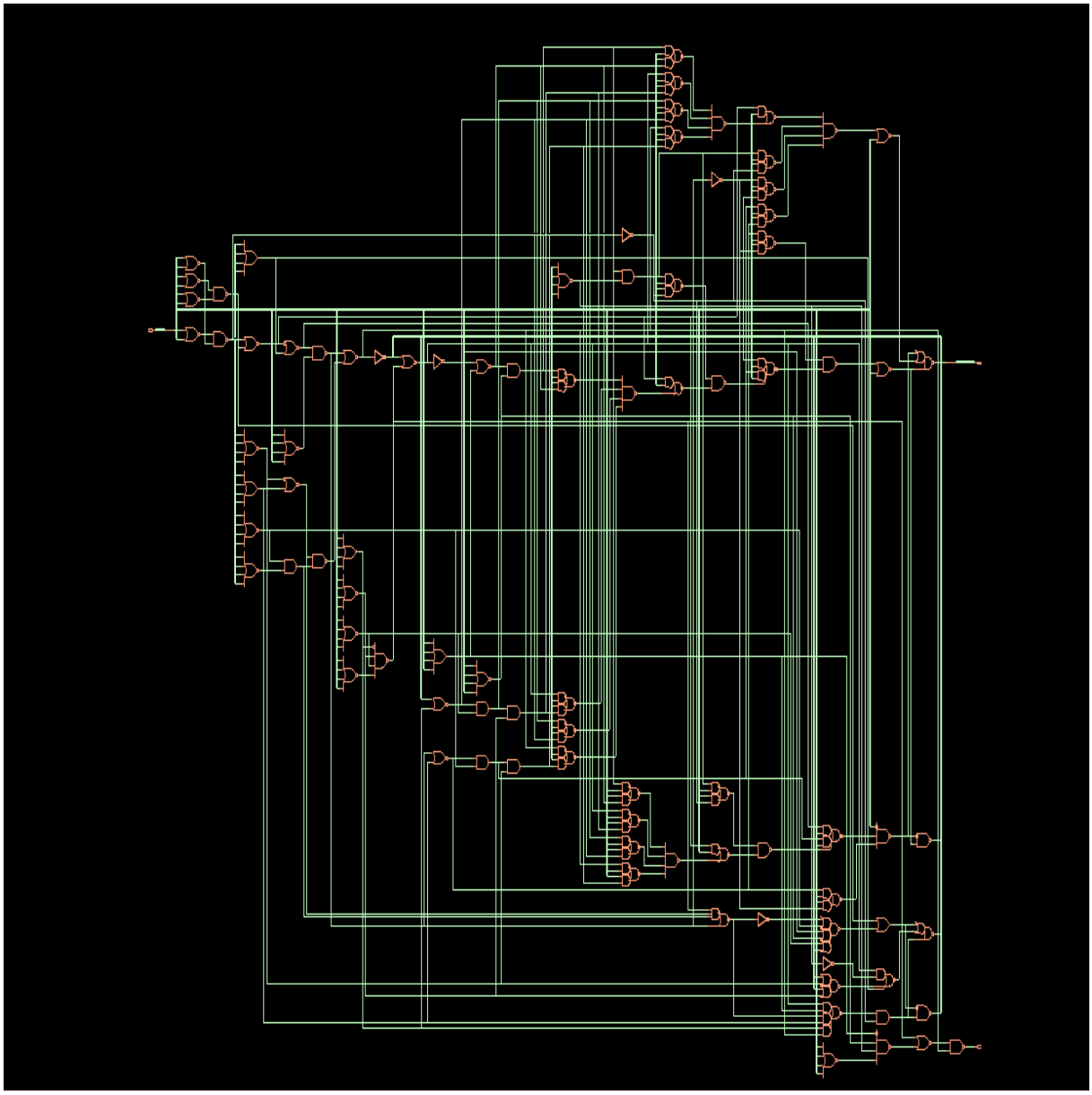
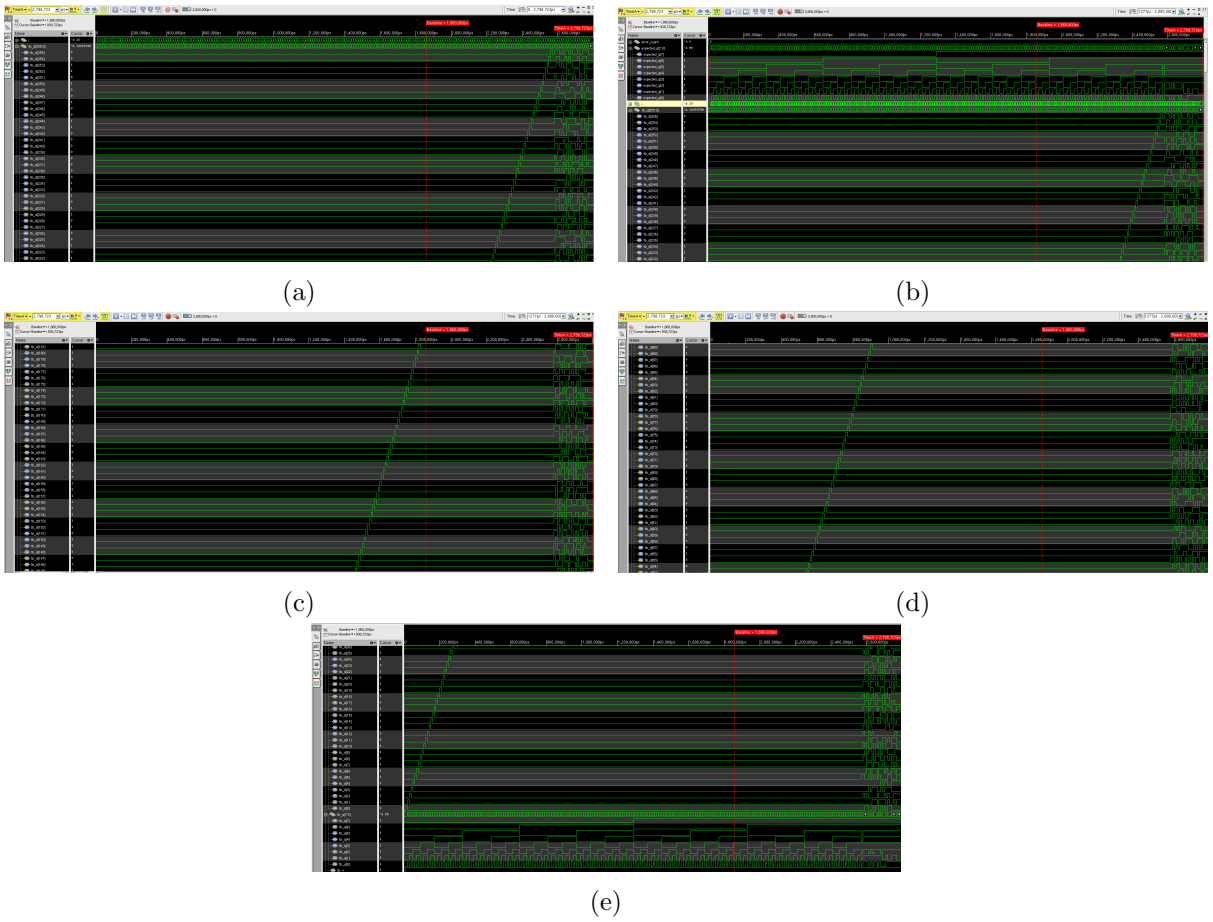


Figure 27: Synthesized schematic of look-ahead `pe64_lookahead` (90nm).

9.2.3 Simulation Waveforms

The functionality of the synthesized 90nm netlist was verified in Cadence NCLaunch.



(a) (b) (c) (d) (e)

Figure 28: Simulation waveforms verifying the synthesized 90nm look-ahead `pe64_lookahead` netlist: (a) Verification of bits 21-55, (b) Expected vs. actual output, (c) Verification of bits 45-81, (d) Verification of bits 53-89, and (e) Verification of bits 1-37.

9.3 Logic Synthesis (Genus)

The `pe64_lookahead` module was synthesized using Cadence Genus with the 90nm slow-corner library.

9.3.1 Synthesis Scripts

The synthesis process was guided by the Tcl script in Listing 25, which sets the 90nm library path. The same 1.2 ns timing constraint (Listing 26) was used. The output SDC file is shown in Listing 27.

Listing 25: Genus synthesis script (`run.tcl`) - 90nm

```
1 set_db init_lib_search_path /home/install/FOUNDRY/digital/90nm/
     dig/lib/
2
3 set_db library slow.lib
4
5 read_hdl {./priority_encoder.v}
6
7 elaborate pe64_lookahead
```

```
read_sdc ./constraint_input.sdc

set_db syn_generic_effort medium
set_db syn_map_effort medium
set_db syn_opt_effort medium

syn_generic
syn_map
syn_opt

write_hdl > priority_netlist.v
write_sdc > priority_output.sdc

report timing > priority_timing.rpt
report power > priority_power.rpt
report area  > priority_cell.rpt
report gates > priority_gates.rpt

gui_show
```

Listing 26: Synthesis design constraints (`constraints_input.sdc`) - 90nm

```
set_max_delay 1.2 -from [get_ports d[]] -to [get_ports {q[] v}]
```

Listing 27: Genus output constraints (`priority_output.sdc`) - 90nm

```
# ############################################################
#  Created by Genus(TM) Synthesis Solution 20.11-s111_1 on Mon
   Oct 06 14:16:48 IST 2025
# ############################################################

set sdc_version 2.0
...
current_design pe64_lookahead

set_max_delay 1.2 -from [list \
  [get_ports {d[63]}]  \
...
  [get_ports {d[0]}] ] -to [get_ports v]
set_clock_gating_check -setup 0.0
set_wire_load_mode "enclosed"
```

9.3.2 Synthesized Netlist

The following is the gate-level netlist (Listing 28) generated by Genus for the 90nm scalable design.

Listing 28: Synthesized Netlist (`priority_netlist.v`) - 90nm

```
// Generated by Cadence Genus(TM) Synthesis Solution 20.11-s111_1
// Generated on: Oct  6 2025 14:16:48 IST (Oct  6 2025 08:46:48
   UTC)
// Verification Directory fv/pe64_lookahead

```

```
module pe64_lookahead(d, q, v);
  input [63:0] d;
  output [5:0] q;
  output v;
...
  NAND2BXL g2789__2398(.AN (n_93), .B (n_92), .Y (q[1]));
  OAI21XL g2788__5107(.A0 (n_93), .A1 (n_89), .B0 (n_92), .Y (q
     [0]));
...
  OR4X2 g2884(.A (d[54]), .B (d[53]), .C (d[55]), .D (d[52]), .Y
       (n_161));
endmodule
```

9.3.3 Synthesis Reports

The following reports were generated by Genus for the 90nm look-ahead design.

Timing Report The 90nm look-ahead design easily **MET** the 1.2 ns timing constraint. The text-based report shows a final **WNS of 61 ps** (0.061 ns). This corresponds to a critical path delay of **1.139 ns** (1139 ps).

```
============================================================
 Generated by:           Genus(TM) Synthesis Solution 20.11-s111_1
 Generated on:           Oct 06 2025  02:16:48 pm
 Module:                 pe64_lookahead
...
============================================================

Path 1: MET (61 ps) Path Delay Check
     Startpoint: (F) d[61]
     Endpoint: (F) v
 ...
     Required Time:=     1200
     Data Path:-         1139
     Slack:=             61
...
# Timing Point      Flags   Arc    Edge  Cell        ...  Delay Arrival
# ------------------------------------------------------------------------
  d[61]             -       -      F     (arrival)   ...  0     0
  g2818/Y           -       A->Y   R     NOR4X1      ...  269   269
  g2806/Y           -       A->Y   F     CLKINVX1    ...  91    360
  g2793__6260/Y     -       A->Y   R     NOR3X1      ...  199   559
  g2790__2398/Y     -       B->Y   F     NAND2XL     ...  173   732
  g2786__6417/Y     -       B->Y   R     NOR2X1      ...  160   892
  g2785/Y           -       A->Y   F     CLKINVX1    ...  104   996
  g2782__5115/Y     -       C->Y   F     OR3XL       ...  143   1139
  v                 -       -      F     (port)      ...  0     1139
```

Power Report The total power consumption was **6.32579e-06 W**, or **6.33 μW**. This is the lowest power of all designs. The power is distributed between **Internal (38.50%)**, **Switch-

ing (38.87%)**, and **Leakage (22.63%)**.

```
Instance: /pe64_lookahead
Power Unit: W
PDB Frames: /stim#0/frame#0
 ------------------------------------------------------------------------
   Category        Leakage     Internal   Switching         Total     Row%
 ------------------------------------------------------------------------
     logic       1.43154e-06  2.43550e-06  2.45875e-06  6.32579e-06 100.00%
 ------------------------------------------------------------------------
   Subtotal      1.43154e-06  2.43550e-06  2.45875e-06  6.32579e-06 100.00%
 Percentage           22.63%       38.50%       38.87%       100.00% 100.00%
 ------------------------------------------------------------------------
```

Area Report The 90nm look-ahead design was the most area-efficient, requiring only **84 cells** and a **Total Area of 431.433** $\mu\mathbf{m}^2$. The gate-level breakdown is also provided.

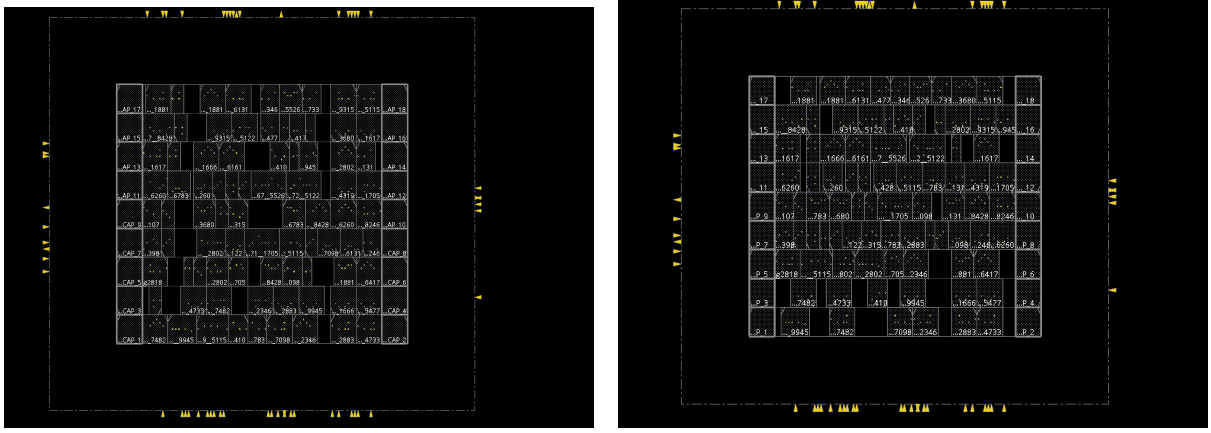
```
============================================================
 Generated by:           Genus(TM) Synthesis Solution 20.11-s111_1
 Generated on:           Oct 06 2025  02:16:48 pm
 Module:                 pe64_lookahead
...
============================================================
   Instance      Module   Cell Count  Cell Area  Net Area   Total Area   Wireload
-------------------------------------------------------------------------------
pe64_lookahead                 84      431.433     0.000      431.433 <none> (D)
...
   Gate       Instances   Area   Library
-------------------------------------
AND2X1              6   27.248     slow
AO22X1              1    7.569     slow
AOI211XL            1    5.298     slow
AOI21XL             2    9.083     slow
AOI221XL            1    7.569     slow
AOI22XL            18  108.994     slow
CLKINVX1            2    4.541     slow
INVXL               3    6.812     slow
NAND2BXL            1    4.541     slow
NAND2XL             6   18.166     slow
NAND3BXL            1    6.055     slow
NAND4XL             8   42.386     slow
NOR2BXL             3   13.624     slow
NOR2X1              1    3.784     slow
NOR2XL              6   18.166     slow
NOR3X1              1    4.541     slow
NOR4X1             14   84.773     slow
OAI21XL             3   13.624     slow
OAI222XL            3   24.978     slow
OR3XL               1    6.055     slow
OR4XL               2   13.624     slow
-------------------------------------
total              84  431.433
```

9.4 Physical Implementation (Innovus)

The 90nm synthesized netlist for the look-ahead design was then passed to Cadence Innovus for physical implementation.

9.4.1 Floorplan and Placement

Figure 29 shows the design's floorplan and cell placement for the 84 standard cells, both before and after nano-optimization.



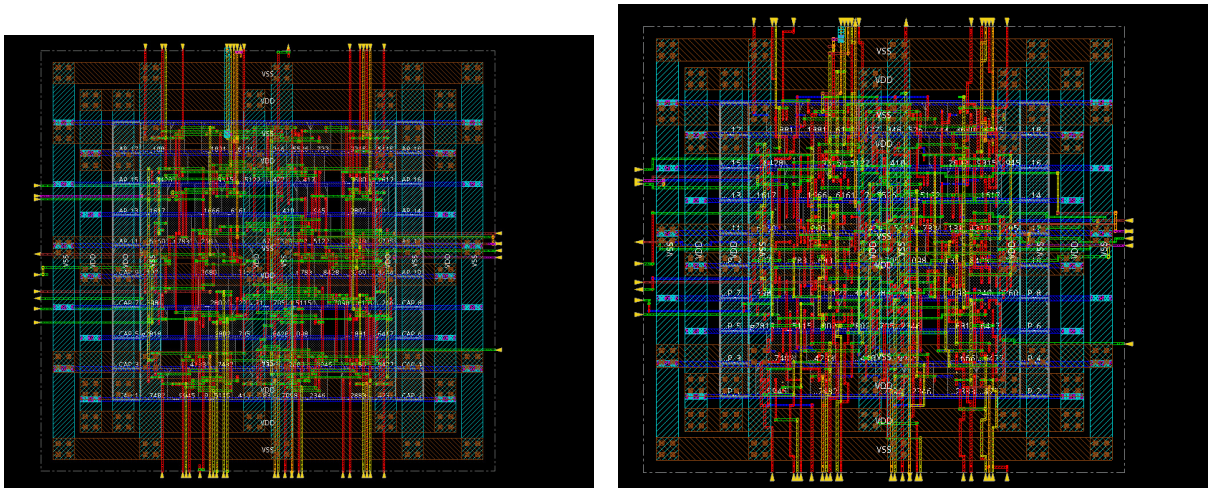
(a) Floorplan and cell placement before optimization.

(b) Floorplan and cell placement after nano-optimization.

Figure 29: Innovus floorplan and placement views for look-ahead `pe64_lookahead` (90nm).

9.4.2 Final Layout (Post-Routing)

The final routed layout is shown in Figure 30. Figure 30a shows the layout before final optimization, and Figure 30b shows the layout after nano-optimization.



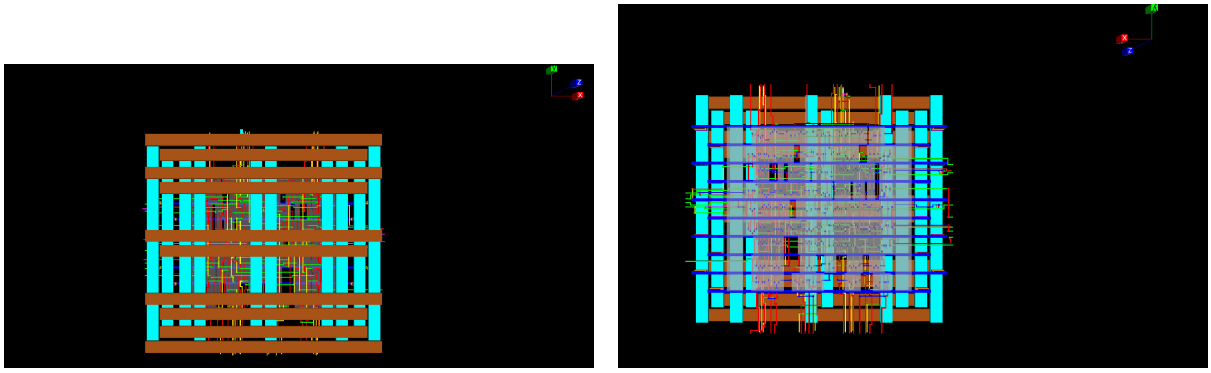
(a) Layout after initial routing, before optimization.

(b) Final layout after post-nano optimization.

Figure 30: Innovus final layout views for look-ahead `pe64_lookahead` (90nm).

9.4.3 3D Layout View

The 3D views in Figure 31 illustrate the physical stack-up of the metal layers for the 90nm look-ahead design.



(a) 3D View (Front). (b) 3D View (Back).

Figure 31: 3D layout visualization for look-ahead `pe64_lookahead` (90nm).

9.4.4 Pre-Optimization Reports (Initial Placement)

After the initial design import and cell placement, the tool generated preliminary reports.

Pre-Placement Timing Report The initial timing analysis shows the design easily **MET** the 1.2 ns constraint. The Worst Negative Slack (WNS) was **0.846 ns**, with 0 violating paths.

```
Initial Summary
----------------------------------------------------------
Setup views included: bc

+-----------------+-------+
| Setup mode      | all   |
+-----------------+-------+
| WNS (ns):       | 0.846 |
| TNS (ns):       | 0.000 |
| Violating Paths:| 0     |
| All Paths:      | 1     |
+-----------------+-------+
...
Density: 84.444%
```

Pre-Placement Area Report The initial area report shows an **Instance Count of 84** and a **Total Area of 431.433** μ**m**^{2}, matching the post-synthesis report from Genus.

```
@innovus 1> report_area
Hinst Name        Module Name     Inst Count     Total Area
----------------------------------------------------------
pe64_lookahead                    84             431.433
```

Pre-Placement Power Report The initial power estimate after placement was **7.91 mW** (0.00790703 W). For this 90nm design, the power is dominated by **Leakage Power (46.03%)** and **Internal Power (41.70%)**.

```
Total Power:                              0.00790703
 Total Internal Power:                    0.00329712   41.6985%
 Total Switching Power:                   0.00096992   12.2666%
 Total Leakage Power:                     0.00363999   46.0349%
...
Group            Internal   Switching  Leakage    Total      Percentage (%)
...
Combinational    0.003297   0.000970   0.003640   0.007907   100
...
Total instances in design: 84
```

9.4.5 Post-Route Optimization Reports

The tool next performed optDesign, which routes the design and performs optimizations.

Post-Optimization Timing Report The optDesign summary shows the design still easily meets the 1.2 ns constraint with a **WNS of 0.821 ns**. The final timeDesign summary confirms a final **Setup WNS of 0.810 ns**. The hold timing check also passed with a **Hold WNS of 0.000 ns**. This indicates a final critical path delay of **0.390 ns** (1.2 ns - 0.810 ns).

```
optDesign Final Summary
---------------------------------------------------------
Setup views included: bc
...
+-----------------+-------+---------+
| Setup mode      | all   | default |
+-----------------+-------+---------+
| WNS (ns):       | 0.821 | 0.821   |
| TNS (ns):       | 0.000 | 0.000   |
| Violating Paths:| 0     | 0       |
| All Paths:      | 1     | 1       |
+-----------------+-------+---------+
...
Density: 84.296%
*** Finished optDesign ***

timeDesign Summary
---------------------------------------------------------
Setup views included: bc
...
+-----------------+-------+---------+
| Setup mode      | all   | default |
+-----------------+-------+---------+
| WNS (ns):       | 0.810 | 0.810   |
| TNS (ns):       | 0.000 | 0.000   |
| Violating Paths:| 0     | 0       |
| All Paths:      | 1     | 1       |
+-----------------+-------+---------+
Density: 84.296%

timeDesign Summary
---------------------------------------------------------
```

```
Hold views included: wc
...
+-----------------+-------+---------+
| Hold mode       | all   | default |
+-----------------+-------+---------+
| WNS (ns):       | 0.000 | 0.000   |
| TNS (ns):       | 0.000 | 0.000   |
| Violating Paths:| 0     | 0       |
| All Paths:      | 0     | 0       |
+-----------------+-------+---------+
Density: 84.296%
```

Post-Optimization Area Report The post-optimization area report shows the **Instance Count (84)** did not change, but the **Total Area decreased slightly to 430.676** $\mu\mathbf{m}^2$. This indicates the tool met timing by swapping cells for smaller, more efficient equivalents.

```
@innovus 3> report_area
Hinst Name        Module Name     Inst Count     Total Area
-----------------------------------------------------------
pe64_lookahead                    84             430.676
```

Post-Optimization Power Report The final power report after nano-routing shows a total power of **7.94 mW** (0.00794132 W). This is a minor increase from the pre-placement estimate.

```
Total Power:                          0.00794132
 Total Internal Power:                0.00329636   41.5090%
 Total Switching Power:               0.00101902   12.8319%
 Total Leakage Power:                 0.00362594   45.6591%
...
Total instances in design: 84
```

9.4.6 Final Verification and Routing Statistics

Finally, the design was verified for manufacturability (DRC) and correctness (LVS/Connectivity).

Physical Verification (DRC & LVS) The design passed all physical checks with **0 DRC Violations** and **0 Connectivity Violations**, confirming the layout is correct and manufacturable.

```
@innovus 3> # check_drc ...
*** Starting Verify DRC (MEM: 1745.0) ***
...
VERIFY DRC ..... Sub-Area : 1 complete 0 Viols.

Verification Complete : 0 Viols.
```

```
@innovus 3> VERIFY_CONNECTIVITY use new engine.
********* Start: VERIFY CONNECTIVITY *********
Design Name: pe64_lookahead
...
Begin Summary
Found no problems or warnings.
End Summary
...
Verification Complete : 0 Viols. 0 Wrngs.
```

Final Routing Statistics The post-routing report confirms the final routing statistics. The **Total Wire Length is 1748 μm**, and the **Total Number of Vias is 739**.

```
#Start Post Route Wire Spread.
...
#Total wire length = 1748 um.
#Total half perimeter of net bounding box = 1742 um.
#Total wire length on LAYER Metal1 = 90 um.
#Total wire length on LAYER Metal2 = 723 um.
#Total wire length on LAYER Metal3 = 456 um.
#Total wire length on LAYER Metal4 = 345 um.
#Total wire length on LAYER Metal5 = 50 um.
#Total wire length on LAYER Metal7 = 73 um.
#Total wire length on LAYER Metal7 = 9 um.
#Total wire length on LAYER Metal8 = 1 um.
#Total number of vias = 739
#Up-Via Summary (total 739):
#-
# Metal1         362
# Metal2         247
# Metal3         93
...

Stream Out Information Processed for GDS version 3:
Units: 2000 DBU
Object         Count
--------------------
Instances      102
...
Nets           1114
...
Via Instances  739
...
Blockages      0
```

9.5 Comparative Analysis and Conclusion

9.5.1 Consolidated Results Table

Table 6 summarizes the synthesis (Genus) and post-layout (Innovus) results for all implemented designs. The "Critical Path Delay" is calculated by subtracting the WNS from the 1.2 ns target constraint.

Table 5: Comparative Analysis of 64-bit Priority Encoder Implementations

Architecture	Tech. Node	Analysis Stage	Cell Count	Total Area (μm^2)	Total Power	WNS (ns) @ 1.2ns	Critical Path Delay (ns)
Unscalable PE (pe64_if_else)	180nm	Genus	164	2012.472	53.83 μW	0.039	1.161
		Innovus	170	2098.958	43.86 mW	0.003	1.197
Unscalable PE (pe64_if_else)	90nm	Genus	139	595.680	14.42 μW	0.280	0.920
		Innovus	139	595.680	19.93 mW	0.910	0.290
Scalable MUX (pe64_lookahead)	180nm	Genus	191	2917.253	74.49 μW	-0.537	1.737
		Innovus	199	3649.061	69.49 mW	-0.743	1.943
Scalable PE (pe64_lookahead)	180nm	Genus	92	1373.803	21.45 μW	0.011	1.189
		Innovus	92	1387.109	10.42 mW	0.016	1.184
Scalable PE (pe64_lookahead)	90nm	Genus	84	431.433	7.91 mW	0.846	0.354
		Innovus	84	430.676	7.94 mW	0.810	0.390

10 Comparative Analysis

This project conducted a comprehensive design space exploration of a 64-bit Priority Encoder (PE), analyzing five distinct implementations across three architectural styles and two CMOS technology nodes. The architectures evaluated were:

- **Unscalable (`pe64_if_else`):** A traditional flat design using a single, long `if-else` priority chain. (Implemented on 180nm & 90nm).
- **Scalable MUX (`pe64_lookahead` from `priority_mux.v`):** A 1D-to-2D design (16×4) using explicit conditional operators (`?:`) to build a MUX tree for column selection. (Implemented on 180nm).
- **Scalable Look-ahead (`pe64_lookahead` from `priority_encoder.v`):** The optimized 1D-to-2D design (16×4) using a `casex` statement to infer priority look-ahead logic for column selection. (Implemented on 180nm & 90nm).

All designs were implemented through a full RTL-to-GDSII flow using Cadence Genus and Innovus.

10.1 Theoretical and Architectural Analysis

The significant performance differences observed in the results are a direct consequence of the underlying architectural differences in handling the critical path.
The performance of the scalable architectures is rooted in the 1D-to-2D array conversion method. This method transforms a 1D input vector D of size L into a 2D matrix with N rows and M columns, where $L = M \times N$. The logic operation is then parallelized:

1. **Row Status (dor):** First, an N-bit "data OR" (`dor`) vector is generated, where each bit $dor[i]$ is the result of a bitwise-OR of all M bits in that row. This is implemented in the Verilog `generate` block.
2. **Row Index (i):** An N-bit Priority Encoder (`pe16` in this project) finds the highest-priority active row from the `dor` vector, producing the row index i.
3. **Column Index (j):** In parallel, a selection logic (the `casex` or MUX tree) chooses the M bits of the highest-priority row. An M-bit Priority Encoder (`pe4` in this project) then finds the highest-priority column index j within that selected row.

The final L-bit priority index, k, is retrieved by combining the row and column indices. The general mathematical formula is:

$$k = (i \times M) + j \tag{1}$$

When M is a power of two, such as $M = 4$ or $M = 16$, this expensive multiplication and addition is replaced by simple, fast bitwise operations: a left-shift and a bitwise-OR.

$$k = (i \ll \log_2(M)) \mid j \tag{2}$$

For this project's 64-bit scalable designs, $L = 64$, $N = 16$, and $M = 4$. Therefore, the final logic implemented is:

$$q = (row\_index \ll 2) \mid col\_index \tag{3}$$

This parallel operation is the key to the $O(\sqrt{N})$ performance of the look-ahead architecture.

10.1.1 Unscalable Architecture: O(N) Delay

The `pe64_if_else` design represents a linear, flat architecture. For the lowest priority bit (`d[0]`) to be selected, the signal must propagate through the enable logic of all 63 higher-priority bits. This creates a critical path delay, T_{pd}, that is directly proportional to the number of inputs, N.

$$T_{pd,unscalable} \approx (N-1) \times T_{gate\_delay} \propto O(N) \tag{4}$$

This serial dependency is the fundamental bottleneck of all unscalable designs. The 180nm post-layout delay of 1.197 ns reflects this, as the tool had to add 6 extra buffer cells just to meet the 1.2 ns constraint. While the 90nm node's raw speed reduced this to 0.290 ns, the underlying $O(N)$ logic remains inefficient and does not scale.

10.1.2 Scalable MUX Architecture: Flawed Serial Dependency

The 1D-to-2D `pe64_standard` (MUX-based) design attempts to parallelize the logic. However, its RTL implementation creates an unintended serial dependency. The logic first calculates the row_index using a pe16 and then uses that row_index as the select line for the 16:1 column MUX. This creates a single, long critical path:

$$T_{pd,mux} = T_{gen\_dor} + T_{PE16\_row} + T_{MUX\_16:1\_select} + T_{PE4\_col} \tag{5}$$

This serial dependency of all major components results in the worst performance of all designs (1.943 ns), proving that this RTL coding style is highly inefficient and misguides the synthesis tool.

10.1.3 Scalable Look-ahead Architecture: O($\sqrt{N}$) Delay

The optimized `pe64_lookahead` (with `casex`) implements the true 1D-to-2D look-ahead concept. It calculates the row_index (Path A) and the column_data (Path B) in parallel, as both depend only on the initial dor (data OR) vector. The synthesis tool can optimize the `casex` statement into a fast, parallel priority logic block. The critical path is the longer of these two parallel operations, plus the final column encoding.

$$T_{pd,lookahead} = T_{gen\_dor} + \max(T_{PE16\_row\_index}, T_{casex\_select} + T_{PE4\_col\_index}) \tag{6}$$

This architecture effectively breaks the $O(N)$ dependency. By splitting N into $M \times K$, the delay becomes proportional to $O(M+K)$. This delay is mathematically minimized when $M \approx K \approx \sqrt{N}$. This $O(\sqrt{N})$ scalability is the key to this architecture's efficiency in both area and, ultimately, speed on modern nodes.

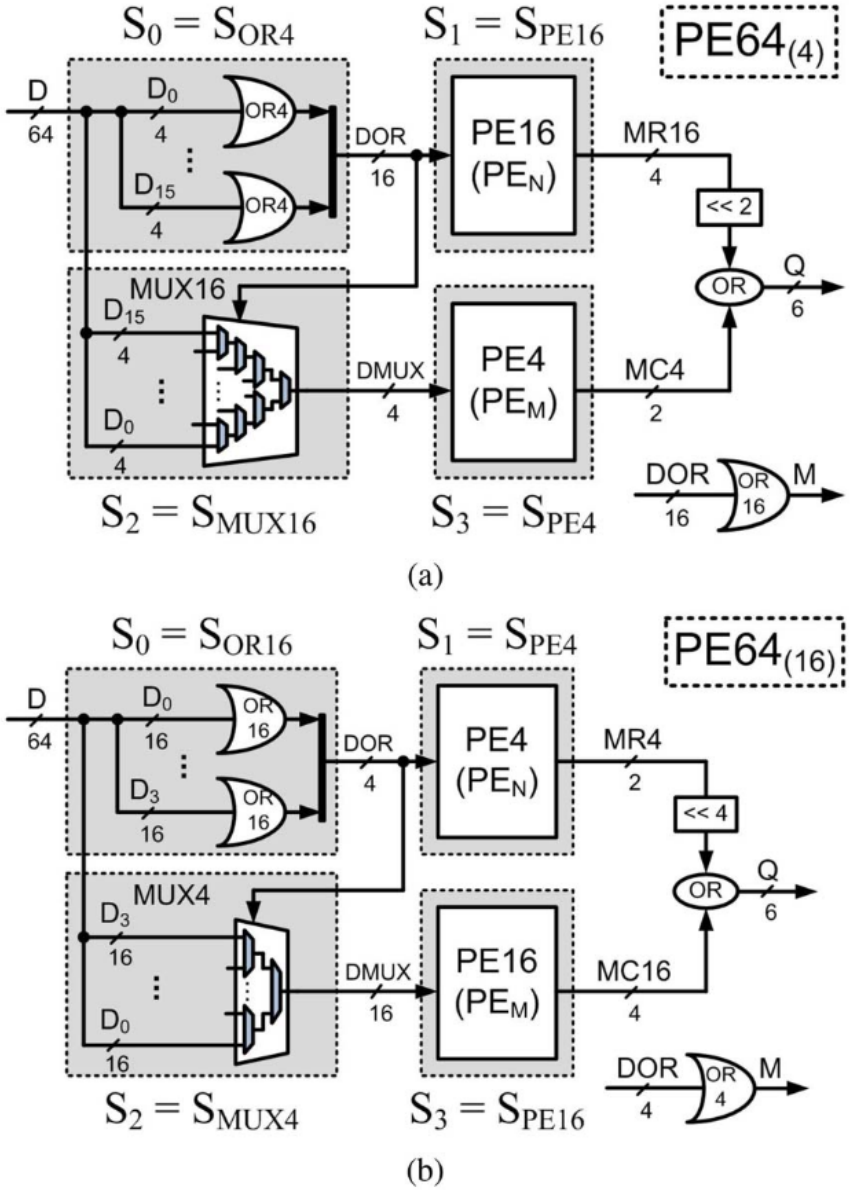


Fig. 6. The architecture of PE64 with (a) (M, N) = (4, 16) and (b) (M, N) = (16, 4).

Figure 32: Scalable 1D-to-2D PE64 architectures: (a) The implemented 16×4 ($N = 16, M = 4$) design, which uses a PE16 for row encoding and a PE4 for column encoding. (b) The alternative 4×16 ($N = 4, M = 16$) design. The same architechture is used in Scalable PE 64 lookahead in both Technologies with 180nm and 90nm

11 Result

Table 6: Comparative Analysis of 64-bit Priority Encoder Implementations

Architecture	Tech. Node	Analysis Stage	Cell Count	Total Area (μm^2)	Total Power	WNS (ns) @ 1.2ns	Critical Path Delay (ns)
Unscalable PE64 (pe64_if_else)	180nm	Genus	164	2012.472	53.83 μW	0.039	1.161
		Innovus	170	2098.958	43.86 mW	0.003	1.197
Unscalable PE64 (pe64_if_else)	90nm	Genus	139	595.680	14.42 μW	0.280	0.920
		Innovus	139	595.680	19.93 mW	0.910	0.290
Scalable MUX (pe64_lookahead)	180nm	Genus	191	2917.253	74.49 μW	-0.537	1.737
		Innovus	199	3649.061	69.49 mW	-0.743	1.943
Scalable PE64 (pe64_lookahead)	180nm	Genus	92	1373.803	21.45 μW	0.011	1.189
		Innovus	92	1387.109	10.42 mW	0.016	1.184
Scalable PE64 (pe64_lookahead)	90nm	Genus	84	431.433	6.33 μW	0.061	1.139
		Innovus	84	430.676	7.94 mW	0.810	0.390

The comprehensive analysis, summarized in Table 6, reveals a clear trade-off between architectural design, RTL coding style, and technology node.

- **Timing (Speed):** The fastest design post-layout was the **90nm Unscalable PE (0.290 ns)**, which leverages the raw speed of the 90nm process to overcome its $O(N)$ architectural flaw. The **90nm Scalable Look-ahead PE (0.390 ns)** was a close second, achieving an exceptional speed of over 2.5 GHz. The 180nm designs were significantly slower, with the Scalable Look-ahead (1.184 ns) and Unscalable (1.197 ns) designs performing similarly, while the 180nm MUX-based design was the slowest by a large margin (1.943 ns), failing its 1.2 ns constraint.
- **Area (Size):** The **Scalable Look-ahead (`casex`) architecture** was the clear winner in area efficiency. It was the smallest design on both 180nm (92 cells) and 90nm (84 cells). The Unscalable `if-else` design was significantly larger (170 cells @ 180nm, 139 cells @ 90nm), and the MUX-based RTL style produced the largest design overall (199 cells @ 180nm).
- **Power Consumption:** A major discrepancy was noted between Genus (μW) and Innovus (mW) reports, with the post-layout Innovus reports being the realistic metric for comparison. At 90nm, the **Scalable Look-ahead PE** was the most power-efficient at **7.94 mW**, which is 2.5 times lower than the unscalable 90nm design (19.93 mW). At 180nm, the Scalable Look-ahead PE was also the most power-efficient at **10.42 mW**.

Overall, while the 90nm unscalable design achieved the absolute fastest speed, the **90nm Scalable Look-ahead PE** represents the best-in-class and most optimized design. It delivers top-tier, sub-400ps performance while being 40% smaller and 2.5x more power-efficient than its unscalable 90nm counterpart, validating the 1D-to-2D conversion method as the superior VLSI design approach. Table 6 summarizes the synthesis (Genus) and post-layout (Innovus) results for all implemented designs. The "Critical Path Delay" is calculated by subtracting the WNS from the 1.2 ns target constraint.

12 Conclusion

The design, implementation, and comparative analysis of the 64-bit priority encoders yield several key findings:

1. **Architecture is Critical for Timing:** The **Scalable 1D-to-2D Look-ahead** (`casex`) architecture is demonstrably superior. The 90nm "look-ahead" design had a post-layout delay of only **0.390 ns**. In stark contrast, the 180nm MUX-based scalable design was the worst, with a post-layout delay of **1.943 ns**.

2. **RTL Coding Style Matters:** The 180nm MUX-based design performed significantly worse than the 180nm unscalable `if-else` design (1.943 ns vs 1.197 ns). This suggests that explicitly guiding the synthesis tool to build a large, serial MUX tree was highly inefficient. The 180nm "look-ahead" (`casex`) design (1.184 ns) and the unscalable (`if-else`) design (1.197 ns) had nearly identical timing, showing the 180nm tool was ableto optimize the `if-else` chain effectively.

3. **Technology Scaling:** The 90nm node provided a dramatic speedup. The unscalable `if-else` design went from 1.197 ns at 180nm to just **0.290 ns** at 90nm (a 4.1x speedup). The scalable `look-ahead` design went from 1.184 ns at 180nm to **0.390 ns** at 90nm (a 3.0x speedup).

4. **Area Efficiency:** The **Scalable Look-ahead (`casex`)** architecture is by far the most area-efficient. At 180nm, it required only **92 cells** (1387.109 μm^2), compared to 170 cells for the `if-else` design and 199 cells for the MUX design. This efficiency holds at 90nm, where it required only 84 cells.

5. **Optimized Solution:** The unscalable `if-else` design was surprisingly the fastest design overall at 90nm (0.290 ns). However, the **90nm Scalable Look-ahead PE** (using `casex`) is the best *balanced* design. It achieves a still-exceptional 0.390 ns (over 2.5 GHz) while being **40% smaller** (84 cells vs 139) and **2.5x more power-efficient** (7.94 mW vs 19.93 mW) than the 90nm unscalable design.

6. **Power Profile Discrepancy:** A significant discrepancy was observed between Genus (pre-synthesis) and Innovus (post-layout) power reports. Genus reported power in the μW range (e.g., 21.45 μW), while Innovus reported power in the mW range (e.g., 10.42 mW). This suggests the Genus reports were likely estimating static or zero-activity power, whereas the Innovus reports provide a more realistic dynamic power analysis based on layout parasitics and activity.

In conclusion, this project successfully identifies the optimal design choice. The traditional `if-else` structure is area-intensive but can be fast on modern nodes. The MUX-based RTL coding style proved to be the worst for both area and timing. The **Scalable Look-ahead (`casex`) architecture** is the superior solution, offering the best-in-class area efficiency while delivering top-tier performance, especially when implemented on the 90nm node.

References

[1] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "A Scalable High-Performance Priority Encoder Using 1D-Array to 2D-Array Conversion," *IEEE Transactions on Circuits and Systems—II: Express Briefs*, vol. 64, no. 9, pp. 1102–1106, Sep. 2017.

[2] N. H. E. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Boston, MA, USA: Addison-Wesley, 2011.

[3] C.-H. Huang, J.-S. Wang, and Y.-C. Huang, "Design of high-performance CMOS priority encoders and incrementer/decrementers using multilevel lookahead and multilevel folding techniques," *IEEE J. Solid-State Circuits*, vol. 37, no. 1, pp. 63–76, Jan. 2002.

[4] S.-W. Huang and Y.-J. Chang, "A full parallel priority encoder design used in comparator," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.*, Seattle, WA, USA, 2010, pp. 877–880.

[5] C. Kun, S. Quan, and A. Mason, "A power-optimized 64-bit priority encoder utilizing parallel priority look-ahead," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2. Vancouver, BC, Canada, 2004, pp. II-753–II-756.

[6] M. Faezipour and M. Nourani, "Wire-speed TCAM-based architectures for multimatch packet classification," *IEEE Trans. Comput.*, vol. 58, no. 1, pp. 5–17, Jan. 2009.

[7] S. Abdel-Hafeez and S. Harb, "A VLSI high-performance priority encoder using standard CMOS library," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 8, pp. 597–601, Aug. 2006.

[8] D. Balobas and N. Konofaos, "Low-power, high-performance 64-bit CMOS priority encoder using static-dynamic parallel architecture," in *Proc. IEEE Int. Conf. Modern Circuits Syst. Technol. (MOCAST)*, Thessaloniki, Greece, 2016, pp. 1–4.

[9] D.-H. Le, K. Inoue, M. Sowa, and C.-K. Pham, "An FPGA-based information detection hardware system employing multi-match content addressable memory," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E95.A, no. 10, pp. 1708–1717, Oct. 2012.

[10] S. K. Maurya and L. T. Clark, "A dynamic longest prefix matching content addressable memory for IP routing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 6, pp. 963–972, Jun. 2011.

[11] X.-T. Nguyen, H.-T. Nguyen, and C.-K. Pham, "An FPGA approach for high-performance multi-match priority encoder," *IEICE Electron. Exp.*, vol. 13, no. 13, pp. 1–9, Jun. 2016.

[12] Sohan Maity, "Scalable High-Performance Priority Encoder Using 1D-Array to 2D-Array Conversion," *GitHub Repository*, 2025. [Online]. Available: `https://github.com/sohan2311/Scalable-High-Performance-Priority-Encoder-Using-1D-Array-to-2D-Array-Conversion`