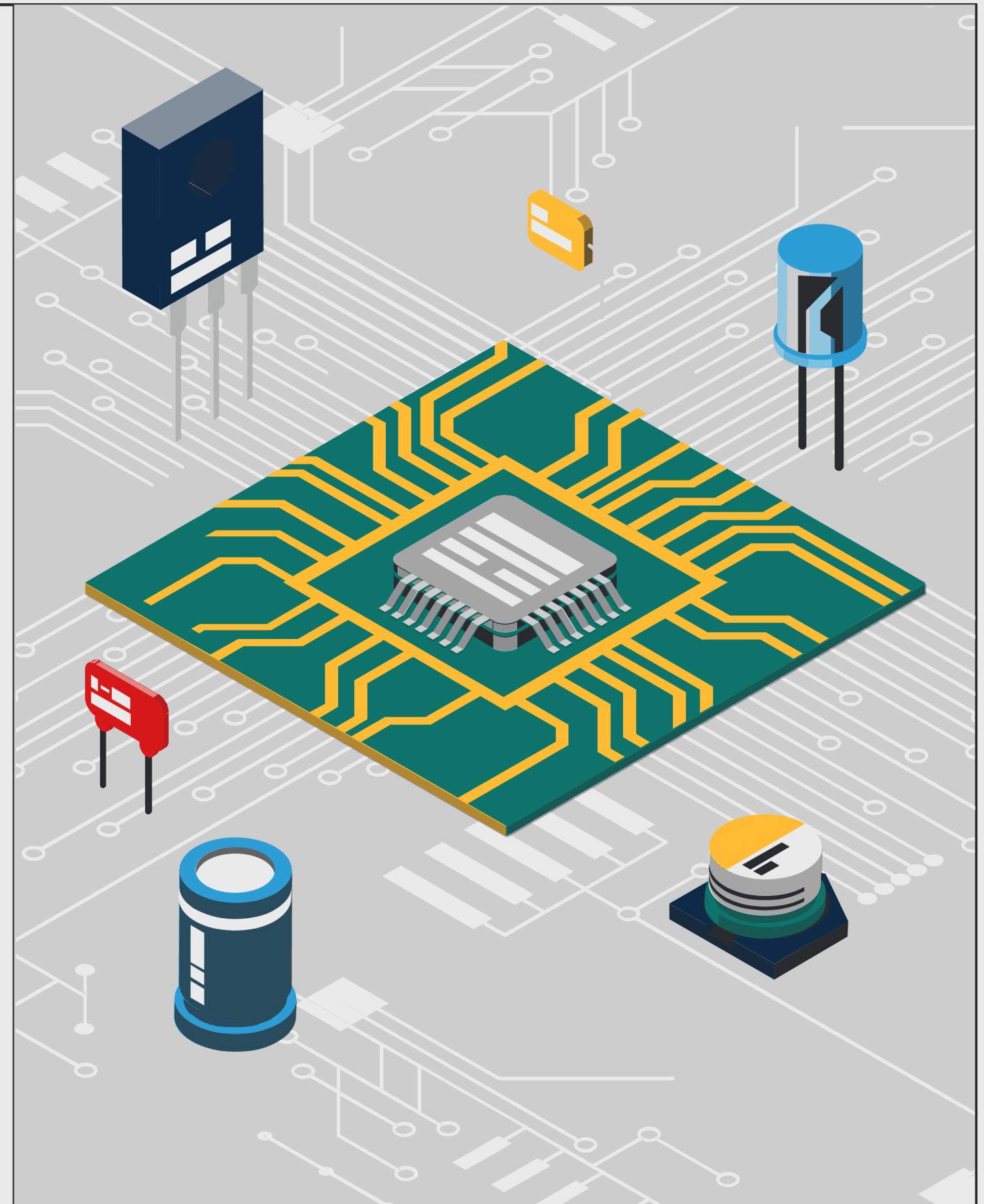


Structured ASIC Platform

Checkpoint 1 Presentation

Mostafa Gaafar
Omar Saqr
Aabed Elghadban



Project Status and GitHub Workflow

Implemented Phase 1 and Phase 2 successfully using the specified workflow.

The screenshot shows a GitHub search results page with the following filters applied:

- Filters: is:pr is:closed
- Labels: 21
- Milestones: 0
- New pull request

The search results list six closed pull requests:

- #20 Feature/issue 6 map output (phase-2)**: Merged 17 hours ago, approved.
- #18 Implement Hybrid Move Set for Simulated Annealing (phase-2)**: Merged 17 hours ago, approved, 7 tasks done.
- #17 Implement Simulated Annealing Core Algorithm (fixes #4) (phase-2)**: Merged 17 hours ago, approved, 6 tasks done.
- #14 Phase 2 Task 1A implementation on Feature/placement implementation**: Merged yesterday.
- #11 Fix #10: Fix cell positioning in visualization - use lower-left corner**: Merged 2 weeks ago.
- #9 Feature/phase1 database validation**: Merged 2 weeks ago.

Project Status and GitHub Workflow

Implemented Phase 1 and Phase 2 successfully using the specified workflow. Some of the issues can be seen here

Open 4 Closed 16

Author Labels Projects Milestones Assignees Newest

- Heatmaps of the runs done are missing** #26 · mostafa21314 opened 48 minutes ago
- Inefficient SA Algorithm** #25 · mostafa21314 opened 2 hours ago
- Issue #10: Integrate Placer into Main Flow and Add Console Reporting** #24 · AabedCodes opened 11 hours ago
- Issue #9: Implement SA Knob Exploration and Analysis** #23 · AabedCodes opened 11 hours ago
- Issue #8: Implement Net Length Histogram Visualization** #22 · by AabedCodes was closed 6 hours ago
- Issue #7: Implement Placement Density Heatmap Visualization** #21 · by AabedCodes was closed 11 hours ago
- Implement Placement Output File Generation** phase-2 #19 · by omarsaqr12 was closed 17 hours ago
- Implement Hybrid Move Set for Simulated Annealing** phase-2 simulated-annealing #16 · by omarsaqr12 was closed 17 hours ago
- Simulated Annealing Core Algorithm for placement optimization.** phase-2 simulated-annealing #15 · by omarsaqr12 was closed 17 hours ago
- Optimize the greedy placement algo in placer.py from quadratic complexity to linear complexity or logarithmic complexity** #13 · by mostafa21314 was closed 13 hours ago
- Implement place_design() function in placer.py** #12 · by mostafa21314 was closed 13 hours ago
- Fix issue in visualization** #10 · by omarsaqr12 was closed 2 weeks ago
- Code review, cleanup, and documentation for Phase 1** code-review documentation phase-1



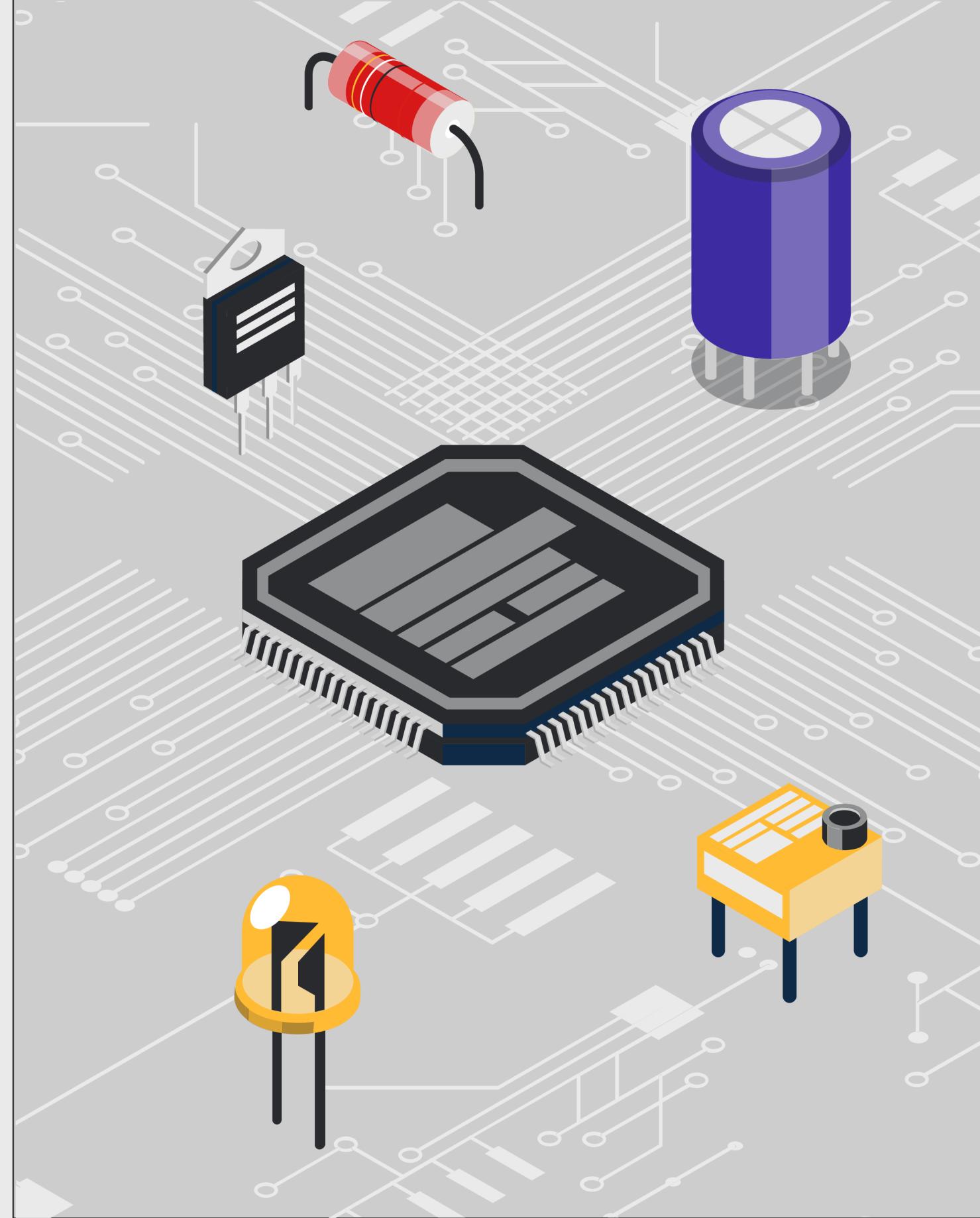
Project Status and GitHub Workflow

Implemented Phase 1 and Phase 2 successfully using the specified workflow.

Pull requests as well as the branches used are included here.

Author ▾		Label ▾	Projects ▾	Milestones ▾	Reviews ▾	Assignee ▾	Sort ▾
0 Open	✓ 6 Closed						
Feature/issue 6 map output phase-2	#20 by omarsaqr12 was merged 17 hours ago • Approved			①			
Implement Hybrid Move Set for Simulated Annealing phase-2	#18 by omarsaqr12 was merged 17 hours ago • Approved 7 tasks done			①			
Implement Simulated Annealing Core Algorithm (fixes #4) phase-2	#17 by omarsaqr12 was merged 17 hours ago • Approved 6 tasks done			①			
Phase 2 Task 1A iplementation on Feature/placement implementation	#14 by mostafa21314 was merged yesterday						
Fix #10: Fix cell positioning in visualization - use lower-left corne...	#11 by omarsaqr12 was merged 2 weeks ago						
Feature/phase1 database validation	#9 by omarsaqr12 was merged 2 weeks ago			④			

Branch	Updated	Check status	Behind	Ahead	Pull request
main	18 hours ago	Default			
feature/placement_implementation	1 hour ago		9	1	
fix/inefficient-sa-algorithm	2 hours ago		3	3	
Phase2-Visulization	6 hours ago		0	6	
feature/issue-6-map-output	18 hours ago		3	0	#20
feature/issue-5-hybrid-moves	20 hours ago		6	0	#18
feature/issue-4-sa-core	20 hours ago		7	0	#17
fix/10-visualization	2 weeks ago		16	0	#11
feature/phase1-database-validation	2 weeks ago		17	0	#9



Phase 1 Results

After implementing parsing, validation, and visualization

Validation implementation and results

Comparison Table

Design	NAND2	OR2	DFF	INV	BUF	Status
6502	2.5%	3.3%	2.2%	2.8%	3.3%	<input checked="" type="checkbox"/> Pass
Arith	0.4%	0.5%	0.2%	0.5%	0.7%	<input checked="" type="checkbox"/> Pass
AES_128	95.2%	91.2%	77.4%	49.0%	46.5%	<input checked="" type="checkbox"/> Pass
Z80	6.9%	17.5%	2.5%	7.8%	0.8%	<input checked="" type="checkbox"/> Pass

CONB	DECAP3	DECAP4
0.0%	0.0%	0.0%
0.0%	0.0%	0.0%
0.0%	0.0%	0.0%
0.0%	0.0%	0.0%

Validation implementation and results

Design: aes_128

Cell Type	Required	Available	Utilization	Status
BUF (clkbuf_4)	4523	9720	46.5%	✓ OK
INV (clkinv_2)	6352	12960	49.0%	✓ OK
CONB (conb_1)	4	9720	0.0%	✓ OK
DECAP3 (decap_3)	0	9720	0.0%	✓ OK
DECAP4 (decap_4)	0	9720	0.0%	✓ OK
DFBBP (dfbbp_1)	5018	6480	77.4%	✓ OK
NAND2 (nand2_2)	46283	48600	95.2%	✓ OK
OR2 (or2_2)	23639	25920	91.2%	✓ OK
TAP (tapvpwrvgnd_1)	0	25920	0.0%	✓ OK

VERDICT: ✓ PASSED

Design: z80

Cell Type	Required	Available	Utilization	Status
BUF (clkbuf_4)	77	9720	0.8%	✓ OK
INV (clkinv_2)	1011	12960	7.8%	✓ OK
CONB (conb_1)	3	9720	0.0%	✓ OK
DECAP3 (decap_3)	0	9720	0.0%	✓ OK
DECAP4 (decap_4)	0	9720	0.0%	✓ OK
DFBBP (dfbbp_1)	160	6480	2.5%	✓ OK
NAND2 (nand2_2)	3356	48600	6.9%	✓ OK
OR2 (or2_2)	4537	25920	17.5%	✓ OK
TAP (tapvpwrvgnd_1)	0	25920	0.0%	✓ OK

Design: arithmetic

Cell Type	Required	Available	Utilization	Status
BUF (clkbuf_4)	67	9720	0.7%	✓ OK
INV (clkinv_2)	68	12960	0.5%	✓ OK
CONB (conb_1)	3	9720	0.0%	✓ OK
DECAP3 (decap_3)	0	9720	0.0%	✓ OK
DECAP4 (decap_4)	0	9720	0.0%	✓ OK
DFBBP (dfbbp_1)	11	6480	0.2%	✓ OK
NAND2 (nand2_2)	187	48600	0.4%	✓ OK
OR2 (or2_2)	127	25920	0.5%	✓ OK
TAP (tapvpwrvgnd_1)	0	25920	0.0%	✓ OK

VERDICT: ✓ PASSED

Design: 6502

Cell Type	Required	Available	Utilization	Status
BUF (clkbuf_4)	322	9720	3.3%	✓ OK
INV (clkinv_2)	360	12960	2.8%	✓ OK
CONB (conb_1)	3	9720	0.0%	✓ OK
DECAP3 (decap_3)	0	9720	0.0%	✓ OK
DECAP4 (decap_4)	0	9720	0.0%	✓ OK
DFBBP (dfbbp_1)	143	6480	2.2%	✓ OK
NAND2 (nand2_2)	1228	48600	2.5%	✓ OK
OR2 (or2_2)	843	25920	3.3%	✓ OK
TAP (tapvpwrvgnd_1)	0	25920	0.0%	✓ OK

VERDICT: ✓ PASSED

Visualizer

What the File Does

Generates a complete visual map of the ASIC fabric:

- Die + core boundaries
- Fabric cell slots (color-coded by type)
- I/O pins (input/output)

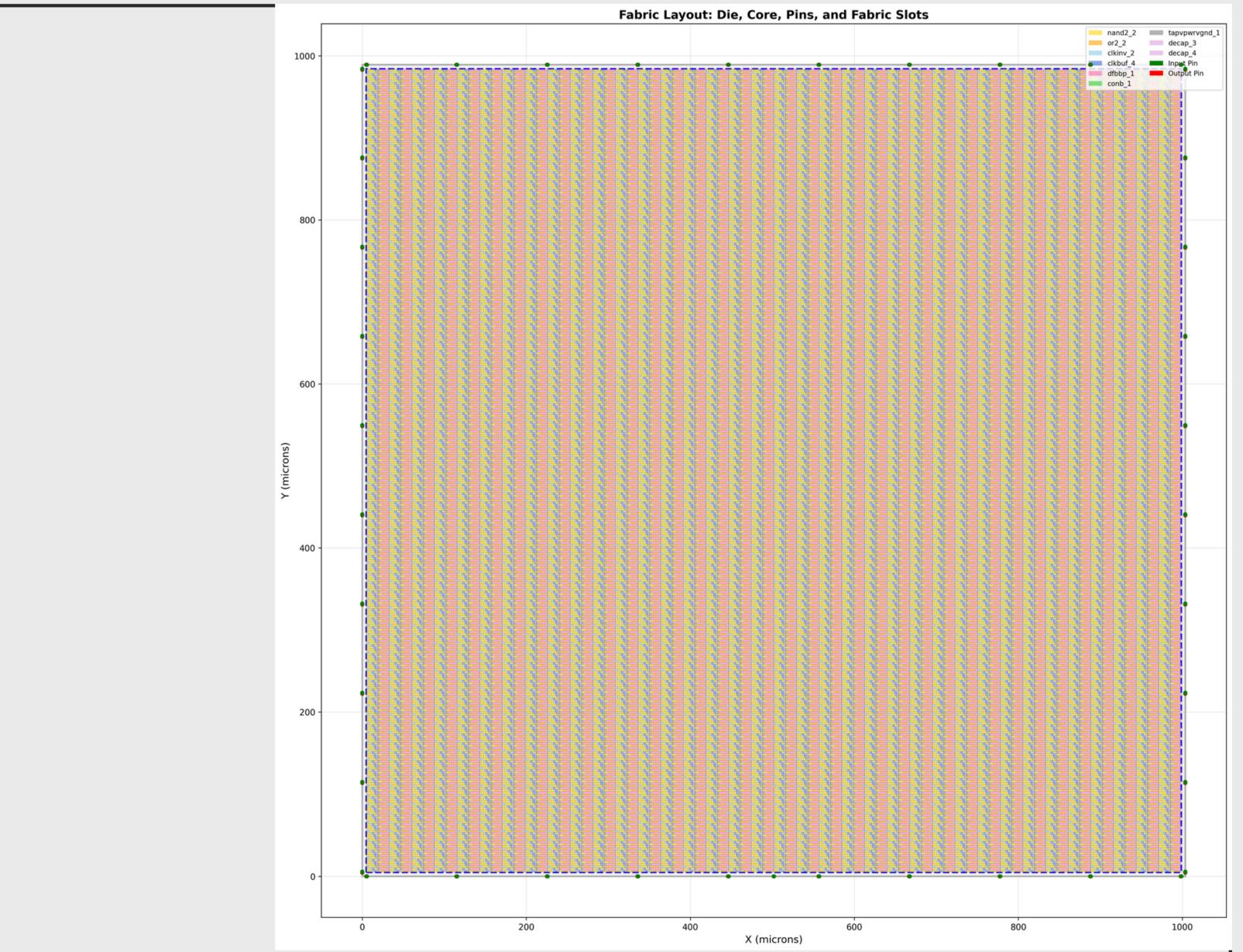
Core Functions

1. Draw Die & Core

- Gray rectangle = die boundary
- Blue dashed rectangle = core boundary
- Core positioned using `core_margin_um`

2. Draw Fabric Slots

- Each cell type assigned a unique color
- Drawn as rectangles at (x, y) using real cell dimensions
- Semi-transparent to show density

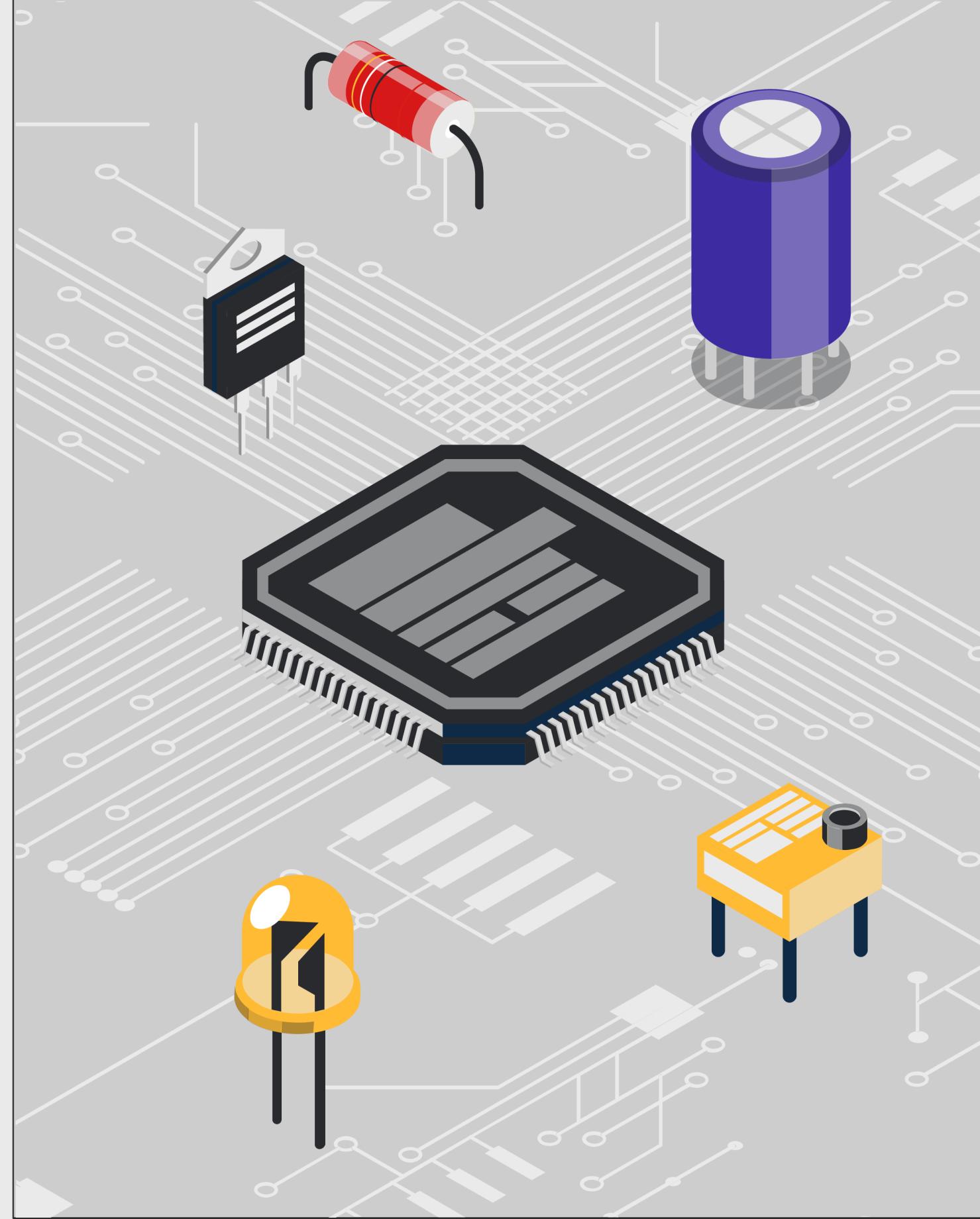


3. Draw I/O Pins

- Green = INPUT
- Red = OUTPUT
- Drawn as small circles on top of slots

4. Add Legend

- Shows color for each cell type
- Includes pin colors (input/output)



Phase 2 Results

After implementing Greedy + SA placing algorithms. As well as analysis and visualization of results

Greedy Key implementation details

Optimizations done

Net-to-Cells Index - Eliminates $O(N)$ scans for neighbor lookups

Incremental Score Updates - Only updates affected cells, not all cells

Efficient I/O Placement - Single pass instead of nested loops

```
# Original: O(N) per lookup
for other_cell in netlist_graph: # Scan all N cells
    if shares_net(cell, other_cell):
        neighbors.append(other_cell)
```

```
# New : Build once:  $O(N \times M)$ 
net_index = {net_id: set_of_cells_on_that_net}
```

```
# Query: O(1) per net
for net_id in cell_nets:
    neighbors.extend(net_index[net_id])
```

For each net, we build a mapping to the set of cells that
are connected to that net.

```
# Original:  $O(N)$  cells  $\times$   $O(N)$  neighbor lookups =  $O(N^2)$ 
# per iteration
for iteration in range(N):
    for cell in unplaced_cells: # O(N)
        score[cell] = count_placed_neighbors(cell) # O(N)
        lookup
        best = max(unplaced_cells, key=score)
        place(best)
# Total:  $O(N^3)$ 
```

```
# Optimized: Only rescore cells on same nets as placed cell
place(best_cell)
affected_cells = get_cells_on_same_nets(best_cell) #  $O(M \times N_{avg})$ 
for cell in affected_cells: # Typically 10-50 cells
    score[cell] = count_placed_neighbors(cell) # Now O(1) with index
# Per iteration:  $O(k \times d)$  where  $k \approx 10$ ,  $d \approx 10$ 
# Total:  $O(N \times k \times d)$ 
```

Only update cells affected by the just-placed cell

Greedy Key implementation details

Optimizations done

Net-to-Cells Index - Eliminates $O(N)$ scans for neighbor lookups

Incremental Score Updates - Only updates affected cells, not all cells

Efficient I/O Placement - Single pass instead of nested loops

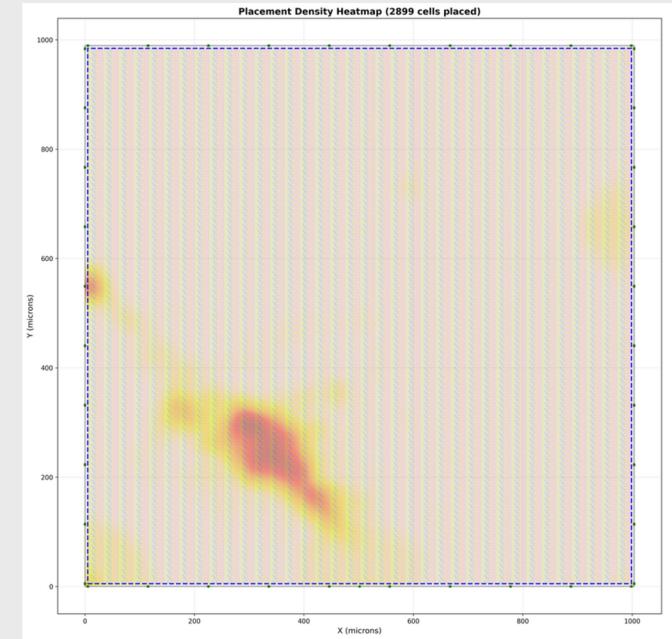
Single pass with pre-computed mappings

```
# Original: O(nets × cells)
for net_id in io_nets:
    for cell in all_cells: # O(N) scan per net
        if cell connects to net_id:
            place_near_io_pin(cell, net_id)

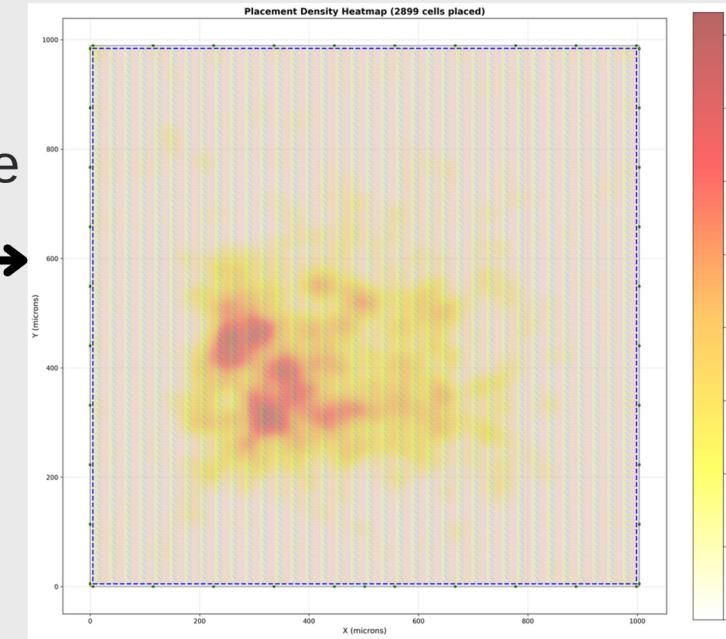
            # Pre-compute: O(N)
cell_to_nets = {cell: set_of_nets_it_connects_to}
net_to_pins = {net_id: list_of_pin_locations}

            # Place: O(C) where C = I/O cells
for cell in io_cells: # Already filtered
    # Get ALL pins this cell connects to
pin_locations = [net_to_pins[net] for net in cell_to_nets[cell]]
    # Place at barycenter of all connected pins
    target = barycenter(pin_locations)
    place_at_nearest_slot(cell, target)
```

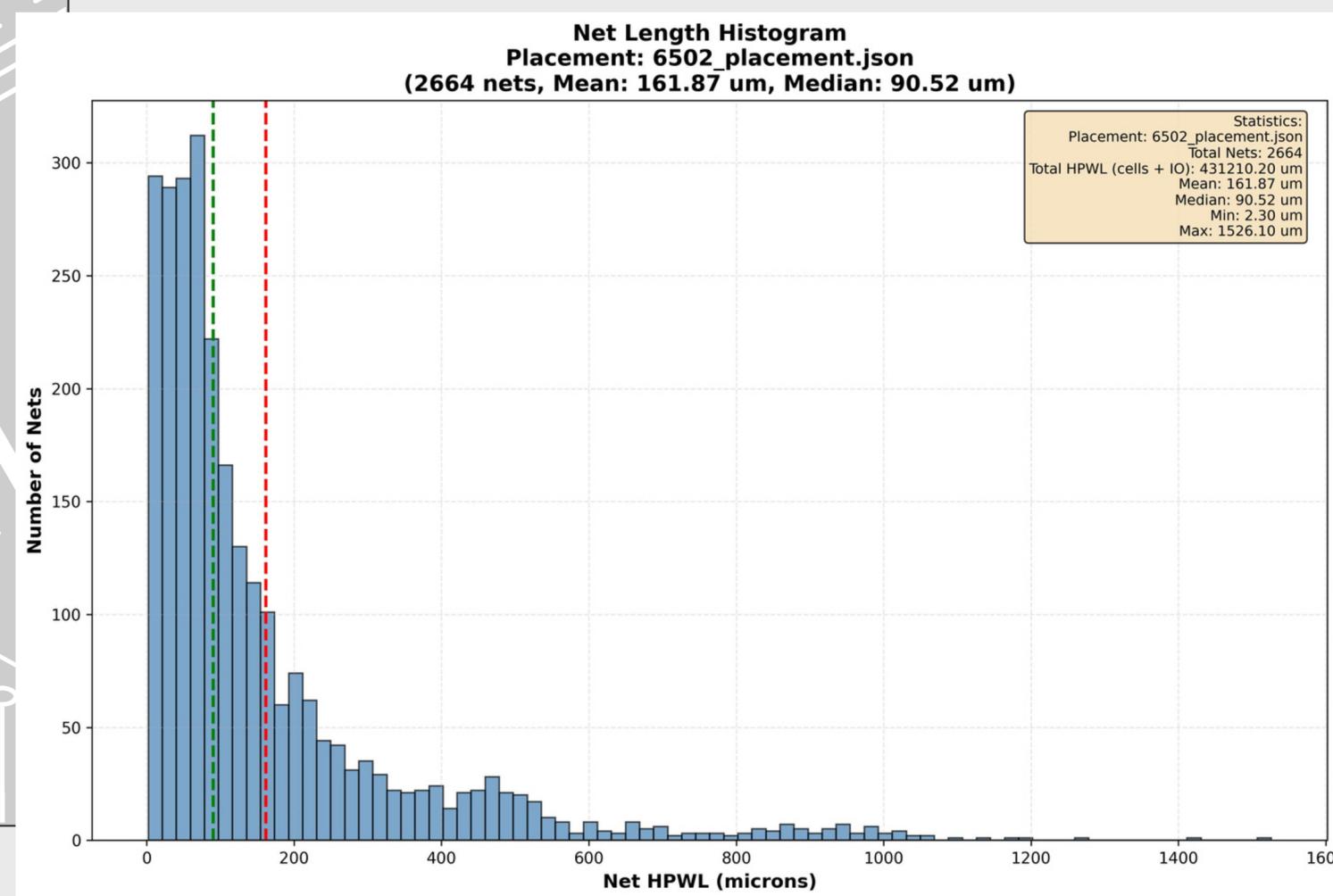
SA effect on 6502



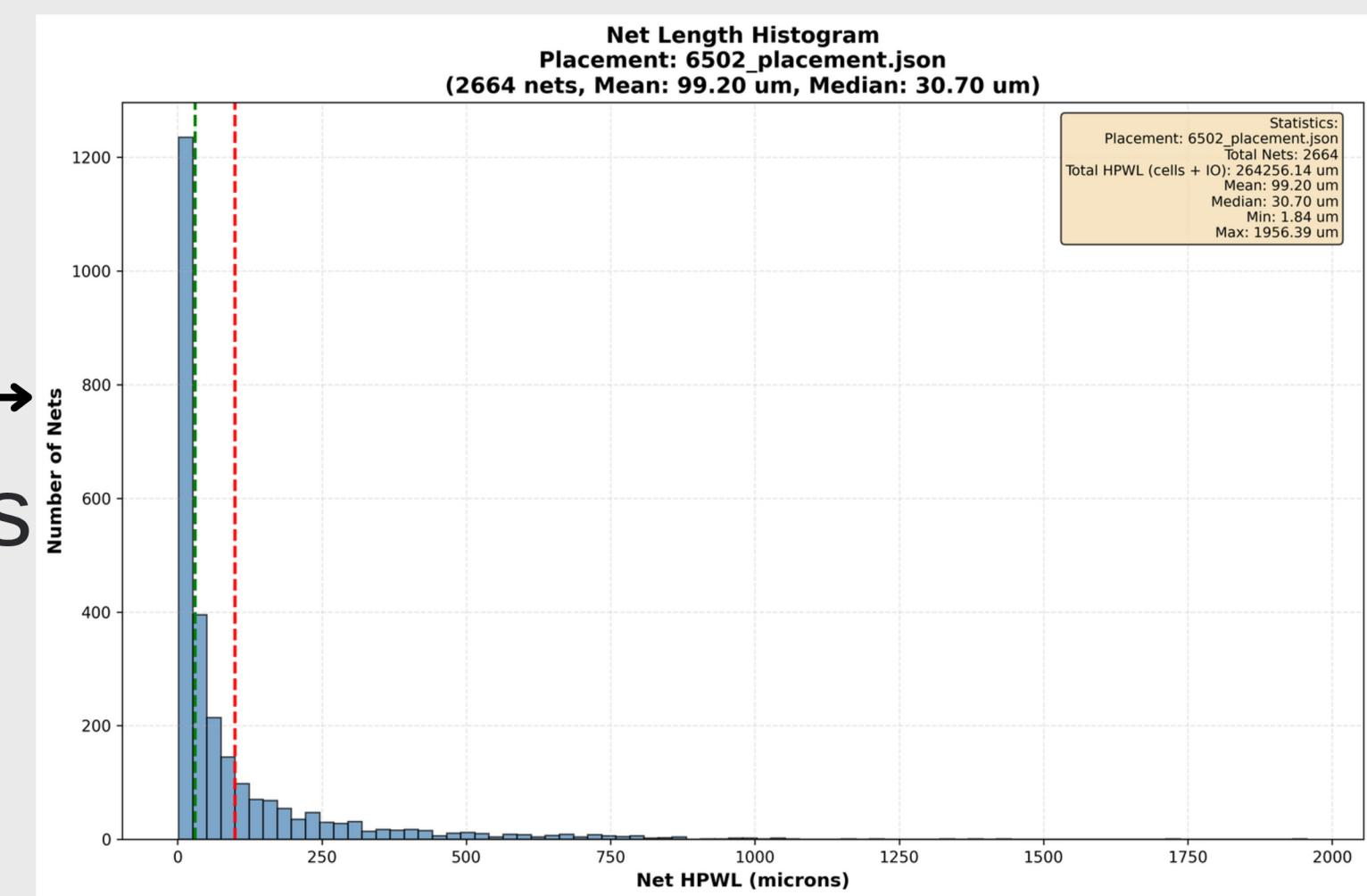
$T_{initial} = 10000 \times \text{initial_HPWL}$
This increased running time unnecessarily, and didn't provide many gains, so we will adjust 10000 to 10s range.



Reduction = $431,210.20 \mu\text{m} - 264,256.14 \mu\text{m}$
= $166,954.06 \mu\text{m}$
Percentage reduction = 38.72% reduction



Max wire-length increased
70s->5000s



SA Key Implementation details

Heat Map:

Greedy:

Very strong bias to place cells near IO pins, creating unnatural pulls.

Locks into poor local minima:

High-density pockets

- greedy clustered many mutually connected cells in the middle, those nets get lower HPWL.
- But nets that connect cluster members to distant cells see increased bounding boxes (they now span from the middle to the edge) → HPWL up for those nets.

SA:

SA breaks the greedy pattern by:

Random exploration

Cooling-based acceptance of uphill (worse) moves early on

As a result:

Clusters loosen. In this instance loosening clusters reduced total HPWL because many nets were global.

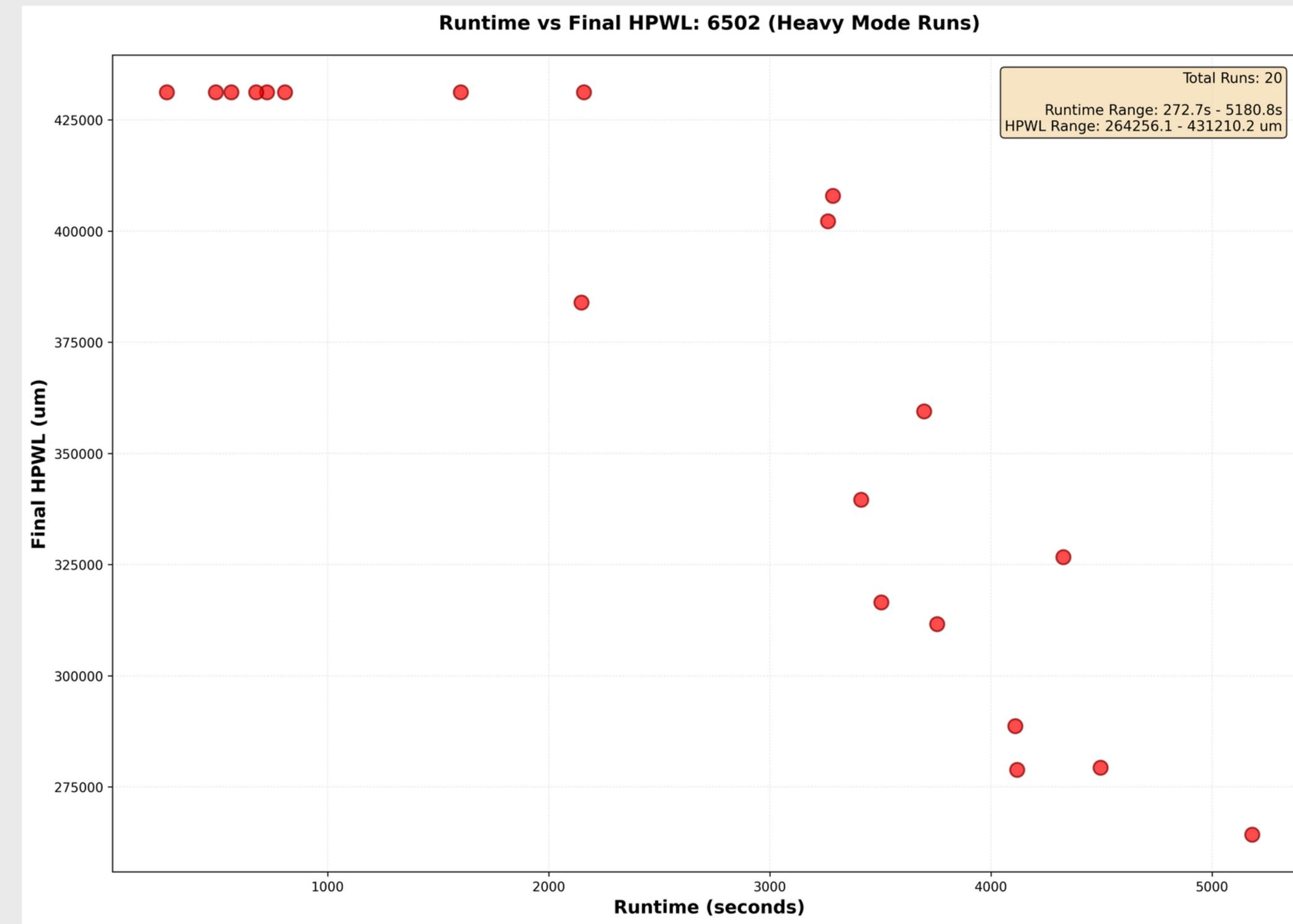
Histogram:

Median(88.8->31.5), mean(155.5->95.6) were reduced noticeably,

SA spreads the design to reduce congestion → some long nets get longer.

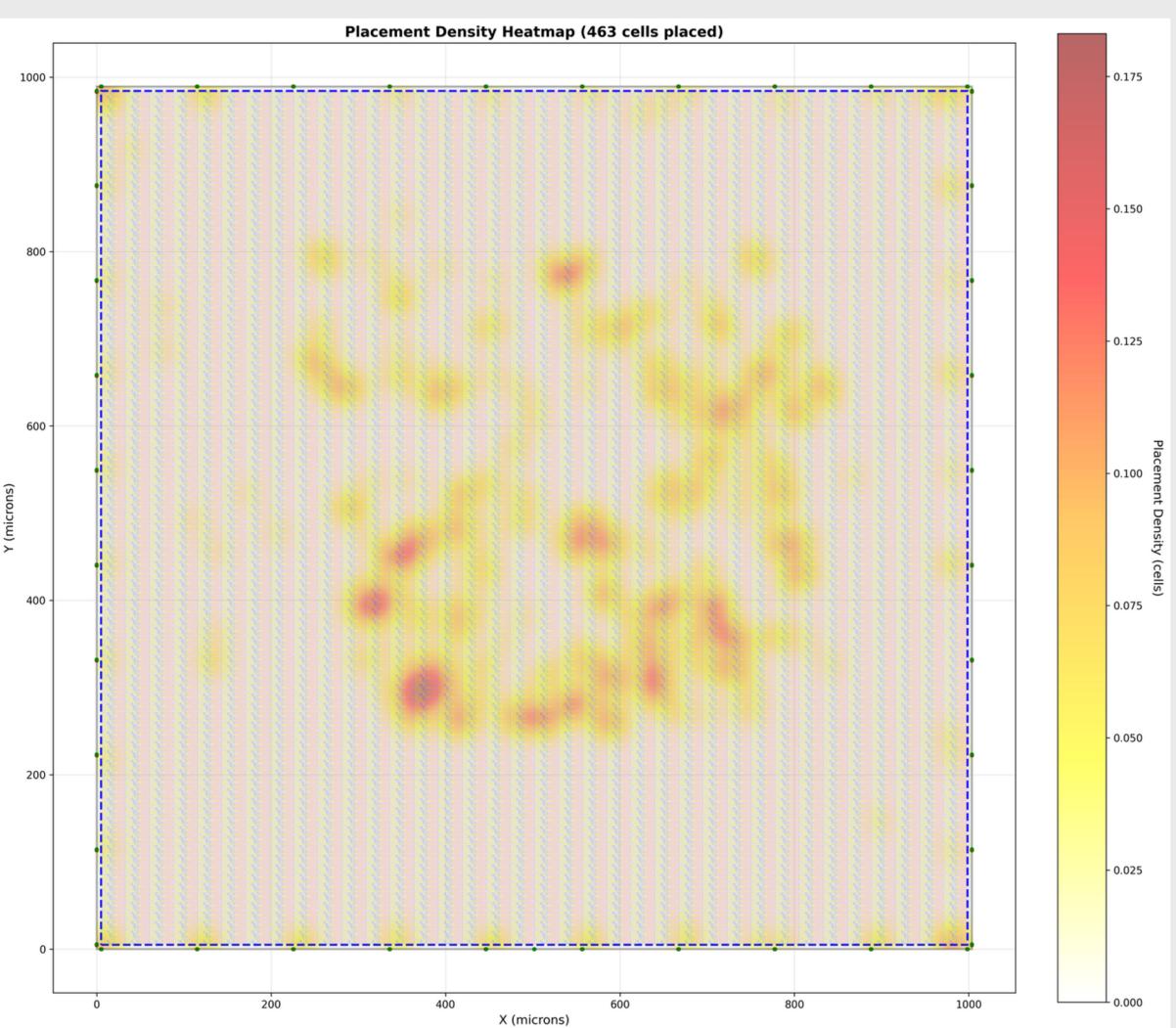
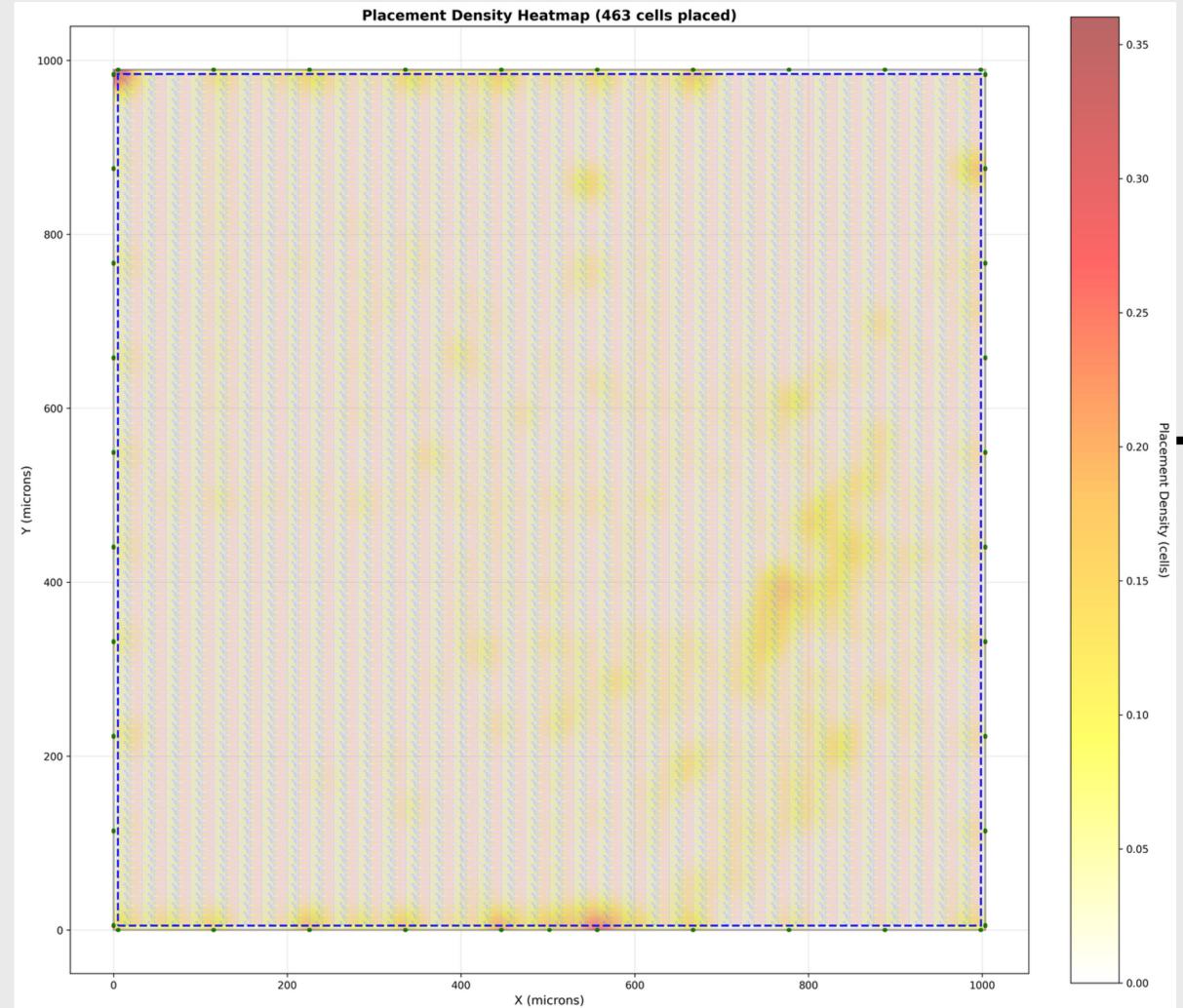
But there are very few of them, so overall average HPWL still drops.

SA effect on 6502

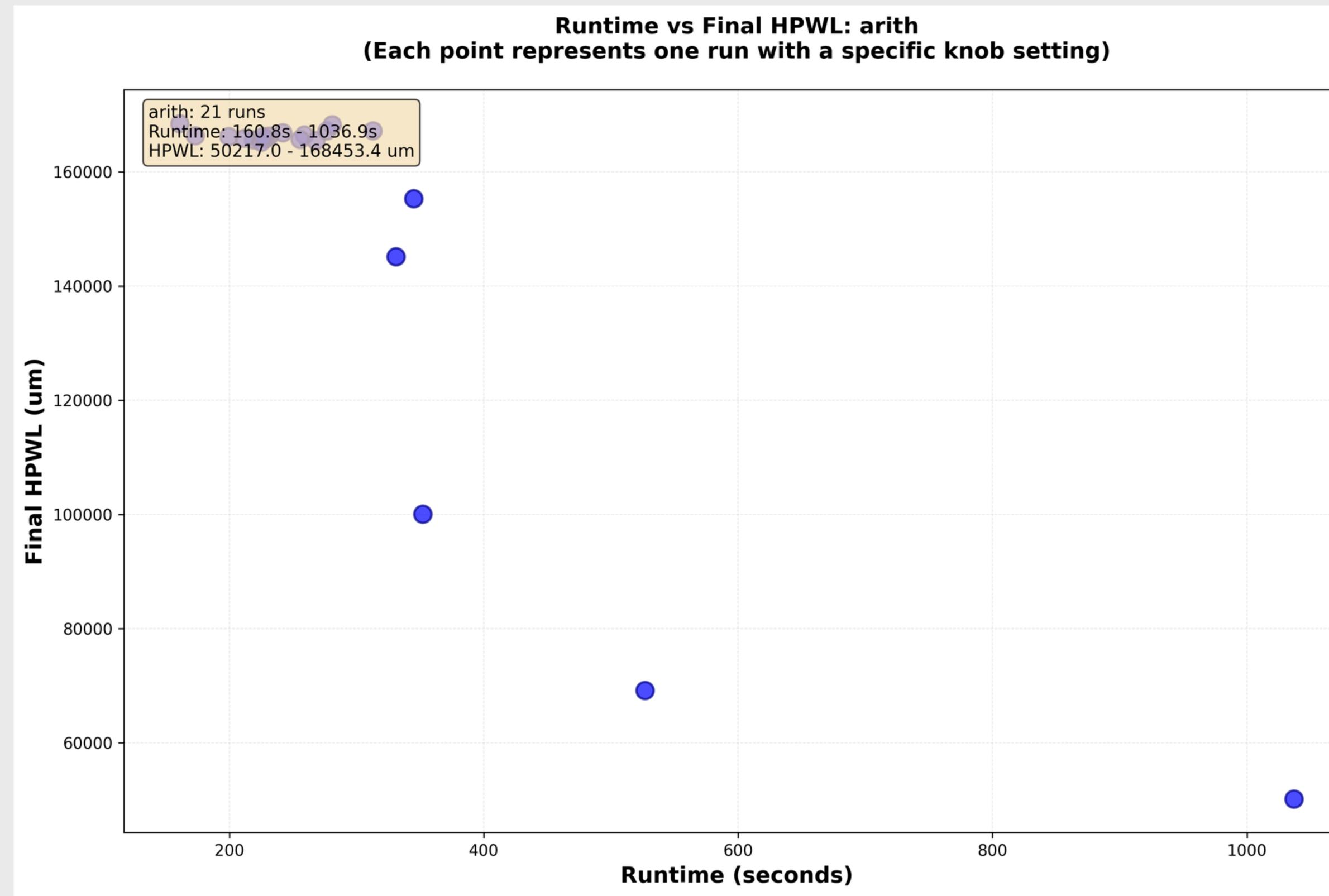


SA effect of Arith

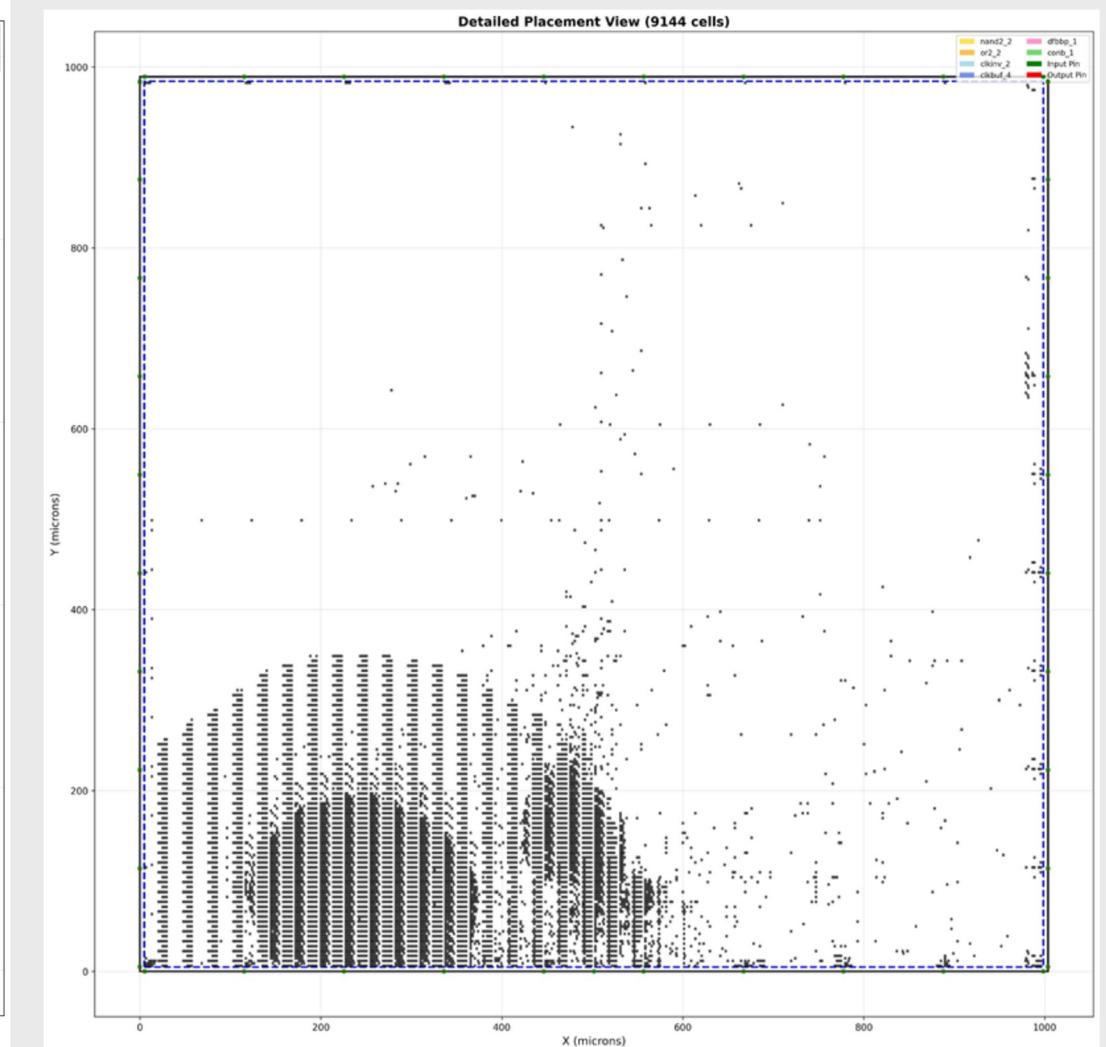
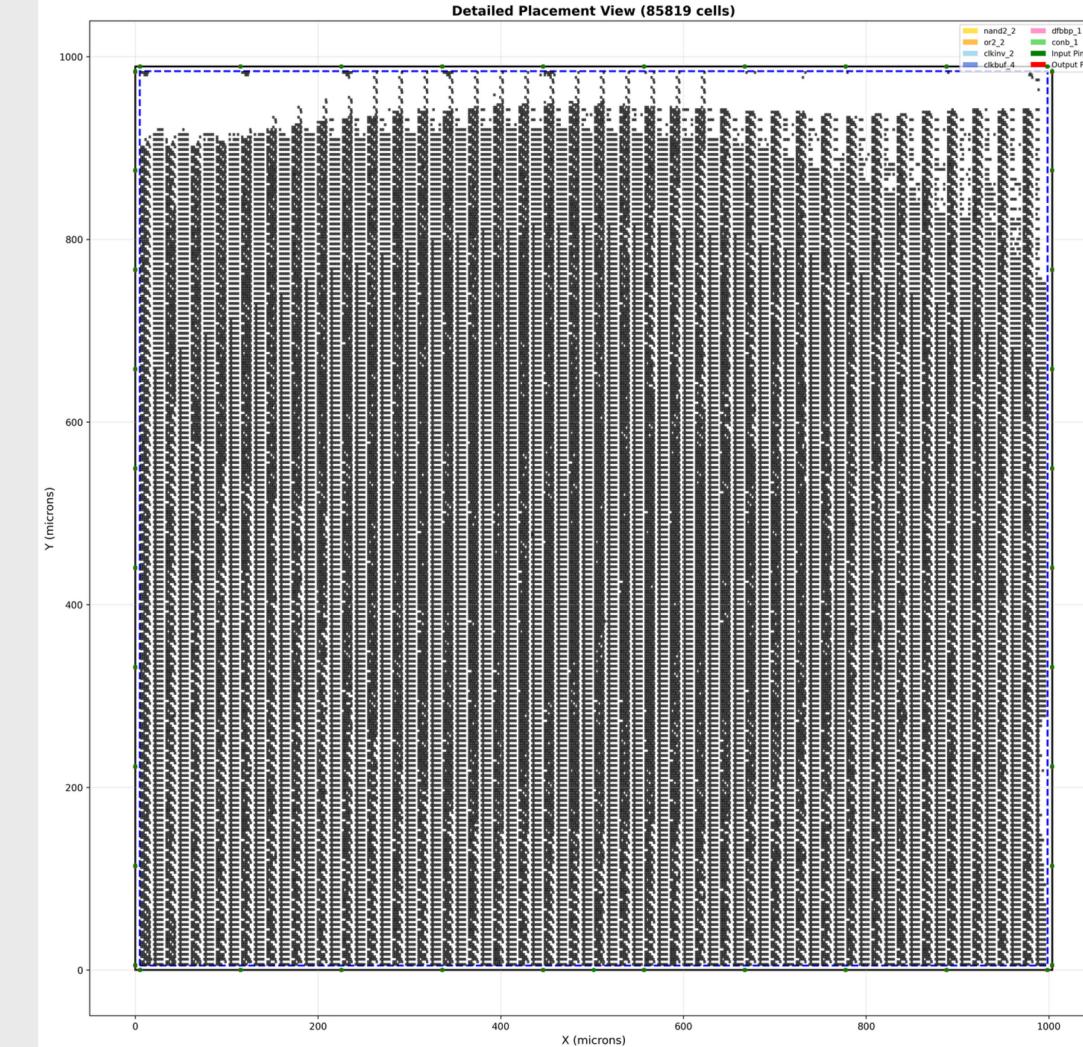
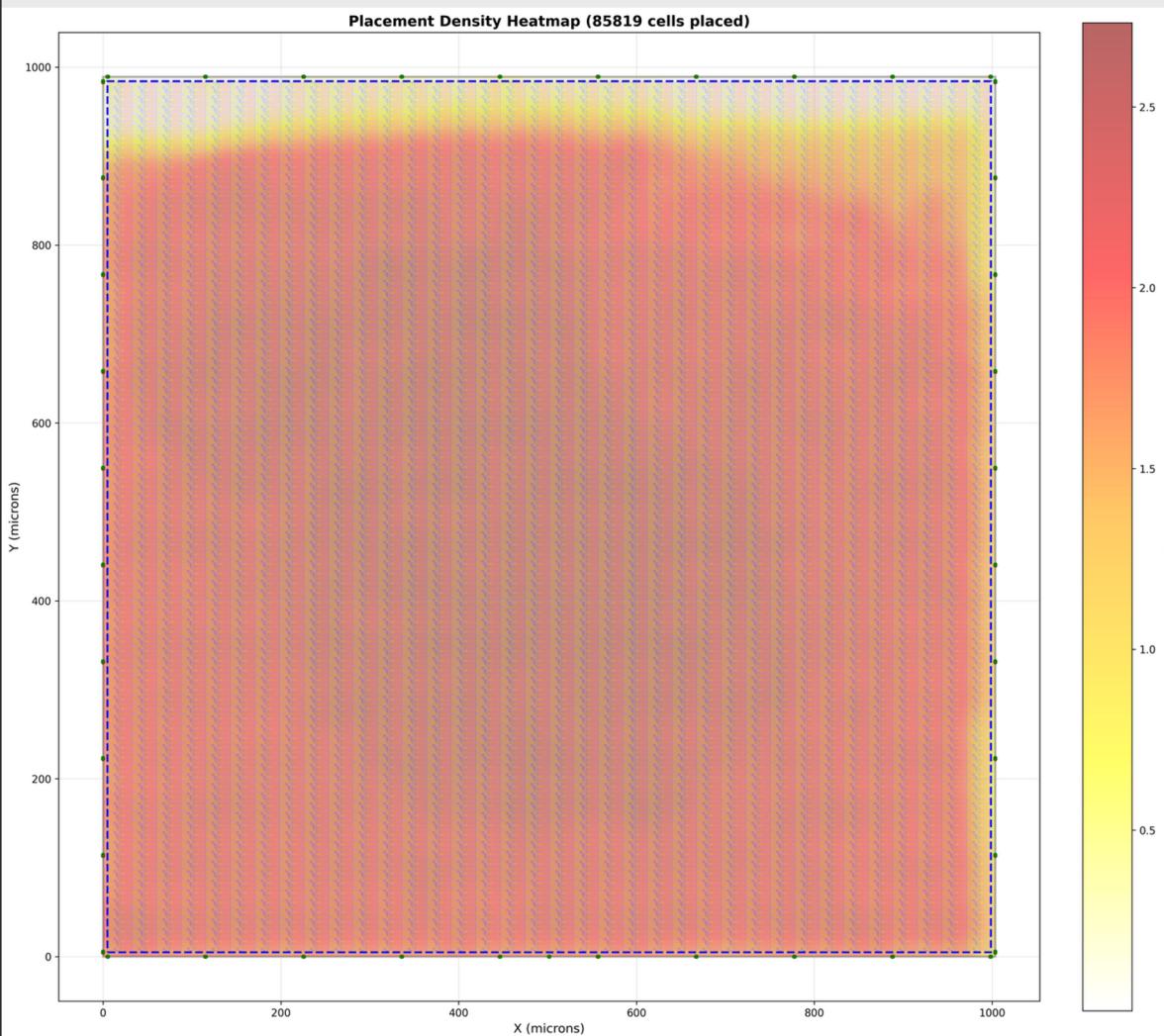
Arith



SA effect of Arith



Results - placement density



aes_128

aes_128

z80

SA Key Implementation details

Key Idea:

SA allows exploring better placements by occasionally accepting worse moves early on

Benefits

Escapes local minima by random exploration at high temperature

Performs fine-tuning at low temperature (few bad moves accepted)

Tries swaps and window-based movements to search the solution space

1. Randomly Propose a Move

SA repeatedly proposes a move such as:

Swap two placed cells

Move a cell to a different slot

Window-based shift (move within a local region)

4. Decide Whether to Accept the Move

Case 1 — If $\Delta E < 0$

Always accept

Move makes the placement better

Case 2 — If $\Delta E \geq 0$ (worse move)

Use the Metropolis acceptance probability: $P=e^{-\Delta E/T}$

Cooling Schedule

Temperature T decreases gradually:

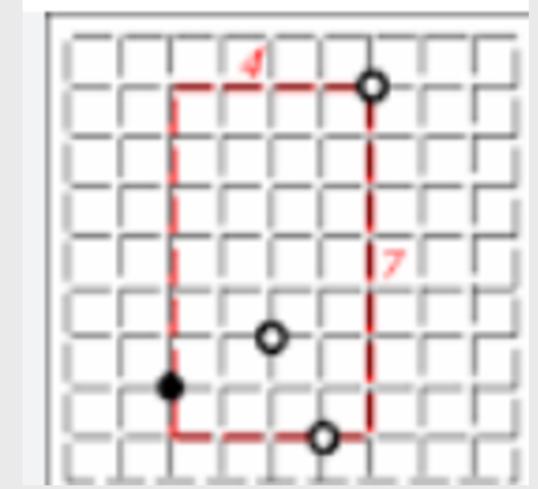
$$T_{\text{new}} = \alpha T_{\text{old}} + T_{\{\text{new}\}} = \alpha T_{\{\text{old}\}} + T_{\text{new}}$$

Where $\alpha \approx 0.90\text{--}0.99$.

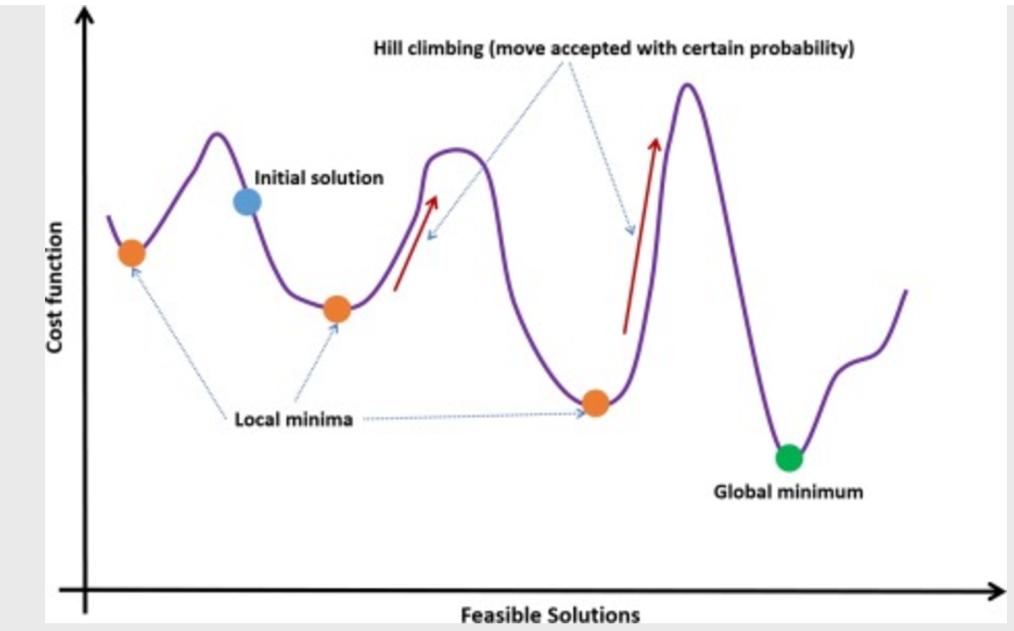
Effects:

High T → SA accepts many bad moves → global exploration

Low T → SA accepts almost no bad moves → fine refinement



$$\text{HPWL} = (\max_x - \min_x) + (\max_y - \min_y)$$



Parse design

Logical Database Construction

Groups all cell instances by their cell type

Helps the placer understand how many cells of each type must be allocated on the layout

```
{  
    'sky130_fd_sc_hd_nand2_2': ['cpu.U_alu.1', 'cpu.U_alu.2', ...],  
    'sky130_fd_sc_hd_clkinv_2': ['cpu.U_clock.1', ...],  
    ...  
}
```

Netlist Graph Construction

- A graph-like data structure for the netlist
- Each cell instance stores:
 - Its type
 - A dictionary of port → connected net IDs

```
'cpu.U_alu.1': {  
    'type': 'sky130_fd_sc_hd_nand2_2',  
    'connections': {  
        'A': [1], # Input connected to net 1  
        'B': [2], # Input connected to net 2  
        'Y': [3]  # Output connected to net 3  
    }  
,
```

Parse Fabric

What the File Does

Builds two databases needed for placement:

fabric_db → Physical slots grouped by Sky130 cell type

pins_db → Die/core dimensions + fixed I/O pin locations

Core Functions

1. parse_fabric_cells()

- Reads fabric_cells.yaml
- For each tile/cell, extracts:
 - slot name, x, y, orient
- Stores slots as:
 - cell_type → list of slot dictionaries

2. parse_pins()

- Reads pins.yaml
- Extracts:
 - die/core sizes
 - I/O pin locations (x, y, side, layer, fixed/placed)

Why It Matters

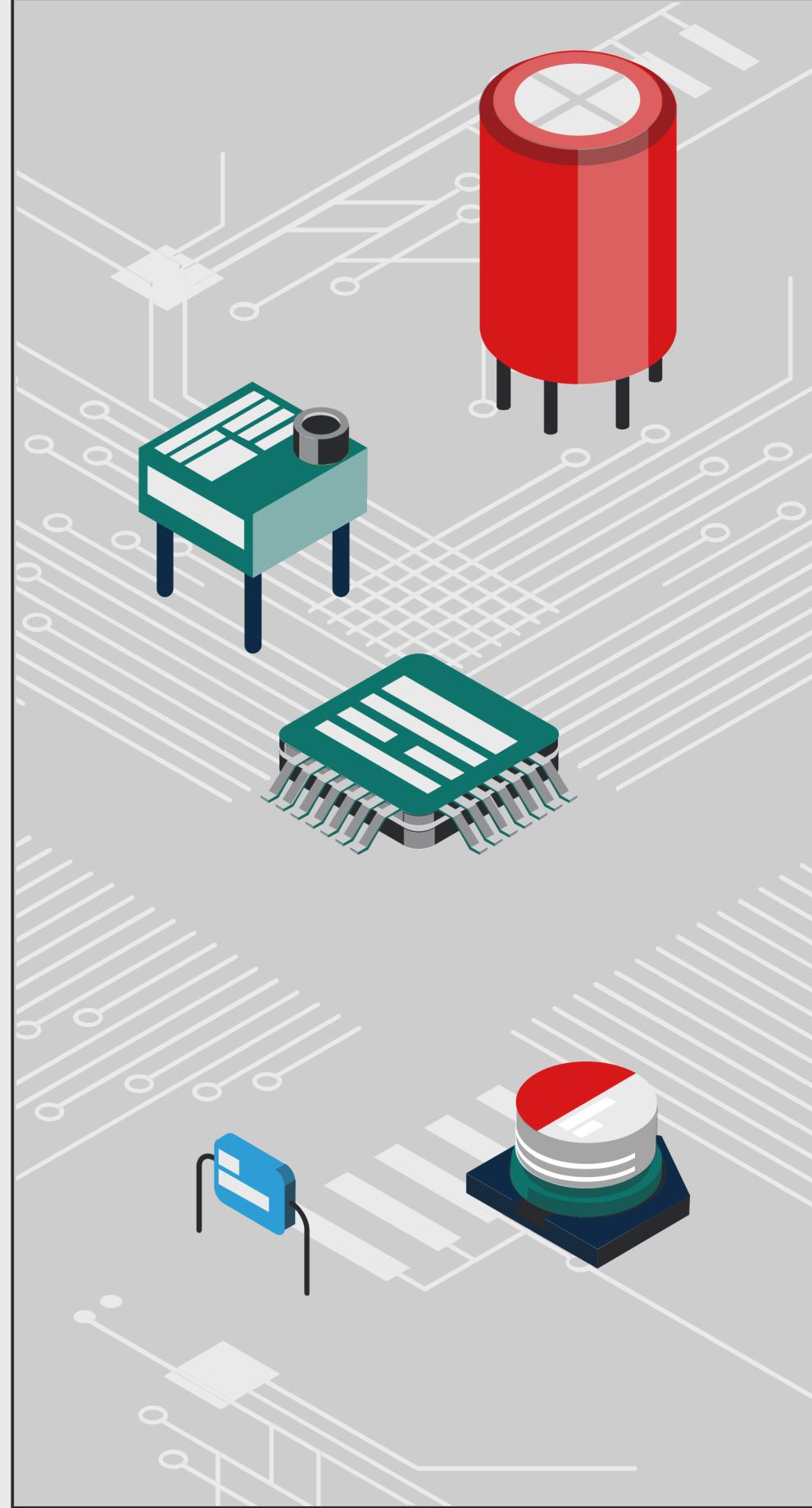
- Enables slot matching, distance/HPWL calculations, and I/O-driven seed placement
- Provides essential physical data required by the placer

Validator

validator.py ensures that a design fits on the structured ASIC fabric before running placement. It compares:

- Required cell counts (from the design netlist)
- Available slot counts (from the fabric description)

This pre-placement validation prevents running placement on designs that physically cannot fit.



Thank you for listening to us !

Any Questions ?