Silicon Sprint
Competition
Round 2

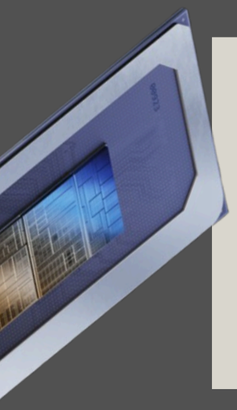
Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

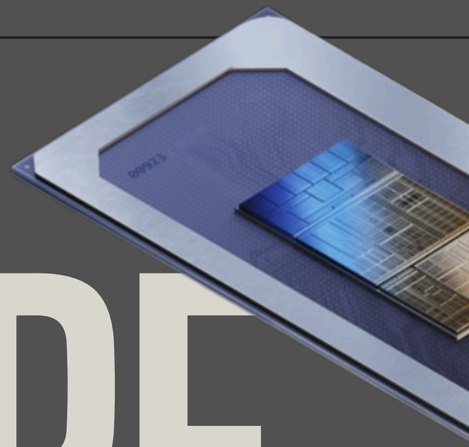
LOGISYM team

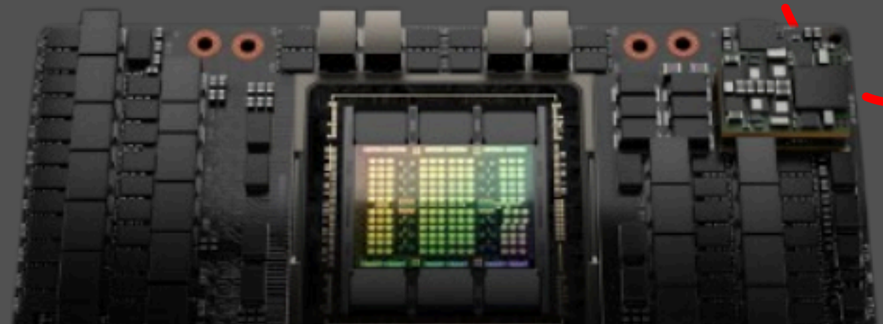
Problem D1

NPU MICROCORE

2025







4x4 Systolic Array
RTL Exploration

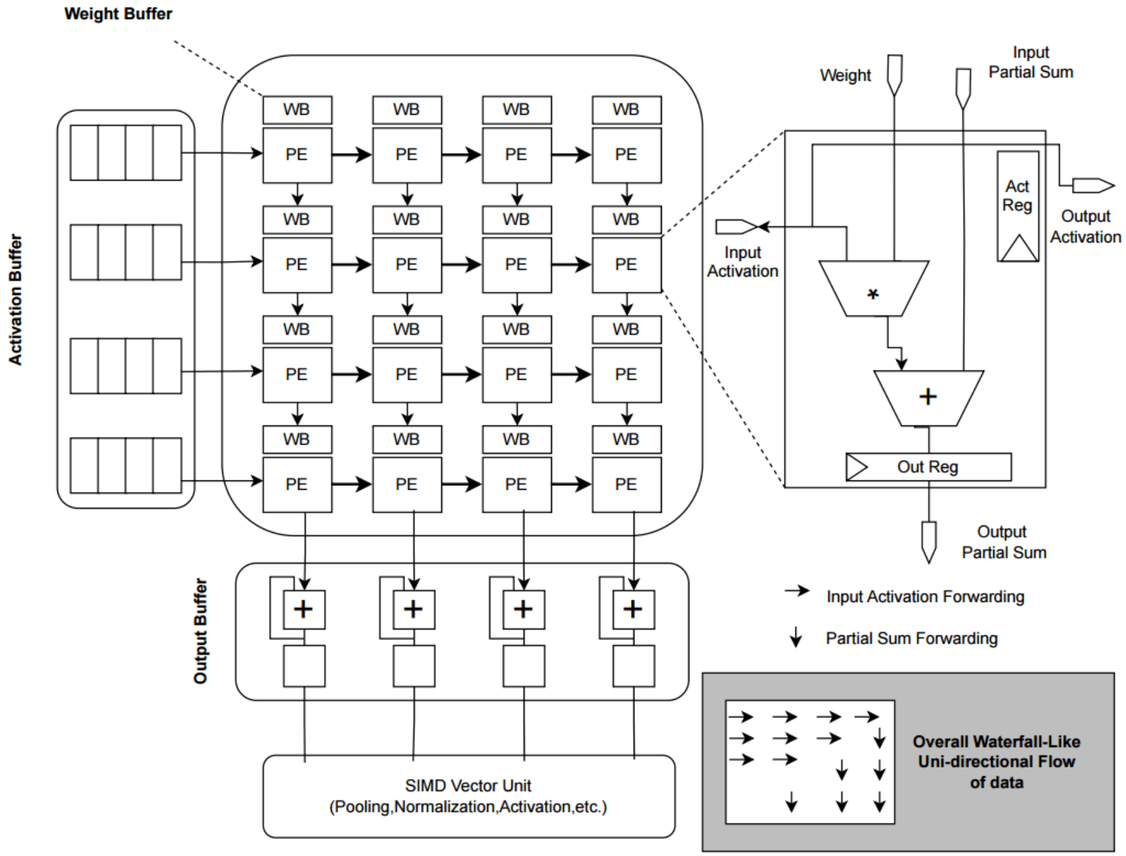
Silicon Sprint
Competition
Round 2

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

Systolic Arrays

It is a grid of interconnected processing units, called Processing Elements (PEs). In this case, it's a 4x4 grid, meaning 16 PEs. Each PE receives data from its neighbors, performs a small computation, and then passes the data to its other neighbors in the next clock cycle.



Silicon Sprint
Competition
Round 2

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

Project Details

Key Design Constraints:

- Data Types:
  - Inputs (A, B): 8-bit signed integers.
  - Accumulator (Internal): 20-bit signed (ACCW_ = 20).
  - Output (C): 16-bit signed, saturated (CW = 16).
- Dataflow:
  - Matrix A elements stream in from the left and move right.
  - Matrix B elements stream in from the top and move down.
- Pipelining: Each PE must register its inputs, passing them to the next PE on the following clock cycle.

Silicon Sprint
Competition
Round 2

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

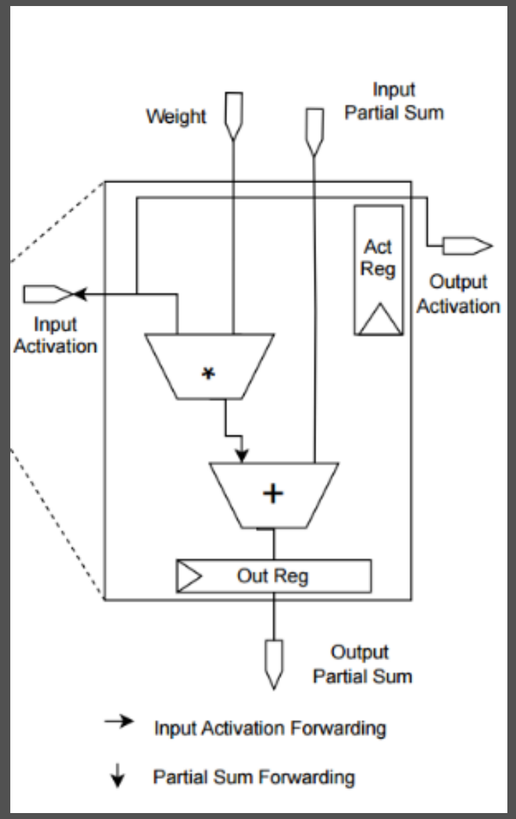
PE Design

Multiply-Accumulate (MAC):

- A combinatorial multiplier calculates the product: prod = a_in * b_in;
- The accumulator register updates if data is valid: if (do_acc) c_acc <= c_acc + prod;

Control Signals:

- clk, rst: Standard clock and reset.
- clr: An extra signal to reset the 20-bit c_acc to zero, preparing for a new matrix calculation.
- a_v_in, b_v_in: Valid bits that are ANDed (do_acc) to ensure accumulation only happens when both inputs are valid.



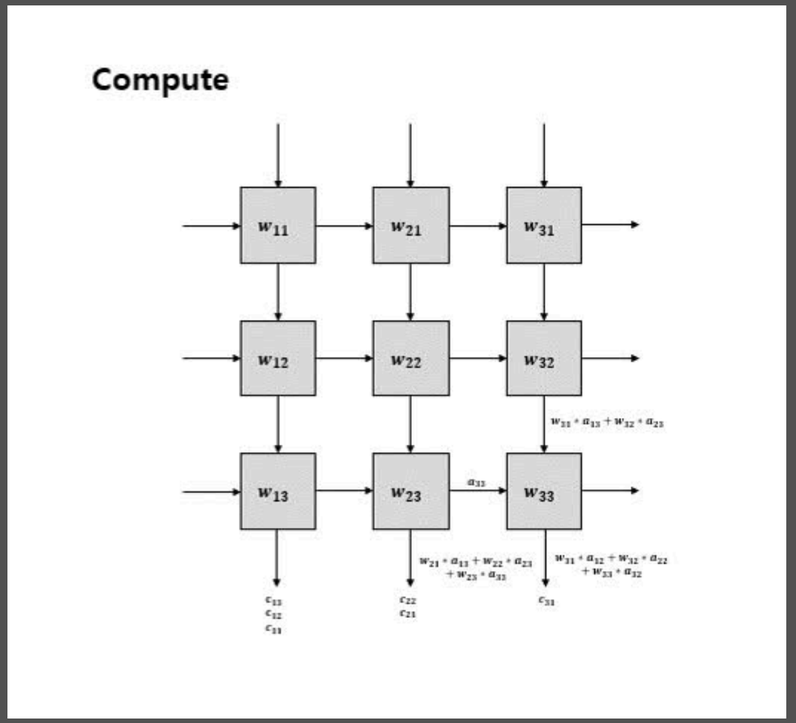
Silicon Sprint
Competition
Round 2

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

PE Dataflow

- 1. UNFLATTEN: genvar i loops to convert the 1D a_left_flat vector into a 2D a_left[i] array
- 2. BOUNDARY: Injects this data into the edge of the PE mesh.
- assign a_bus[i][0] = a_left[i]; (Left Edge)
- assign b_bus[0][i] = b_top[i]; (Top Edge)
- 3. ROW/COL Mesh: A nested genvar r, c loop instantiates the 16 pe_cell modules.
- A's Output: a_out(a_bus[r][c+1]) feeds the PE to the right.
- B's Output: b_out(b_bus[r+1][c]) feeds the PE to the bottom.



Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

PE Output Handling

1. Saturation (Preventing Overflow):
    - Our internal accumulators (c_acc) are 20-bits wide to prevent overflow during summation.
    - The output must be 16-bits.
    - We use a sat16 function to clip the 20-bit result to the 16-bit signed range ('sh7FFF to -'sh8000) before outputting.
2. Signaling (Knowing When We're Done):
    - A systolic array has a fixed, deterministic latency.
    - LATENCY = K + (N-1) + (N-1) = 4 + 3 + 3 = 10 cycles.
    - A state machine detects the start_beat (first valid data) and asserts the done signal exactly 10 cycles later, telling the testbench the data is ready.

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

RTL Issues

- The Problem: Our first simulations failed. The array was calculating the wrong values.
- The Cause: We were turning the valid signals off too early (after 4 cycles). We didn't realize the last data elements (A[i][3] and B[3][j]) were still propagating through the pipeline
- The Fix: The valid signals must stay HIGH for the entire latency (K+N-1+N-1 = 10 cycles). This ensures all data is accumulated, even as it reaches the final PE.

```
========================================
Systolic Array Matrix Multiplication
========================================

Matrix A (4x4):
  1   2   3   4
  5   6   7   8
  2   4   6   8
  1   3   5   7

Matrix B (4x4):
  1   0   0   1
  0   1   0   1
  0   0   1   1
  1   1   1   2

Expected C = A × B:
  5   6   7  14
 13  14  15  34
 10  12  14  28
  8  10  12  23

========================================
Computed C matrix:
========================================
  5   6   7  14
 13  14  15  34
 10  12  14  12
  8  10   5   4
```

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

RTL Design Considerations

- Parameterized Design (parameter N=4):

The same code can generate a 2x2, 8x8, or 16x16 array just by changing parameters. This allows for rapid design exploration.

- Flattened I/O (a_left_flat [N*DW-1:0]):

Using packed 1D vectors for module I/O is a Verilog-2001 standard.
This ensures maximum compatibility with synthesis tools (like Yosys) that have poor support for unpacked arrays in ports.

- generate Blocks for Mesh:

Using generate for... creates a regular, predictable structure that synthesis tools can optimize well.

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

Verification Strategy

1. Golden Model Calculation:
   - The testbench first initializes A and B matrices.
   - It computes the expected_C matrix using simple, behavioral for loops.
2. Stimulus Generation:
   - The testbench asserts rst and clr to put the array in a known-good state.
   - It then streams the skewed A and B matrices into the DUT, one "wave" per clock cycle, for K+N cycles.
3. Results Verification:
   - The testbench waits for the done signal from the DUT and when high reads the 16-bit saturated values from C_flat.
   - It compares every C[i][j] from the DUT against the expected_C[i][j].
   - If all 16 values match, it prints "PASS". Otherwise, it prints "FAIL" and reports the mismatch.

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

Verification Testing

A for loop (from t=0 to K+N-1) feeds the correct data at the correct time.

- t=0: A[0][0] and B[0][0] are fed.
- t=1: A[0][1], A[1][0] and B[0][1], B[1][0] are fed.
- t=2: A[0][2], A[1][1], A[2][0] and B[0][2], B[1][1], B[2][0] are fed...
- This "wavefront" correctly mimics how the array would be fed data.

The testbench halts and does a wait(done). When done goes high, it compares its expected_C with the C_flat output from the DUT.

Silicon Sprint
Competition
Round 2

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

Results

- Successfully designed and verified a 4x4, pipelined, output-stationary systolic array.
- Architecture: The design is modular (pe_cell) and scalable (systolic_array_4x4 parameterized with N=4).
- Verification: The implementation was fully verified against a software-based golden model.

```
Matrix A (4x4):
  1   2   3   4
  5   6   7   8
  2   4   6   8
  1   3   5   7

Matrix B (4x4):
  1   0   0   1
  0   1   0   1
  0   0   1   1
  1   1   1   2

Expected C = A × B:
  5   6   7  14
 13  14  15  34
 10  12  14  28
  8  10  12  23
```

```
========================================
Computed C matrix:
========================================
  5   6   7  14
 13  14  15  34
 10  12  14  28
  8  10  12  23

========================================
Verification:
========================================
PASS - All results match expected values!

========================================
```

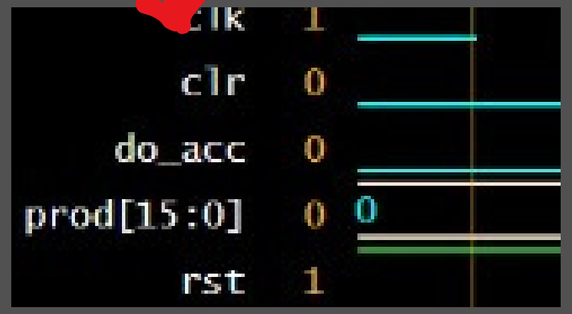
```
========================================
```

Silicon Sprint
Competition
Round 2

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

Results



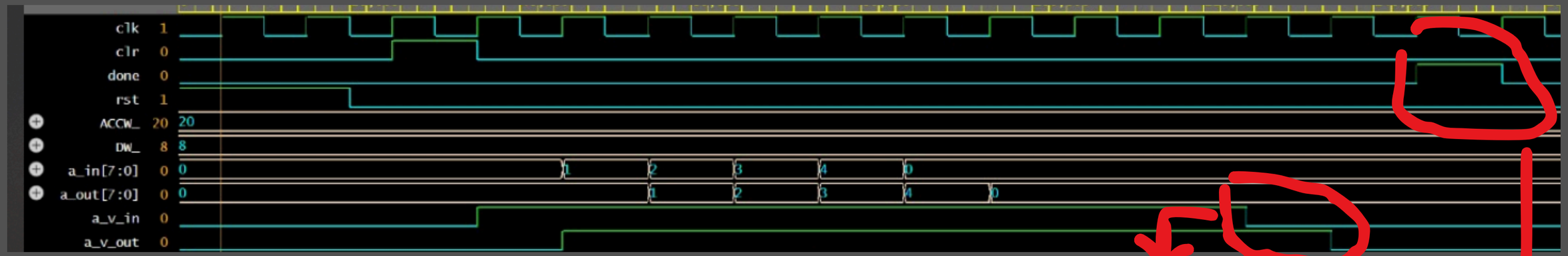
1. Initialization (at ~20ns): The clr signal pulses high, correctly resetting the c_acc (accumulator) to 0 before the calculation begins.
2. The Data-Valid Fix (at ~40ns): The a_v_in and b_v_in signals go high and stay high for the entire streaming phase. This confirms our bug fix is implemented and the PEs are enabled for accumulation.

Silicon Sprint
Competition
Round 2

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

Results



1. Pipelining (, a_v_in vs. a_v_out): There is a clear 1-clock cycle (10ns) delay between a_v_in and a_v_out. This proves the PE is correctly registering its outputs, which is the core of a pipelined design.
2. Completion (at ~140ns): After the fixed latency, the done signal asserts high, signaling to the testbench that the calculation is complete and the final result is ready.

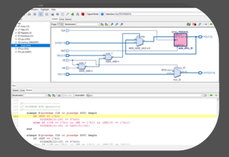
Silicon Sprint
Competition
Round 2

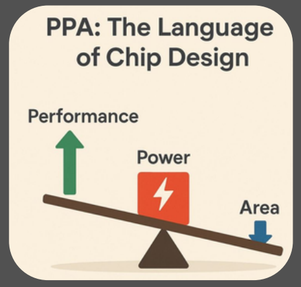
Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

Simulation Environment

RTL to Netlist Automated flow on Windows





WHY?

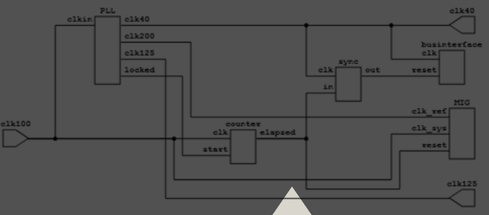
❌ **EDAPlayground is unreliable** (server **downtime**) → need local setup on Windows

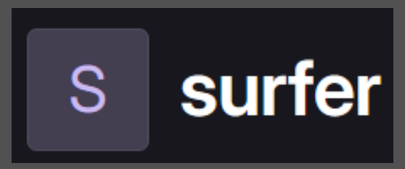
❌ **Manual RTL testing not scalable** → need automated pipeline to avoid typing several terminal commands

❌ **No hardware metrics** (area, cells, timing) → need synthesis + analysis

Solution: *Automated RTL-to-Netlist Flow – one-click RTL → Netlist → Reports*

While our team was designing the RTL, we realized and built the infrastructure that validates and measures that work.





Silicon Sprint
Competition
Round 2

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

Simulation Environment

What we built

The Pipeline: Code →Compile → Simulate → Verify → Synthesize → Report

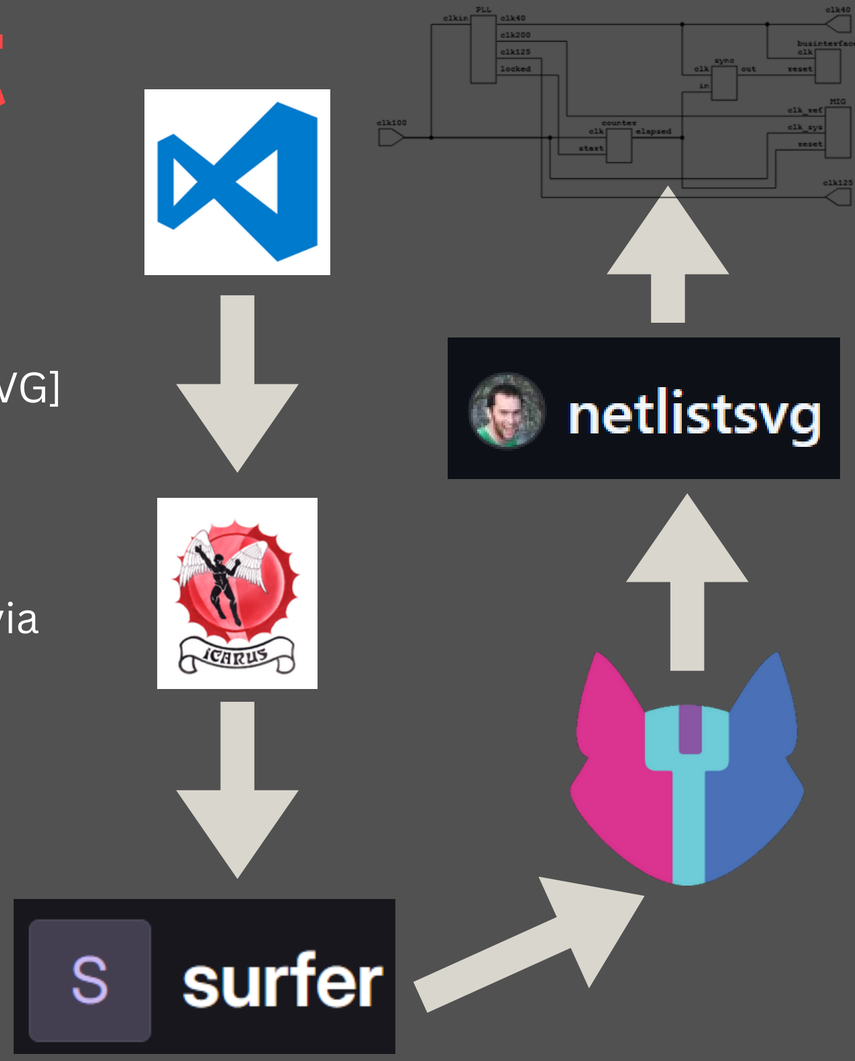
[IcarusVerilog] → [VVP Sim] → [Surfer Waveforms] → [Yosys+ABC] → [Area.rpt + SVG]

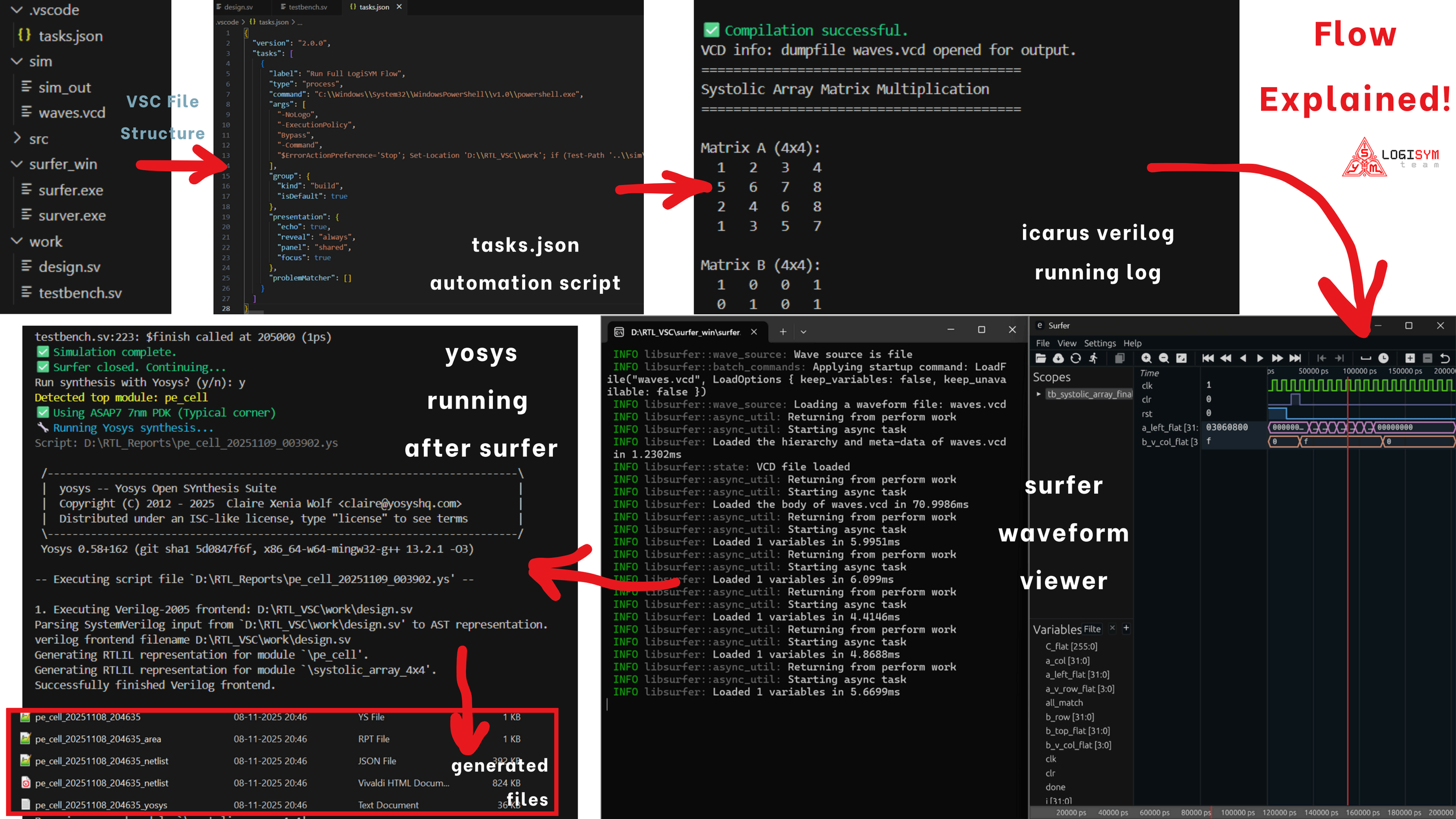
↓	↓	↓	↓	↓
Compile	Run Test	View Waveforms	Gate Mapping	Metrics

Execution: Single keystroke (Ctrl+Shift+B in VS Code) = entire flow automated via PowerShell!

Code + Syntax Highlighting: VS Code
Compilation and VCD generation: IcarusVerilog g2012
Waveform Viewer: Surfer
Synthesizer: Yosys 58.0
Netlist viewer: NetlistsSVG via node.js

Bonus: **This flow is reusable for any other Verilog design on Windows that we want to implement in the future.**




.vscode
tasks.json
sim
sim_out
waves.vcd
src
surfer_win
surfer.exe
surver.exe
work
design.sv
testbench.sv
VSC File Structure
design.sv
testbench.sv
tasks.json
.vscode > tasks.json > ...
{
"version": "2.0.0",
"tasks": [
{
"label": "Run Full LogiSYM Flow",
"type": "process",
"command": "C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe",
"args": [
"-NoLogo",
"-ExecutionPolicy",
"Bypass",
"-Command",
"$ErrorActionPreference='Stop'; Set-Location 'D:\\RTL_VSC\\work'; if (Test-Path '..\\sim
],
"group": {
"kind": "build",
"isDefault": true
},
"presentation": {
"echo": true,
"reveal": "always",
"panel": "shared",
"focus": true
},
"problemMatcher": []
}
]
}
tasks.json
automation script
Compilation successful.
VCD info: dumpfile waves.vcd opened for output.
Systolic Array Matrix Multiplication
Matrix A (4x4):
1 2 3 4
5 6 7 8
2 4 6 8
1 3 5 7
Matrix B (4x4):
1 0 0 1
0 1 0 1
icarus verilog
running log
Flow
Explained!
LOGISYM
team
testbench.sv:223: $finish called at 205000 (1ps)
Simulation complete.
Surfer closed. Continuing...
Run synthesis with Yosys? (y/n): y
Detected top module: pe_cell
Using ASAP7 7nm PDK (Typical corner)
Running Yosys synthesis...
Script: D:\RTL_Reports\pe_cell_20251109_003902.ys
yosys -- Yosys Open SYnthesis Suite
Copyright (C) 2012 - 2025 Claire Xenia Wolf <claire@yosyshq.com>
Distributed under an ISC-like license, type "license" to see terms
Yosys 0.58+162 (git sha1 5d0847f6f, x86_64-w64-mingw32-g++ 13.2.1 -O3)
-- Executing script file `D:\RTL_Reports\pe_cell_20251109_003902.ys' --
1. Executing Verilog-2005 frontend: D:\RTL_VSC\work\design.sv
Parsing SystemVerilog input from `D:\RTL_VSC\work\design.sv' to AST representation.
verilog frontend filename D:\RTL_VSC\work\design.sv
Generating RTLIL representation for module `\pe_cell'.
Generating RTLIL representation for module `\systolic_array_4x4'.
Successfully finished Verilog frontend.
yosys
running
after surfer
pe_cell_20251108_204635 08-11-2025 20:46 YS File 1 KB
pe_cell_20251108_204635_area 08-11-2025 20:46 RPT File 1 KB
pe_cell_20251108_204635_netlist 08-11-2025 20:46 JSON File 382 KB
pe_cell_20251108_204635_netlist 08-11-2025 20:46 Vivaldi HTML Docum... 824 KB
pe_cell_20251108_204635_yosys 08-11-2025 20:46 Text Document 36 KB
generated
files
D:\RTL_VSC\surfer_win\surfer
INFO libsurfer::wave_source: Wave source is file
INFO libsurfer::batch_commands: Applying startup command: LoadFile("waves.vcd", LoadOptions { keep_variables: false, keep_unavailable: false })
INFO libsurfer::wave_source: Loading a waveform file: waves.vcd
INFO libsurfer::async_util: Returning from perform work
INFO libsurfer::async_util: Starting async task
INFO libsurfer: Loaded the hierarchy and meta-data of waves.vcd in 1.2302ms
INFO libsurfer::state: VCD file loaded
INFO libsurfer::async_util: Returning from perform work
INFO libsurfer::async_util: Starting async task
INFO libsurfer: Loaded the body of waves.vcd in 70.9986ms
INFO libsurfer::async_util: Returning from perform work
INFO libsurfer::async_util: Starting async task
INFO libsurfer: Loaded 1 variables in 5.9951ms
INFO libsurfer::async_util: Returning from perform work
INFO libsurfer::async_util: Starting async task
INFO libsurfer: Loaded 1 variables in 6.099ms
INFO libsurfer::async_util: Returning from perform work
INFO libsurfer::async_util: Starting async task
INFO libsurfer: Loaded 1 variables in 4.4146ms
INFO libsurfer::async_util: Returning from perform work
INFO libsurfer::async_util: Starting async task
INFO libsurfer: Loaded 1 variables in 4.8688ms
INFO libsurfer::async_util: Returning from perform work
INFO libsurfer::async_util: Starting async task
INFO libsurfer: Loaded 1 variables in 5.6699ms
surfer
waveform
viewer
Surfer
File View Settings Help
Scopes
tb_systolic_array_final
Time
clk 1
clr 0
rst 0
a_left_flat [31: 03060800
b_v_col_flat [3 f
50000 ps
100000 ps
150000 ps
Variables Filte
C_flat [255:0]
a_col [31:0]
a_left_flat [31:0]
a_v_row_flat [3:0]
all_match
b_row [31:0]
b_top_flat [31:0]
b_v_col_flat [3:0]
clk
clr
done
i [31:0]
20000 ps
40000 ps
60000 ps
80000 ps
100000 ps
120000 ps
140000 ps
160000 ps
180000 ps
200000

Silicon Sprint
Competition
Round 2

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

Simulation Environment

Outputs:

Tech agnostic - ABC Area Report: 668 gate cells (AND×28, NAND×40, NOR×257, XOR×24, XNOR×171, OR×10, NOT×27, ORNOT×42, ANDNOT×30, MUX×1**) , 4254+ transistors, 623 wires** considering generic 130nm values: **57,065 µm²** (reference)

PDK Synthesis pre-crash: 729 cells (AND×23, NAND×53, NOR×304, XOR×14, XNOR×222, OR×8, NOT×15, ORNOT×26, ANDNOT×26**)**

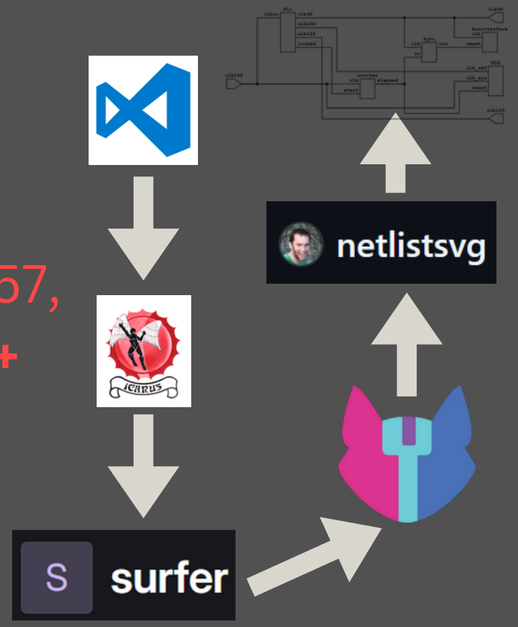
Difference Observed: Gate distribution +8.9% cells, +18.3% NOR usage

Netlist Visualization:

See in next slide!

Testbench case:

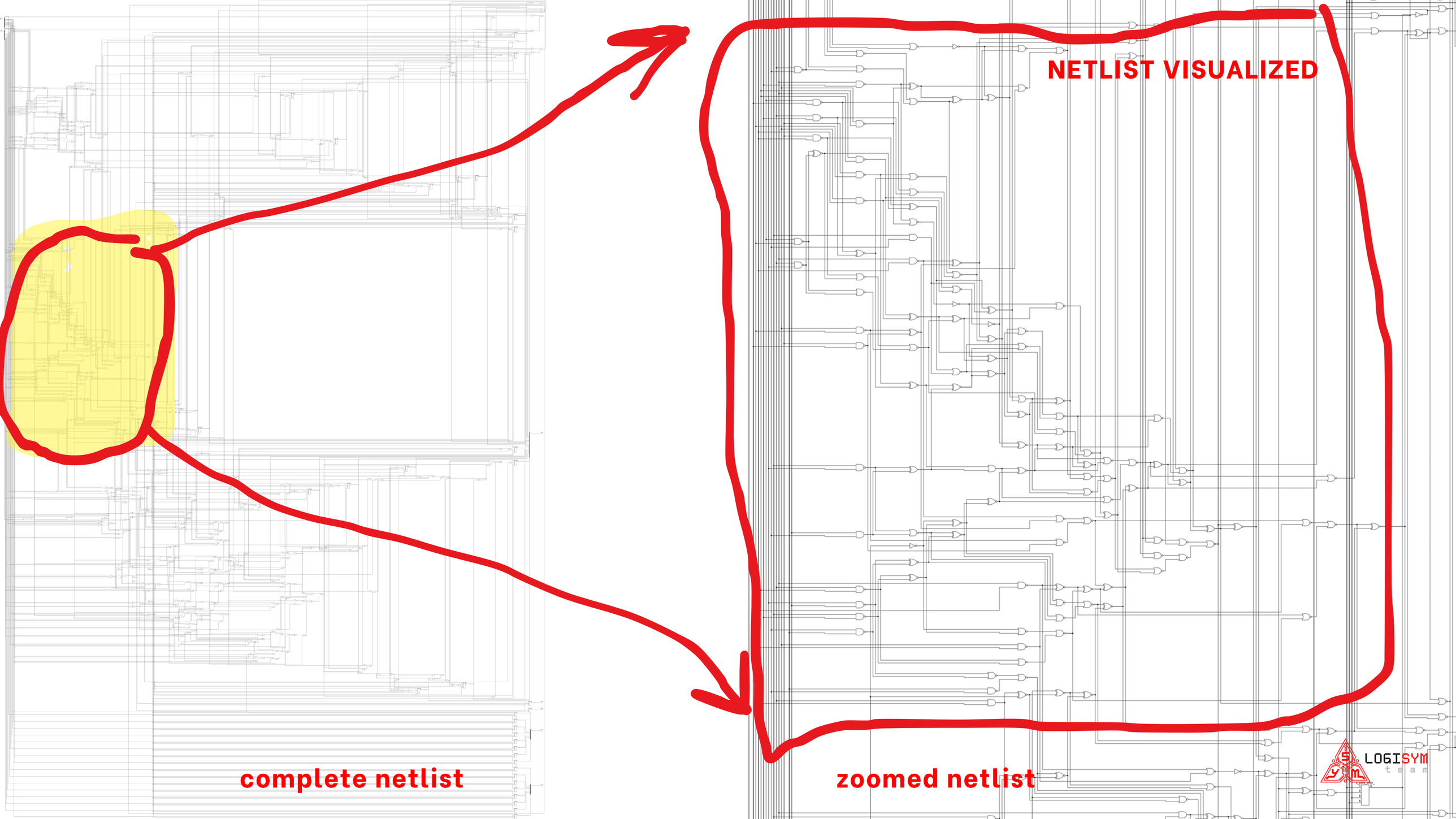
All Passed!



Challenges we faced:

Challenge 1: Yosys Language Limitations
our original RTL used unpacked arrays (Yosys 0.58 doesn't fully support SystemVerilog 2012)
Solution: Flattened array structure, re-verified with testbench → PASSed the tb

Challenge 2: PDK Integration on Windows
Tried Google Skywater 130nm PDK → Liberty file syntax errors (lines 549, 8546, 13331)
Tried OpenROAD ASAP7 7nm PDK → Liberty file parsing crash (line 4151)
Root cause: Yosys Liberty parser has Windows-specific issues (known limitation)
Decision: Stick with generic gate mapping (reliable, portable)


NETLIST VISUALIZED
complete netlist
zoomed netlist
LOGISYM
team

Silicon Sprint
Competition
Round 2

Yashvardhan Singh
Megh Ashok Giri
Siddharth Penumatsa

LOGISYM team

THANKYOU