

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is light green. They are positioned diagonally, with the blue one partially covering the green one.

RLE Chip Design

Jack Decker, Colby Whitehouse, Gerardo
Muniz, Mahmoud Hafez

Description of the Project

- **Goal:** implement a 32-bit hardware Run-Length Encoder (RLE) that computes (1) run_count = number of runs of consecutive 0s/1s, and (2) max_run = longest contiguous run length in the 32-bit input word.
- **8-bit interface:** to meet the ≤ 23 pin constraint while still supporting full 32-bit input/output, the top-level streams data over an 8-bit bus (4 cycles in, 4 cycles out).
- **Top-level I/O (23 pins):** inputs {clk, rst_n, start, in_valid, in_byte[7:0]} and outputs {out_valid, out_byte[7:0], ready, busy}.
- **Architecture:** two cooperating blocks—an external I/O wrapper (rle_project) that handles byte assembly/serialization, and an internal compute FSM that performs the RLE scan.
- https://zfgames.org/rle_engine.html

INPUT INTERFACE

8FFF2314

100011111111111110010001100010100

BYTE 0
0x14

BYTE 1
0x23

BYTE 2
0xFF

BYTE 3
0x8F

OUTPUT INTERFACE

BYTE 0
0x0C

BYTE 1
0x00

BYTE 2
0x0C

BYTE 3
0x00

out_valid

32-BIT OUTPUT

0x000C000C

MAX RUN LENGTH

12

RUN COUNT

12

Description of Testbench

- **Clock + reset:** a free-running clock (forever #DELAY_NS clk=~clk) and a reset_dut() task that holds rst_n=0 for several cycles, then releases and starts tests from a known state.
- **Input matches I/O protocol:** send_word() transmits the 32-bit test vector as 4 bytes.
- **Self-checking output:** receive_output() waits for out_valid and reconstructs the 32-bit output from 4 output bytes LSB-first, matching the design's 4-cycle output serialization.
- **Reference model:** compute_expected_rle() is a software RLE model that scans the 32-bit input and computes expected {max_run, run_count}, run_test() extracts hw_max_run and hw_run_count from the received 32-bit output and prints PASS/FAIL details.

```
//=====
// Task: Software reference RLE model
//=====
task compute_expected_rle(
    input  [31:0] data,
    output [5:0] exp_max_run,
    output [5:0] exp_run_count
);
    integer i;
    logic curr_bit;
    integer curr_run_len;
    integer max_run_local;
    integer run_count_local;
    begin
```

```
204 // check results
205 if (hw_max_run == exp_max_run && hw_run_count ==
exp_run_count) begin
206     $display("*** PASS ***");
207 end else begin
208     $display("*** FAIL ***");
209     $display("MISMATCH!");
```

Description of Testbench Continued

- **Assembly Verification:** test_input_assembly() intentionally looks under the DUT once the FSM reaches S_LOAD to confirm the byte-to-word assembly is correct before relying on the rest of the pipeline.
- **Directed test set for coverage:** all-zeros, all-ones, alternating bits for max transitions, mixed long/short runs, repeated-byte patterns (A5/F0), a two-run split (16/16), a random word (DEADBEEF), and an edge-case with a single 1. These collectively stress: long runs, many short runs, frequent run closures, and boundary conditions in the final-run close.

```
// wait until S_LOAD state
wait ( uut.state == uut.S_LOAD );
@(posedge clk);

// Check assembled data
$display("Assembled input_buffer: 0x%08h", uut.input_buffer);
```

```
// Test 2: All zeros (one run of 32)
run_test(32'h00000000, "All zeros");

// Test 3: All ones (one run of 32)
run_test(32'hFFFFFFFF, "All ones");

// Test 4: Alternating bits (32 runs of length 1)
run_test(32'h55555555, "Alternating 01010101...");

// Test 5: Pattern with longer runs
run_test(32'b00001111_11111100_00111100_00001111, '');
```

Code Coverage

- Code coverage: the DUT shows ~96% overall, ~96% code, 100% block, 100% expression, ~91% toggle, and 100% FSM coverage
- FSM 100%: our directed test suite was built to force the machine through every internal state and key transitions by using patterns with long runs (all-0/all-1), maximum transitions (alternating), mixed runs, and edge cases
- Toggle coverage isn't 100%: some bits/registers don't naturally toggle in every run, so toggle tends to be the last metric to reach 100% even when functionality is correct.

Overall	96.18%
Code	95.91%
Block	100%
Statement	n/a
Expression	100%
Toggle	91.28%
FSM	100%
Functional	n/a
Assertion	n/a
CoverGroup	n/a
FaultNode	n/a

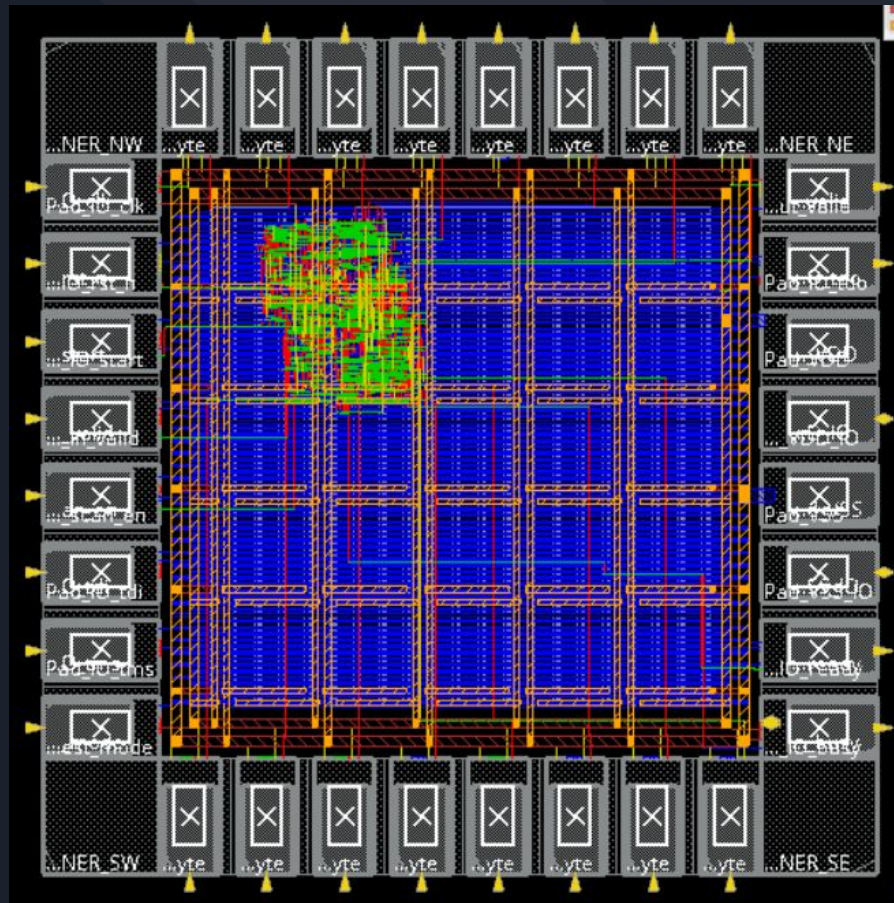
EX	UNIT	Name	Overall Average Grade
		(no filter)	(no filter)
		Verification Metrics	83.45%
		Types	92.27%
		Instances	74.64%
		rlc_engine_tb	74.64%
		uut	96.18%
		S_INPUT_dp	100%
		S_LOAD_dp	100%
		S_LOAD_FIRST_dp	85.96%
		S_CHECK_DONE_dp	100%
		S_TEST_BIT_dp	100%
		S_INC_RUN_dp	100%
		S_CLOSE_RUN_dp	92.11%
		S_SHIFT_dp	100%
		S_CLOSE_LAST_dp	86.49%
		state	100%

Summary of the Flow

Summary of flow:

```
-----
| Snapshot | WNS (ns) | TNS (ns) | FEPS | WNS(R) (ns) | TNS(R) (ns) | FEPS(R) |
| DRV(T) (ns) | DRV(C) (pF) | Power(L) (mW) | Density (%) | Insts | Area (um^2) |
| DRC | Wall (s) |
|-----+-----+-----+-----+-----+-----+-----|
| syn_generic | 5.128 | 0 | 0 | 0.00 | 0.00 | 773 | 237145 | |
| | 0 | 0 | 0.00 | 0.00 | 773 | 237145 |
| | 0:01:43 | 4.237 | 0 | 0 | 0.00 | 0.00 | 463 | 232885 |
| syn_map | -149 | 0 | 0 | 0.00 | 0.00 | 463 | 232885 |
| | 0:01:32 | 1.634 | 0 | 0 | 0.00 | 0.00 | 478 | 233223 |
| syn_opt | 0 | 0 | 0.00 | 0.00 | 478 | 233223 |
| | 0:01:43 | 8.40 | 446 | 12423 |
| floorplan | 0 | 0 | 0.00 | 8.42 | 450 | 12453 |
| | 0:02:25 | 1.229 | 0 | 0 | 5.481 | 0 | 0 |
| prects | 0 | 0 | 0.00 | 8.42 | 450 | 12453 |
| | 0:03:19 | 1.256 | 0 | 0 | 5.495 | 0 | 0 |
| cts | 34 | 0 | 0.00 | 8.41 | 451 | 12434 |
| | 0 | 0:03:33 | 1.256 | 0 | 0 | 5.495 | 0 | 0 |
| postcts | 34 | 0 | 0.00 | 8.41 | 451 | 12434 |
| | 0:02:37 | 1.273 | 0 | 0 | 5.212 | 0 | 0 |
| route | 78 | 0 | 0.00 | 8.41 | 451 | 12434 |
| | 0 | 0:02:31 | 1.273 | 0 | 0 | 5.211 | 0 | 0 |
| postroute | 34 | 0 | 0.00 | 8.41 | 451 | 12440 |
| | 0 | 0:03:58 | 1.265 | 0 | 0 | 5.188 | 0 | 0 |
| signoff | 79 | 0 | 0.00 | 5.43 | 451 | 12440 |
| | 0:03:23 | 1.042 | 0 | 0 | 4.921 | 0 | 0 |
| opt_signoff | 34 | 0 | 0.00 | 8.38 | 448 | 12396 |
| | 0 | 0:08:39 |
|-----+-----+-----+-----+-----+-----+-----|
|-----+-----+-----+-----+-----+-----+-----|
|-----+-----+-----+-----+-----+-----+-----|
```

Innovus Implementation Flow



Total Instance Power

Static Power Analysis: Key Takeaways

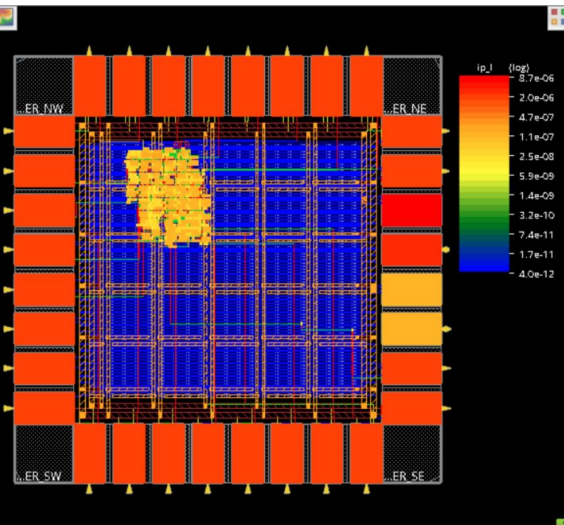
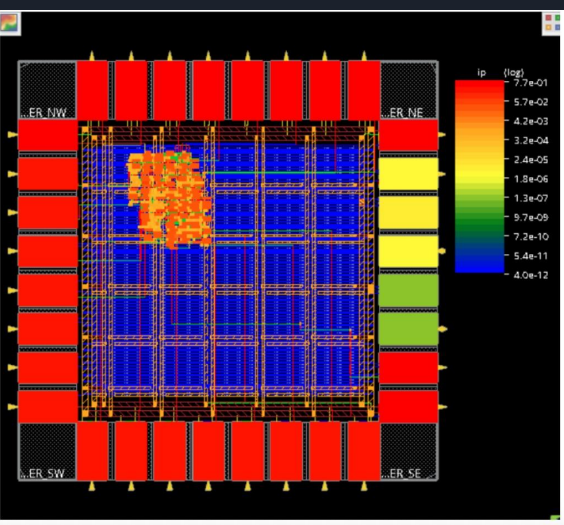
Overall Power Distribution:

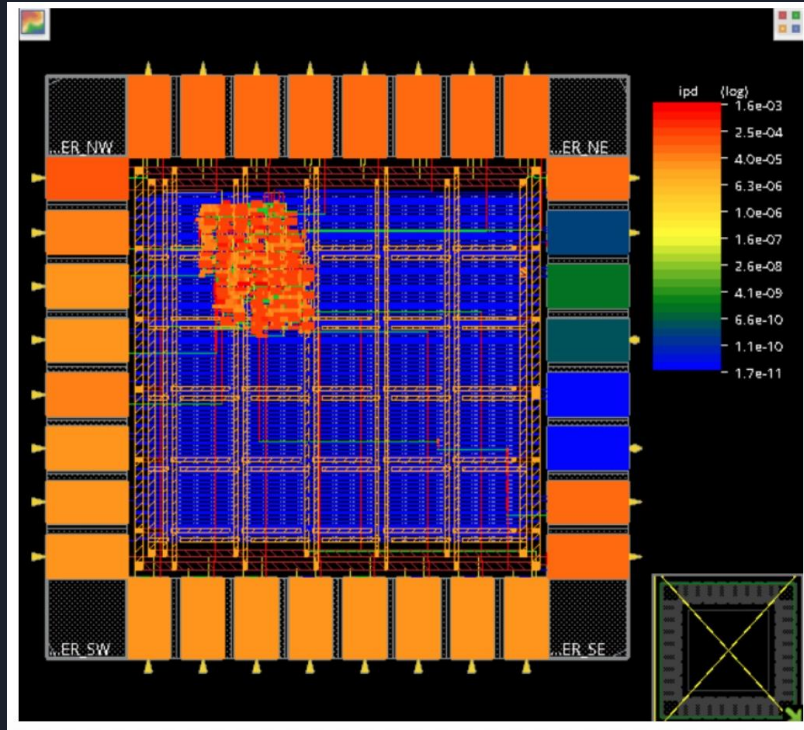
- Pads dominate total power consumption, far exceeding the power required by internal logic.
- I/O pads and clock signals consume the most power due to driving large external capacitances. (Switches)
- Core logic power is significant but localized, mainly concentrated in the top left region of the chip
- Large portions of the chip consume near-zero power, indicating efficient area utilization.

Major Finding:

- The design is I/O power dominated, not logic power dominated.

Instance Leakage Power





Power Types & Density Insights

Switching vs. Leakage Power:

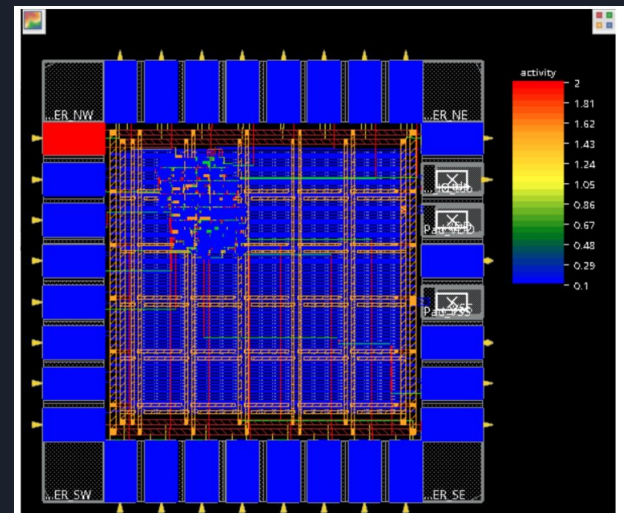
- Switching power is almost entirely attributed to the pads, primarily due to large external loads
- Internal logic has minimal switching power, indicating limited internal activity.
- Leakage power is low
- Leakage is highest in pads and core logic, but remains negligible overall.

Power Density Observations:

- Highest power density occurs in internal logic, not pads
- This is due to the small area and active transistors in the core
- Confirms expected behavior for a logic-centric region.

Major Finding:

- While pads consume more total power, core logic is the thermal hotspot due to high density.



Instance Activity

Performance, Activity & Capacitance

Timing & Slack:

- Near zero slack across the chip
- Very fast signal propagation
- Strong timing performance
- Potential for scaling to larger designs

Switching Activity:

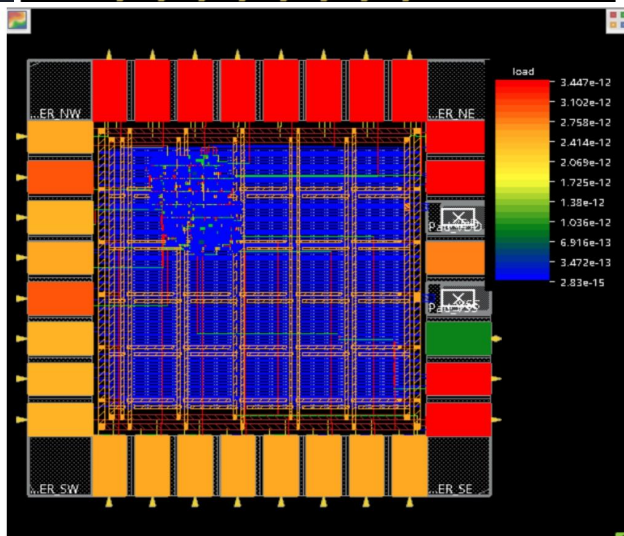
- Clock shows the highest activity, as expected.
- Most logic switches < 0.2 times per cycle, confirming efficient operation.

Loading Capacitance:

- Output pads have the highest load capacitance
- Internal logic has almost no capacitive loading
- Input pads show moderate loading driven by outputs

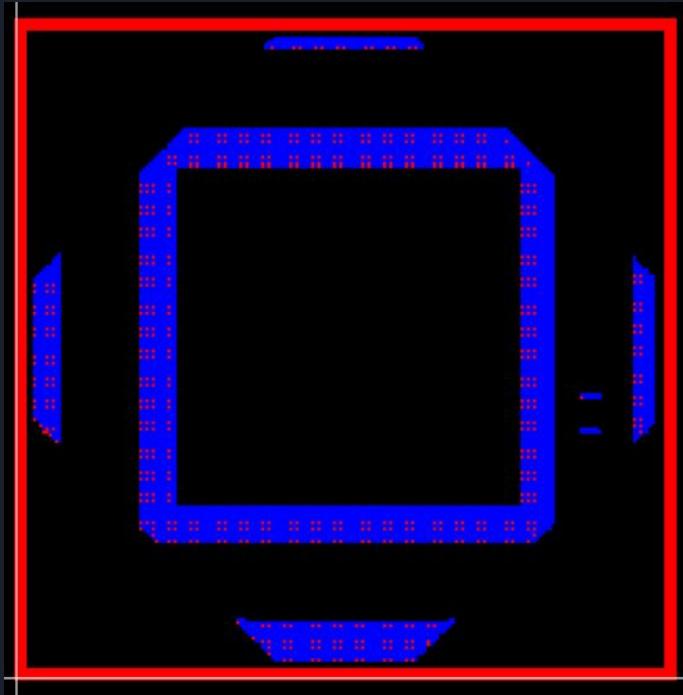
Major Finding:

- Performance is clock dominated, with low internal switching and minimal delay.

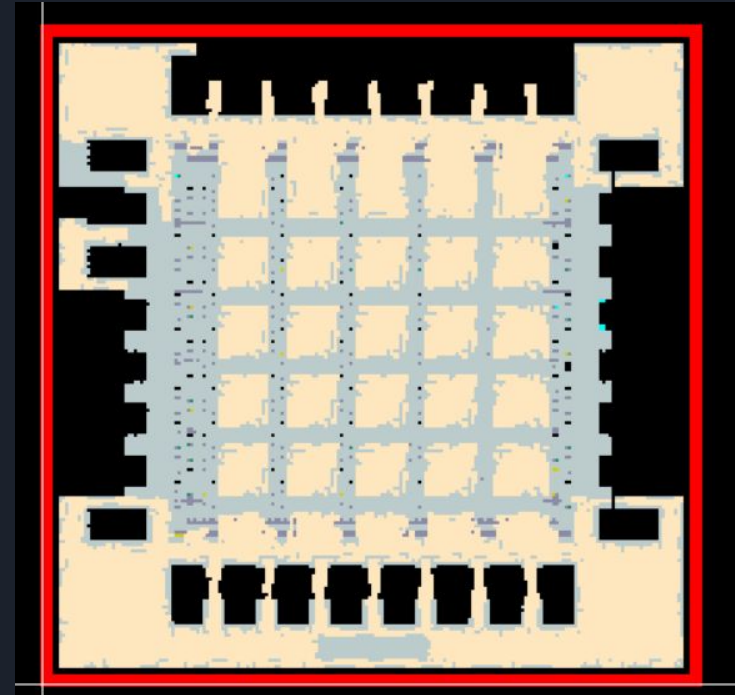


Instance Loading Capacitance

Dummy OD and Dummy Metal Layers

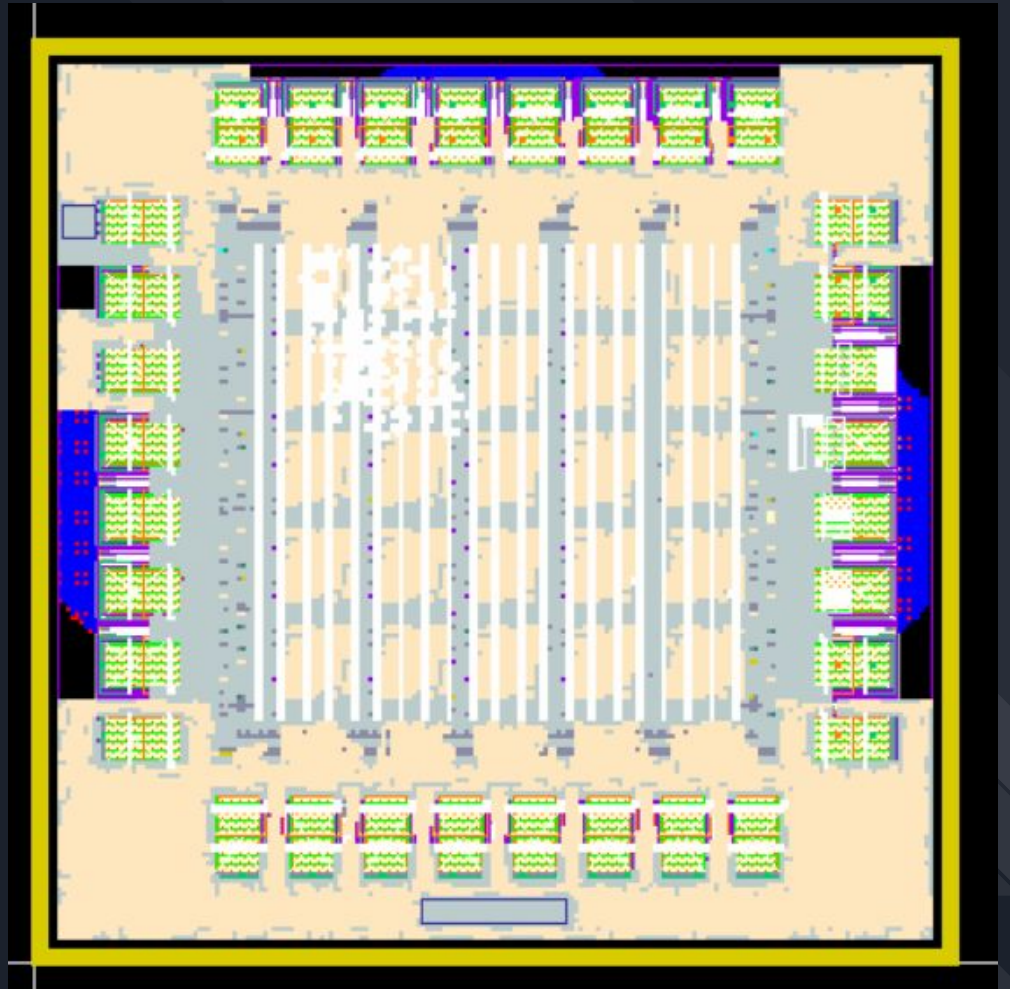


Dummy Oxide Diffusion

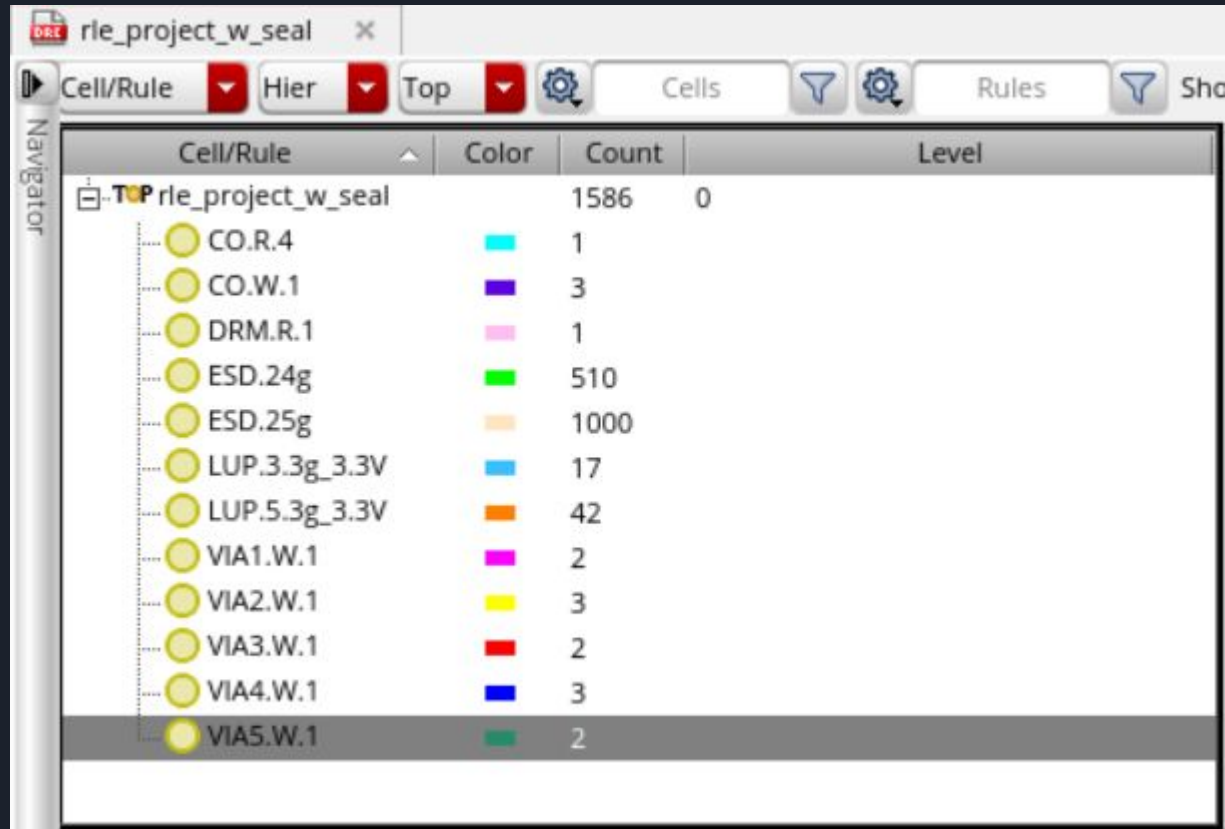


Dummy Metal

Final Chip Design



Passing DRCs



Cell/Rule Hier Top Cells Rules Show

Cell/Rule	Color	Count	Level
TOP rle_project_w_seal		1586	0
CO.R.4		1	
CO.W.1		3	
DRM.R.1		1	
ESD.24g		510	
ESD.25g		1000	
LUP.3.3g_3.3V		17	
LUP.5.3g_3.3V		42	
VIA1.W.1		2	
VIA2.W.1		3	
VIA3.W.1		2	
VIA4.W.1		3	
VIA5.W.1		2	