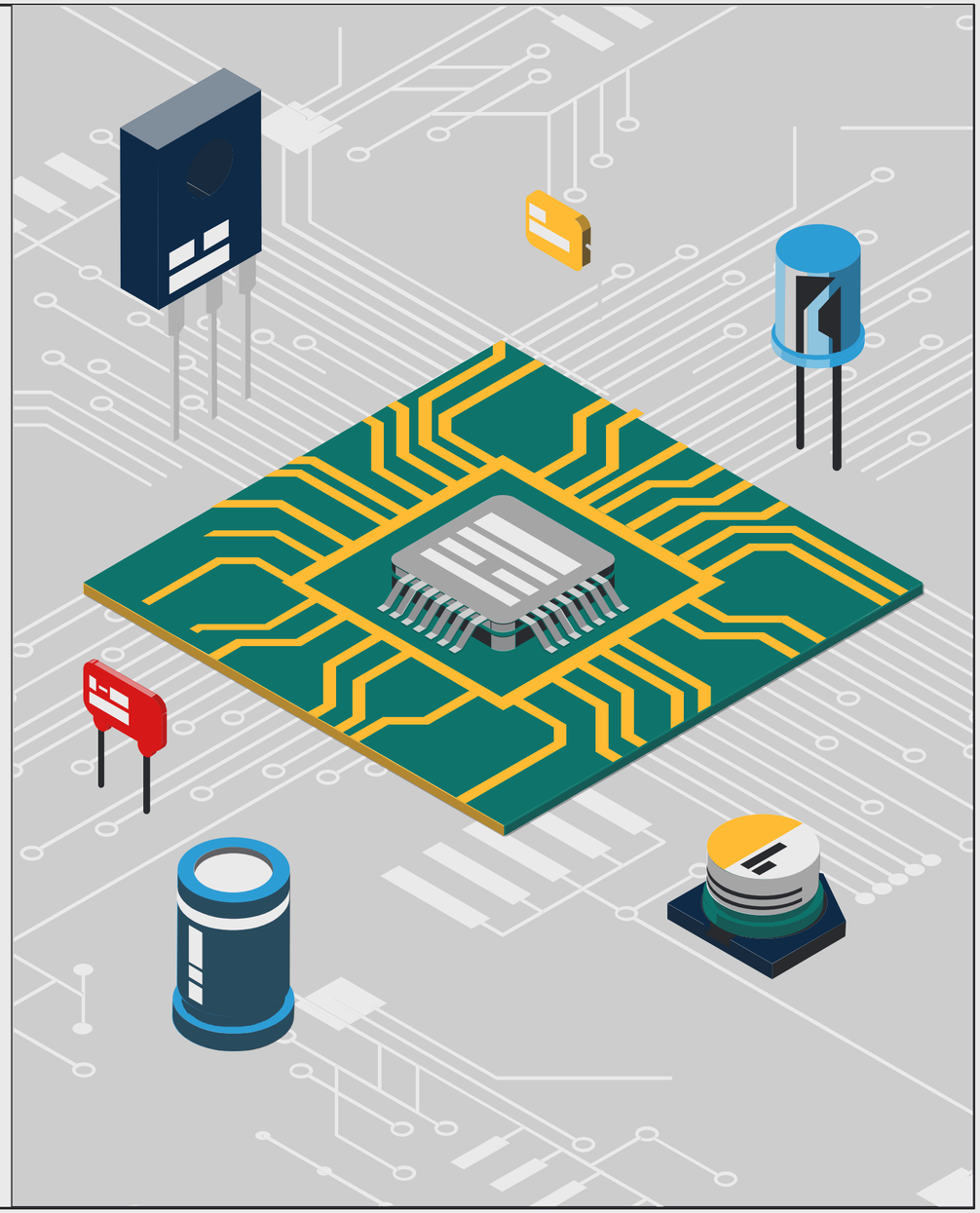
Structured ASIC Platform

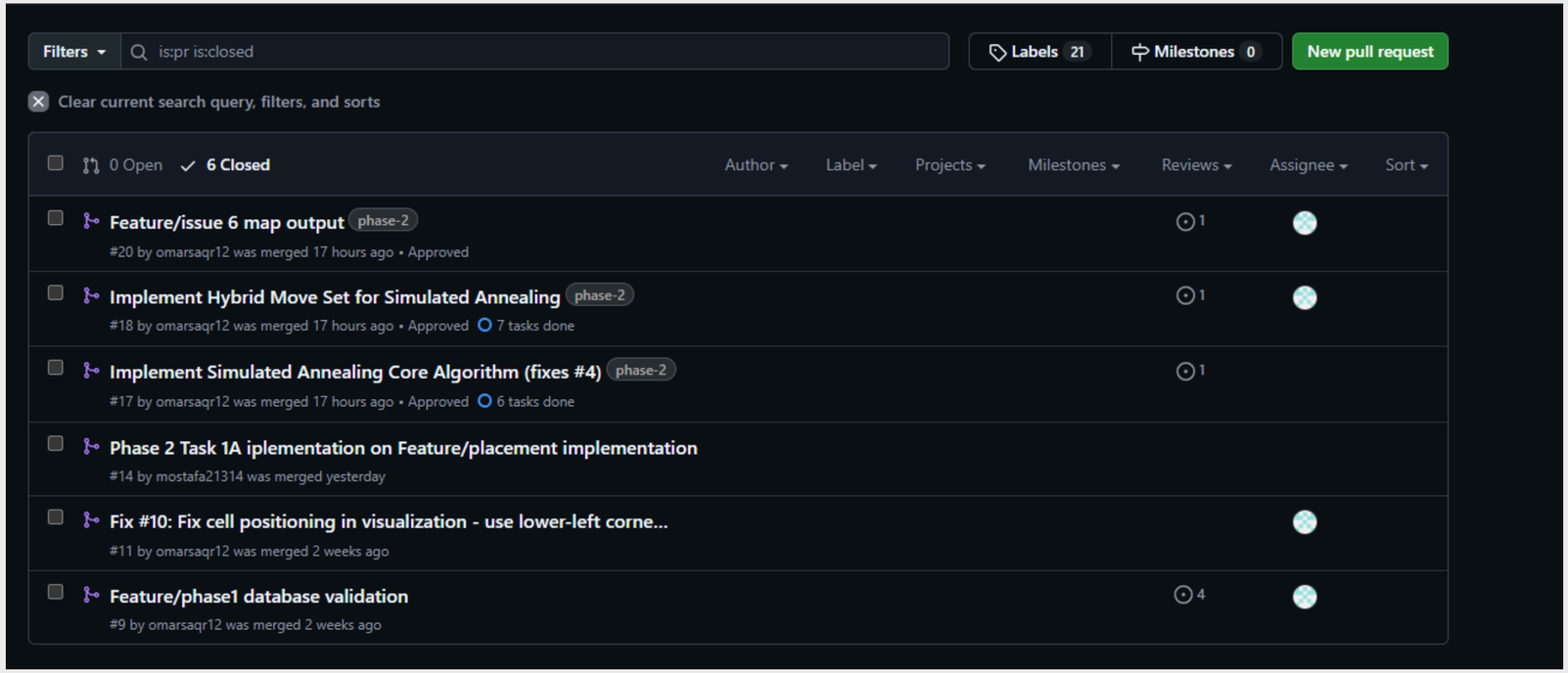
Checkpoint 2 Presentation

Mostafa Gaafar
Omar Saqr
Aabed Elghadban



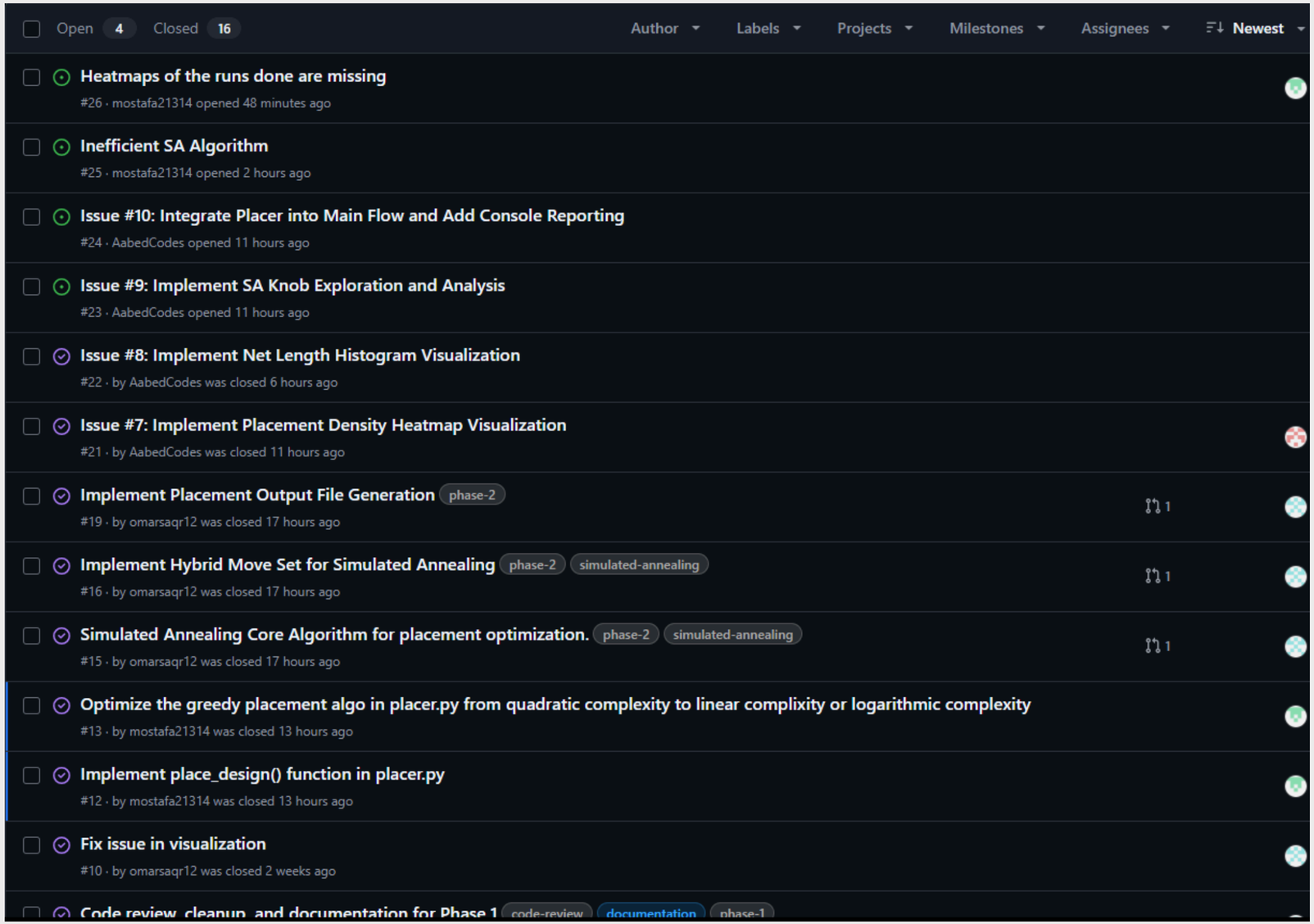
Project Status and GitHub Workflow

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



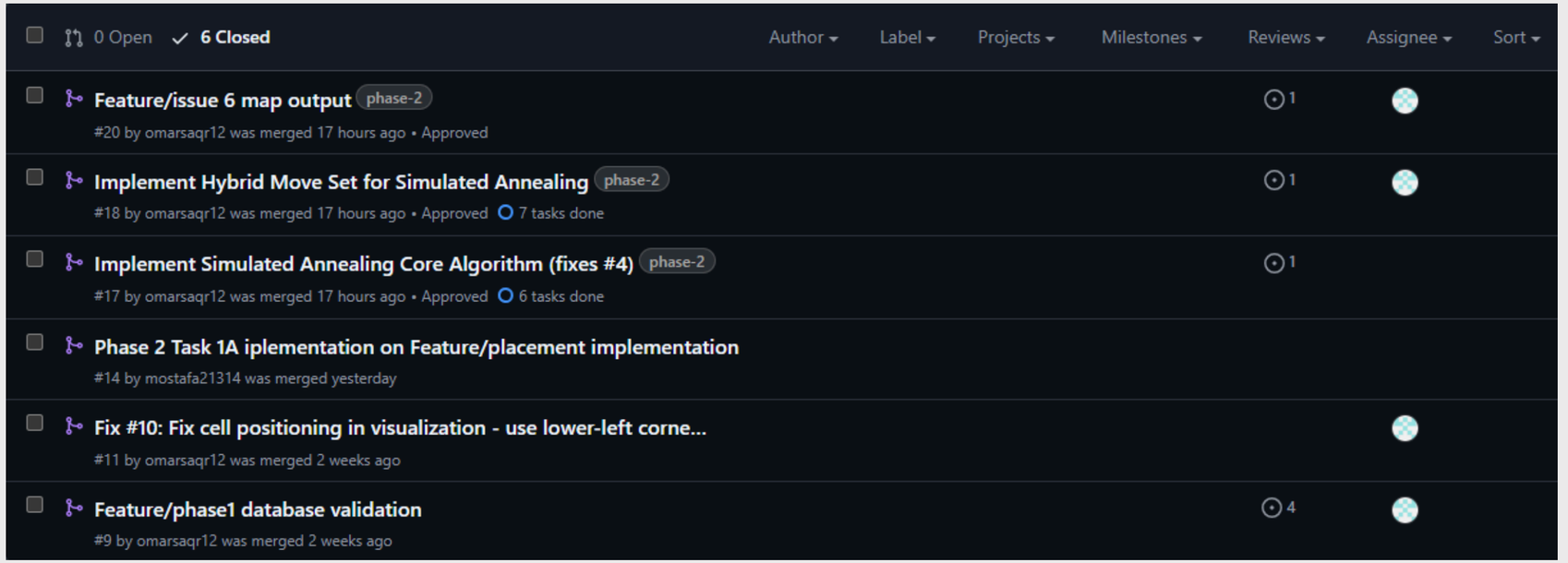
Project Status and GitHub Workflow

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

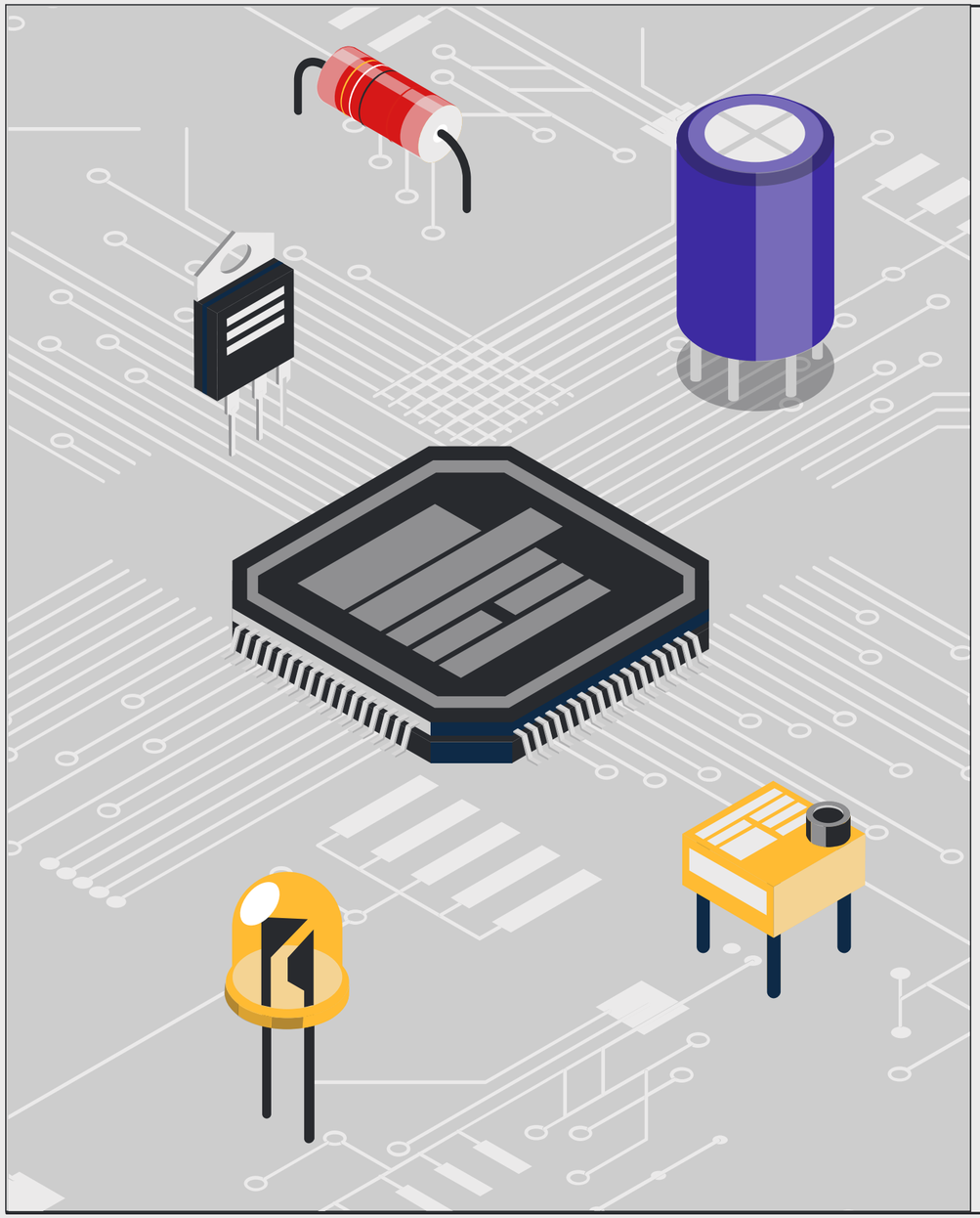


Project Status and GitHub Workflow

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



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



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%	✅ Pass
Arith	0.4%	0.5%	0.2%	0.5%	0.7%	✅ Pass
AES_128	95.2%	91.2%	77.4%	49.0%	46.5%	✅ Pass
Z80	6.9%	17.5%	2.5%	7.8%	0.8%	✅ 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: arith

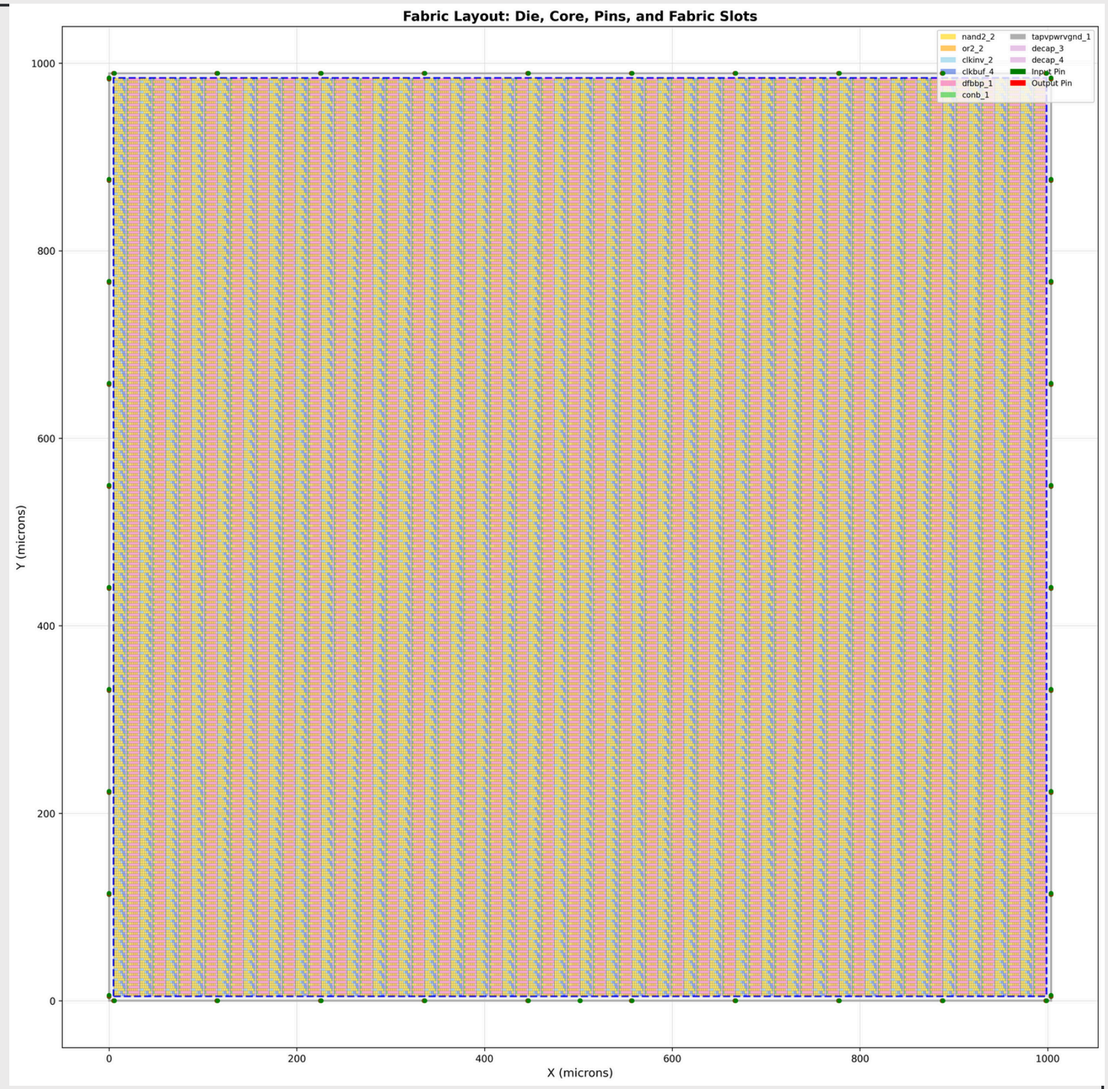
Cell Type	Required	Available	Utilization	Status
BUF (clkbuf_4)	67	9720	**0.7%**	✅ OK
INV (clkinv_2)	68	12960	0.5%	✅ OK
CONB (conb_1)	3	9720	0.0%	✅ OK
DECAP3 (decap_3)	0	9720	0.0%	✅ OK
DECAP4 (decap_4)	0	9720	0.0%	✅ OK
DFBBP (dfbbp_1)	11	6480	0.2%	✅ OK
NAND2 (nand2_2)	187	48600	0.4%	✅ OK
OR2 (or2_2)	127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

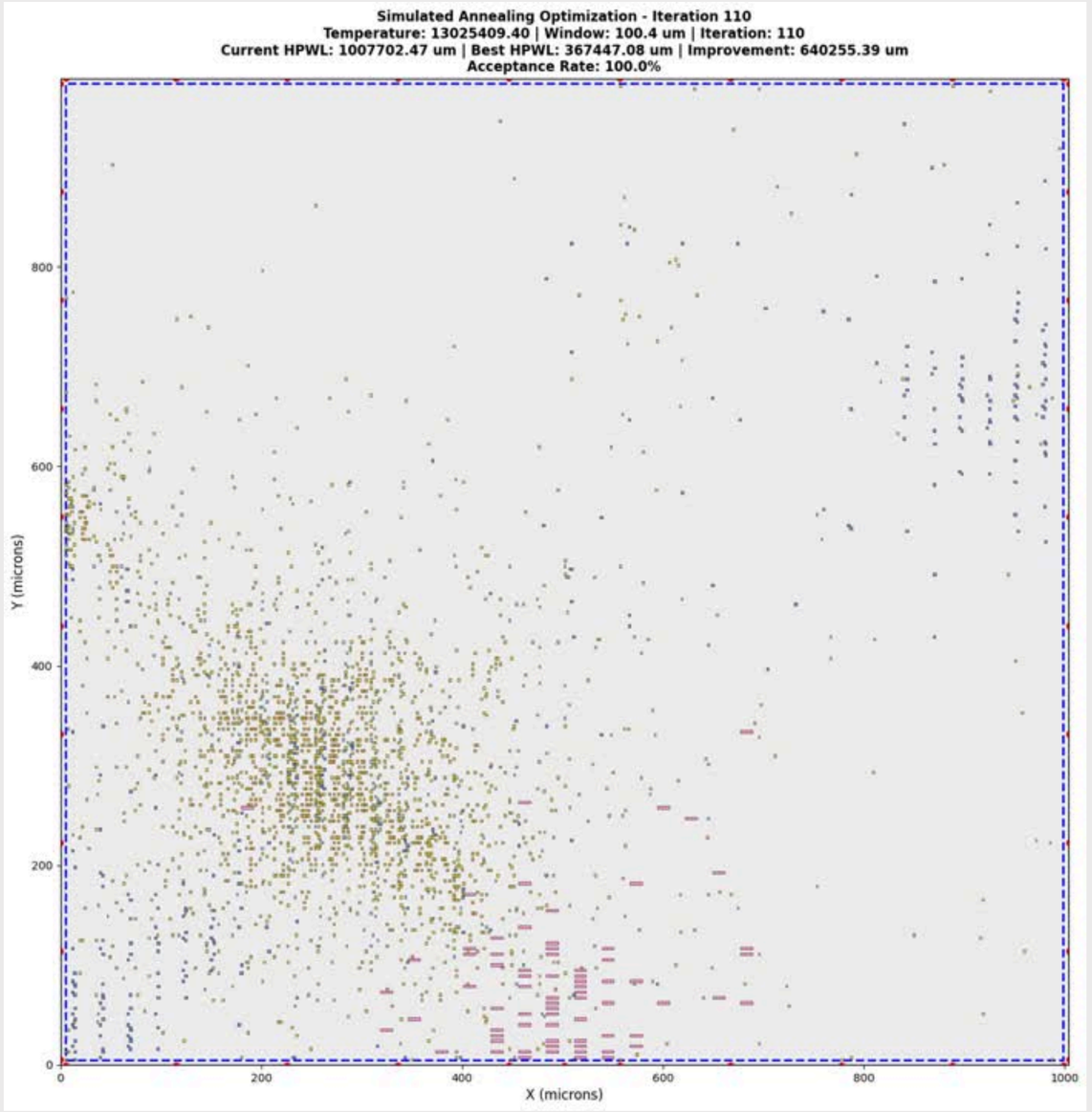
VERDICT: ✅ PASSED

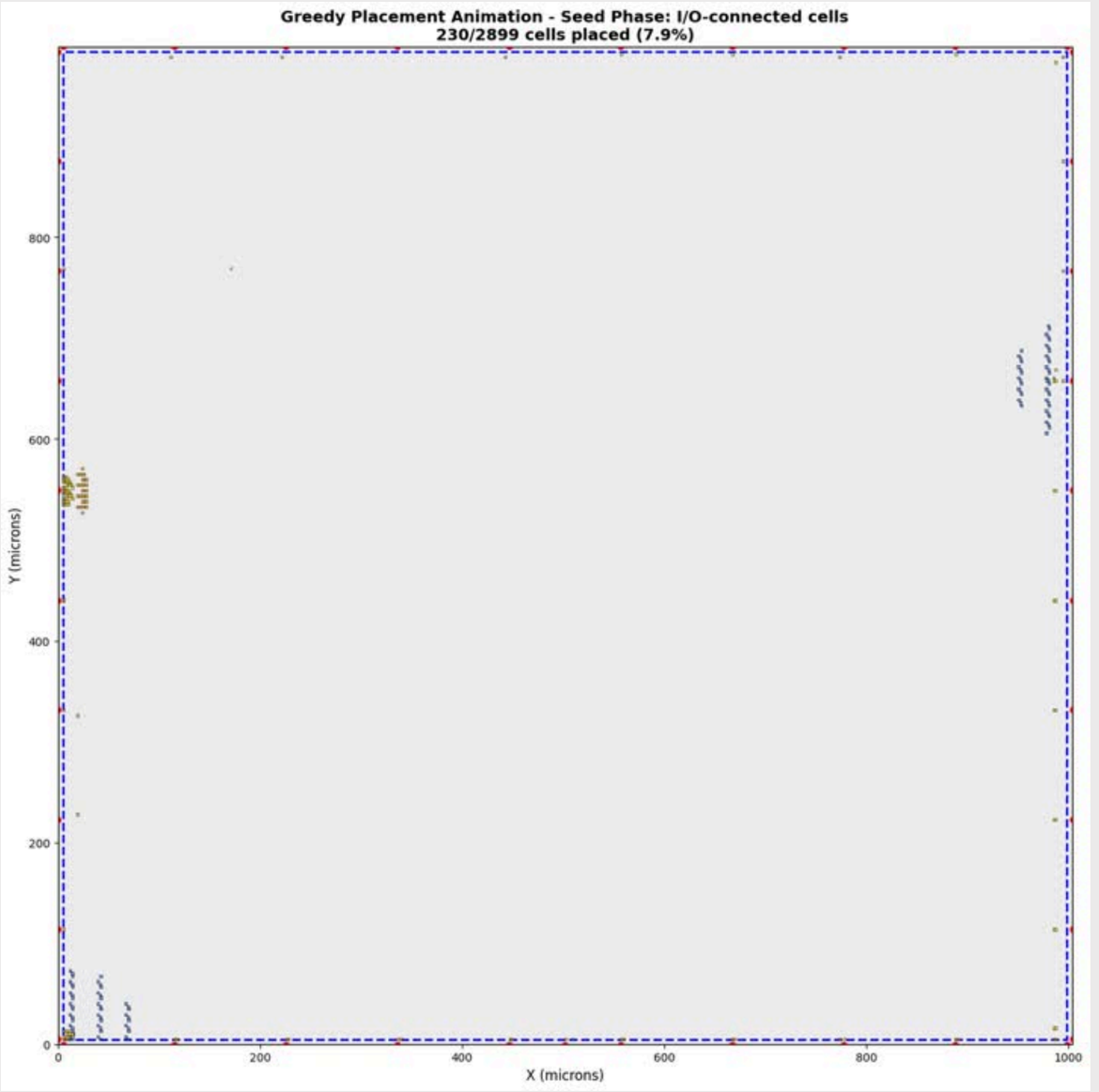
Design: 6502

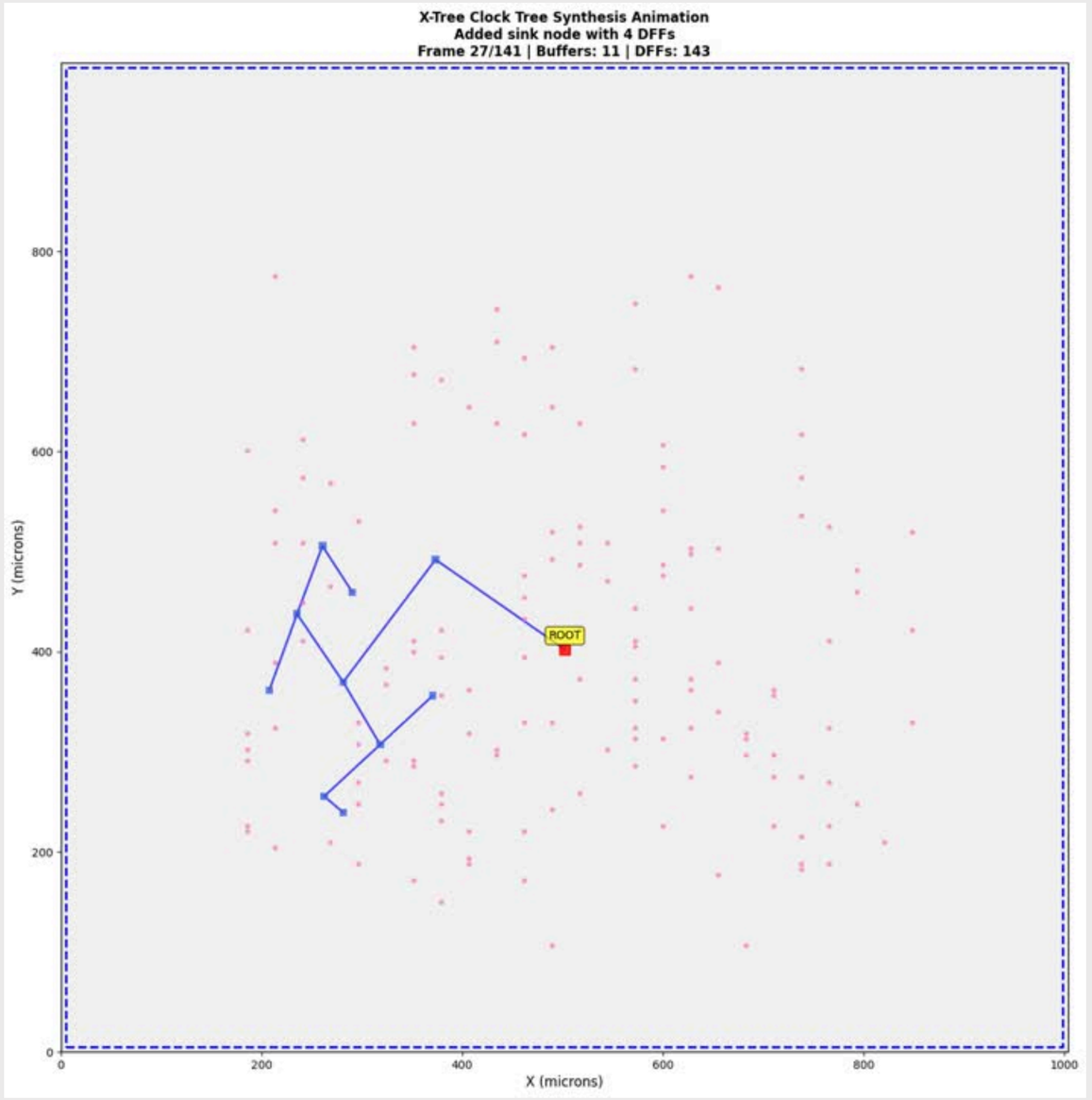
Cell Type	Required	Available	Utilization	Status
BUF (clkbuf_4)	322	9720	**3.3%**	✅ OK
INV (clkinv_2)	360	12960	2.8%	✅ OK
CONB (conb_1)	3	9720	0.0%	✅ OK
DECAP3 (decap_3)	0	9720	0.0%	✅ OK
DECAP4 (decap_4)	0	9720	0.0%	✅ OK
DFBBP (dfbbp_1)	143	6480	2.2%	✅ OK
NAND2 (nand2_2)	1228	48600	2.5%	✅ OK
OR2 (or2_2)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

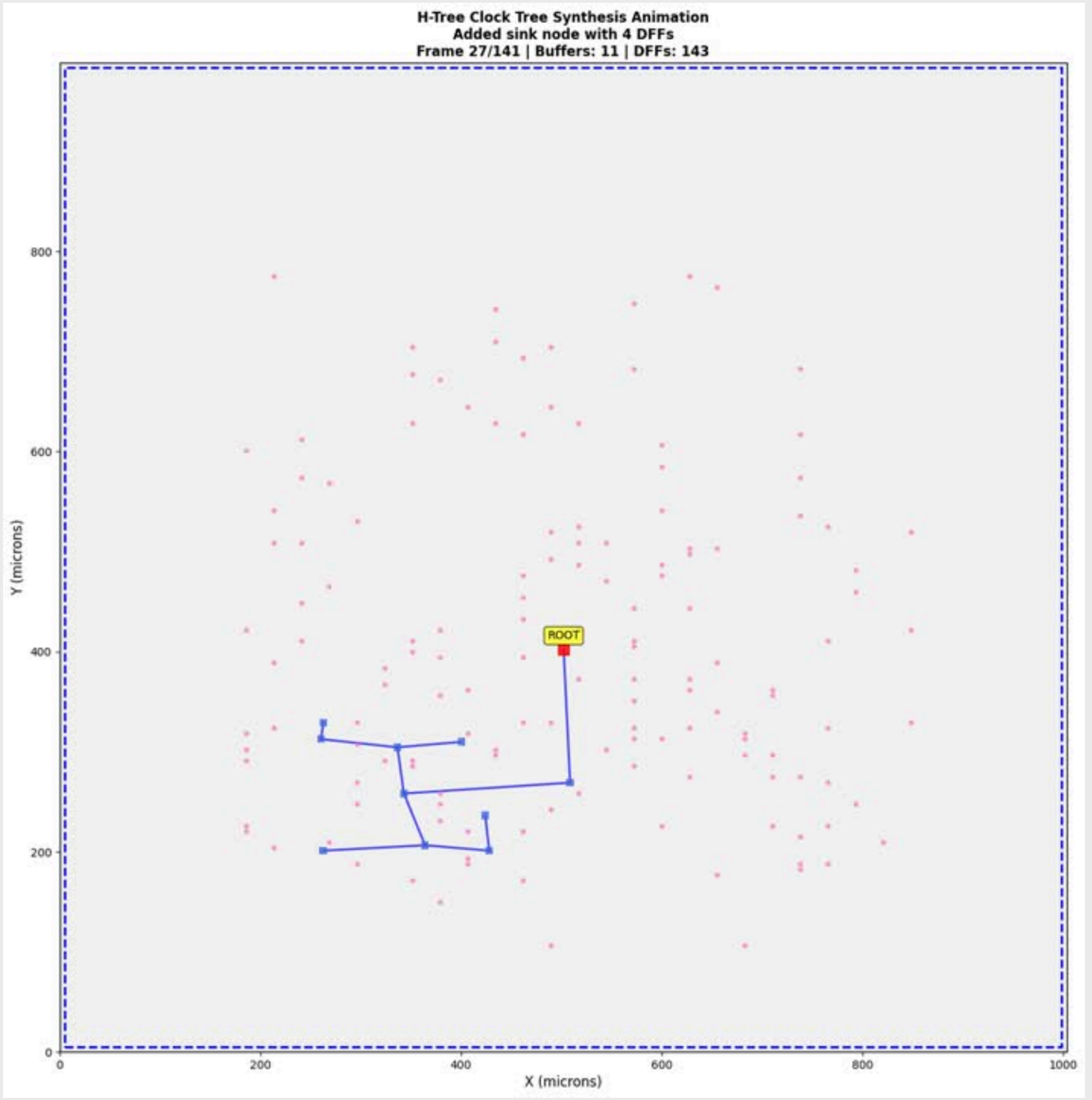
VERDICT: ✅ PASSED


Fabric Layout: Die, Core, Pins, and Fabric Slots
nand2_2
or2_2
clkinv_2
clkbuf_4
dfbbp_1
conb_1
tapvpwrvgnd_1
decap_3
decap_4
Input Pin
Output Pin
Y (microns)
X (microns)
0
200
400
600
800
1000

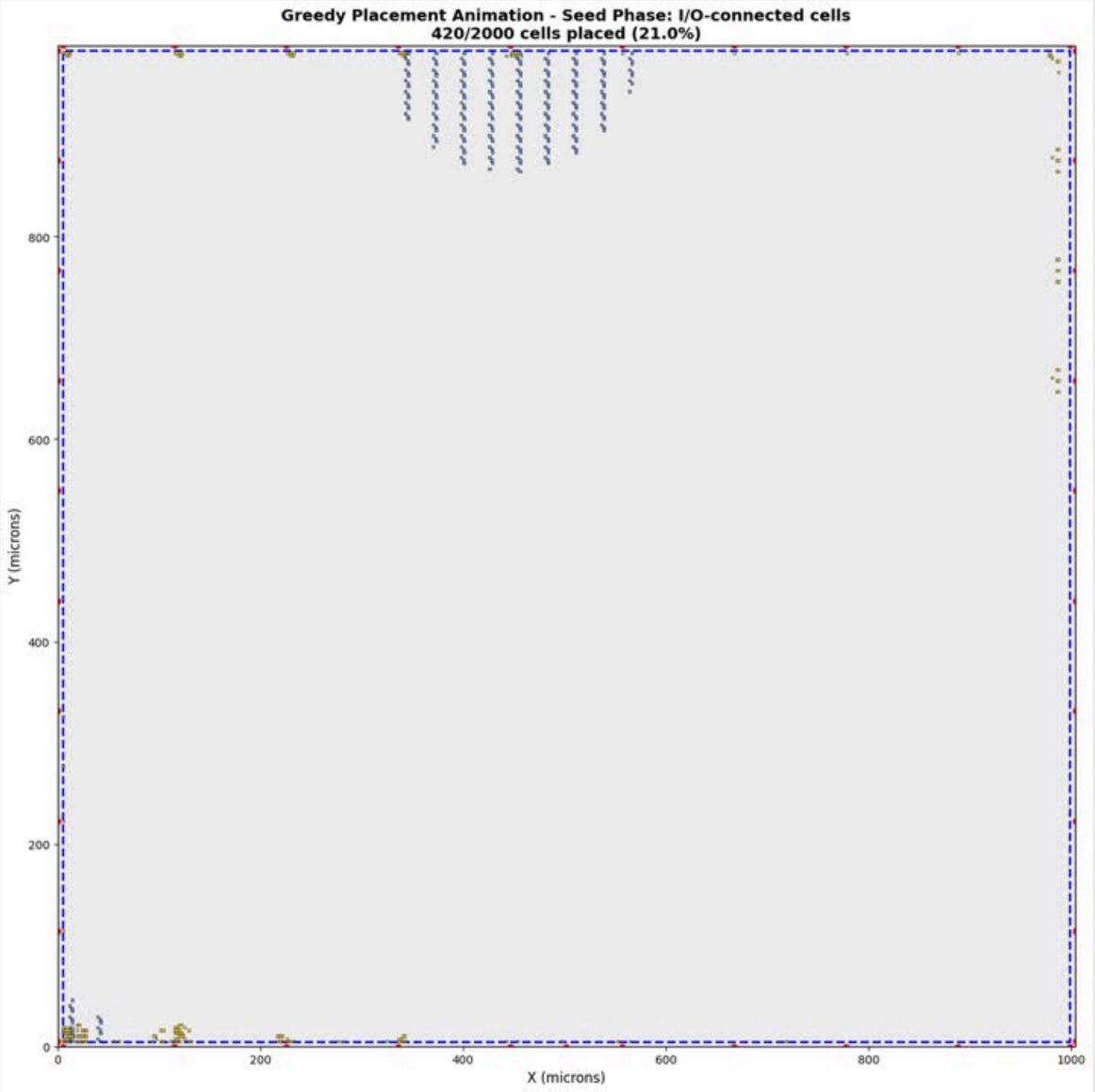

Simulated Annealing Optimization - Iteration 110
Temperature: 13025409.40 | Window: 100.4 um | Iteration: 110
Current HPWL: 1007702.47 um | Best HPWL: 367447.08 um | Improvement: 640255.39 um
Acceptance Rate: 100.0%
Y (microns)
X (microns)
0
200
400
600
800
1000


Greedy Placement Animation - Seed Phase: I/O-connected cells
230/2899 cells placed (7.9%)
Y (microns)
X (microns)
0
200
400
600
800
1000

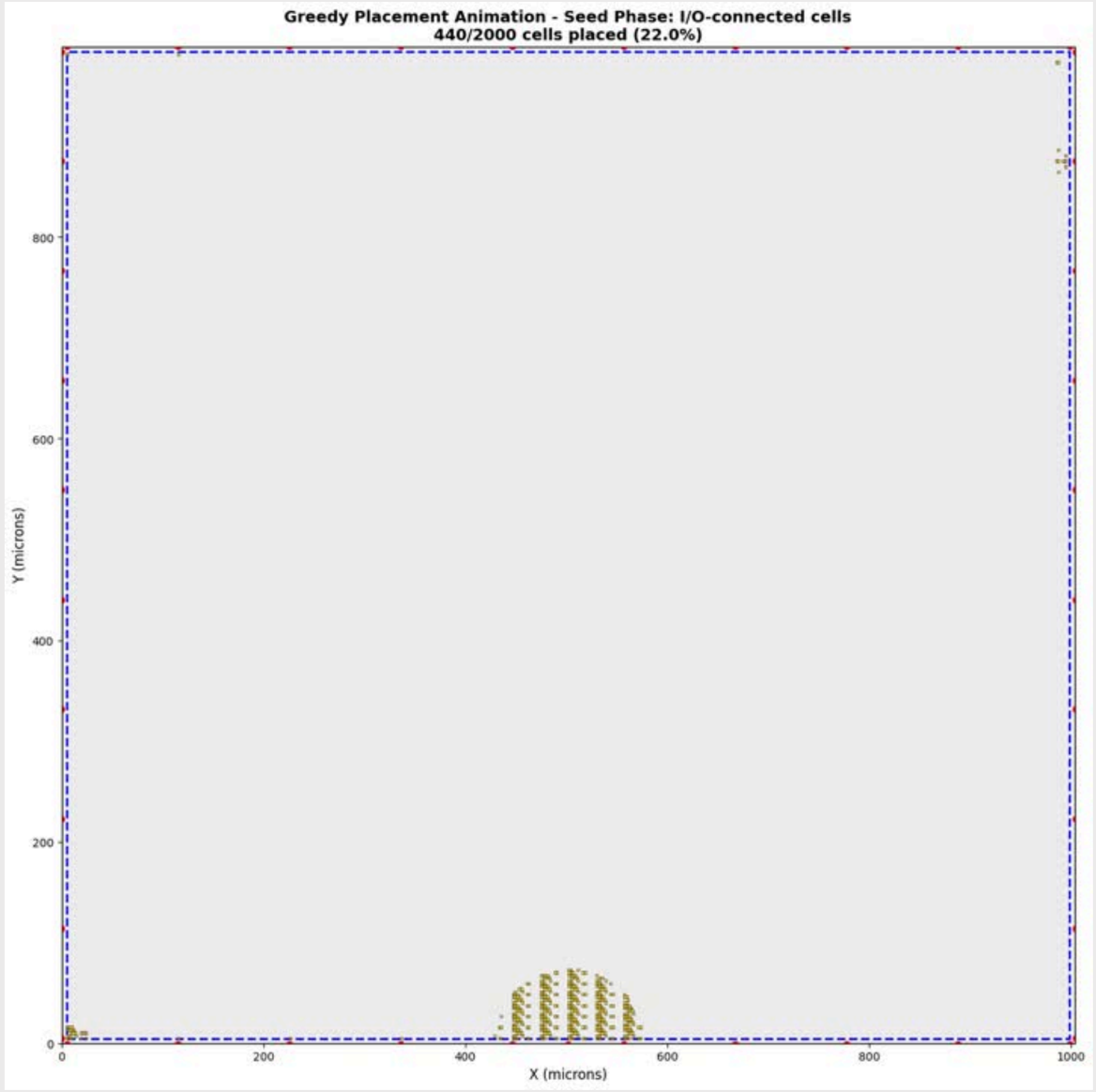

X-Tree Clock Tree Synthesis Animation
Added sink node with 4 DFFs
Frame 27/141 | Buffers: 11 | DFFs: 143
ROOT
Y (microns)
X (microns)
0
200
400
600
800
1000


H-Tree Clock Tree Synthesis Animation
Added sink node with 4 DFFs
Frame 27/141 | Buffers: 11 | DFFs: 143
ROOT
Y (microns)
X (microns)
0
200
400
600
800
1000

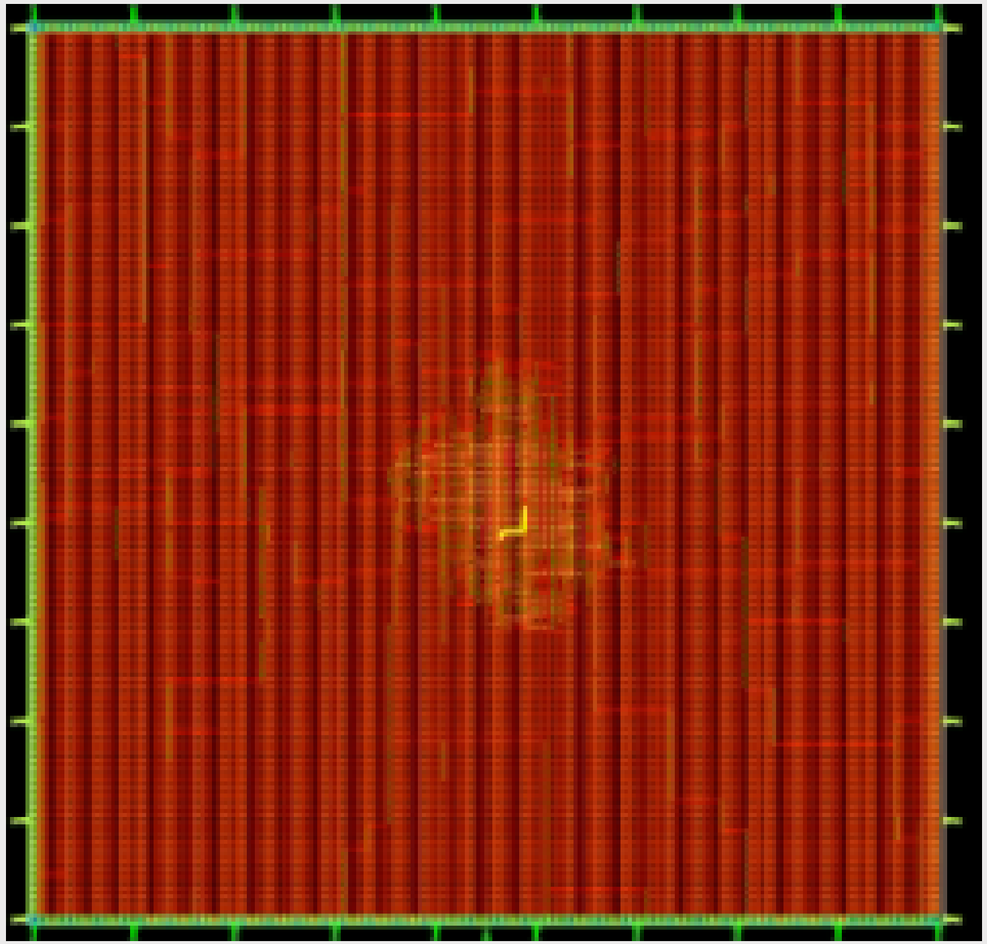
Aes_128


Greedy Placement Animation - Seed Phase: I/O-connected cells
420/2000 cells placed (21.0%)
Y (microns)
X (microns)
0
200
400
600
800
1000

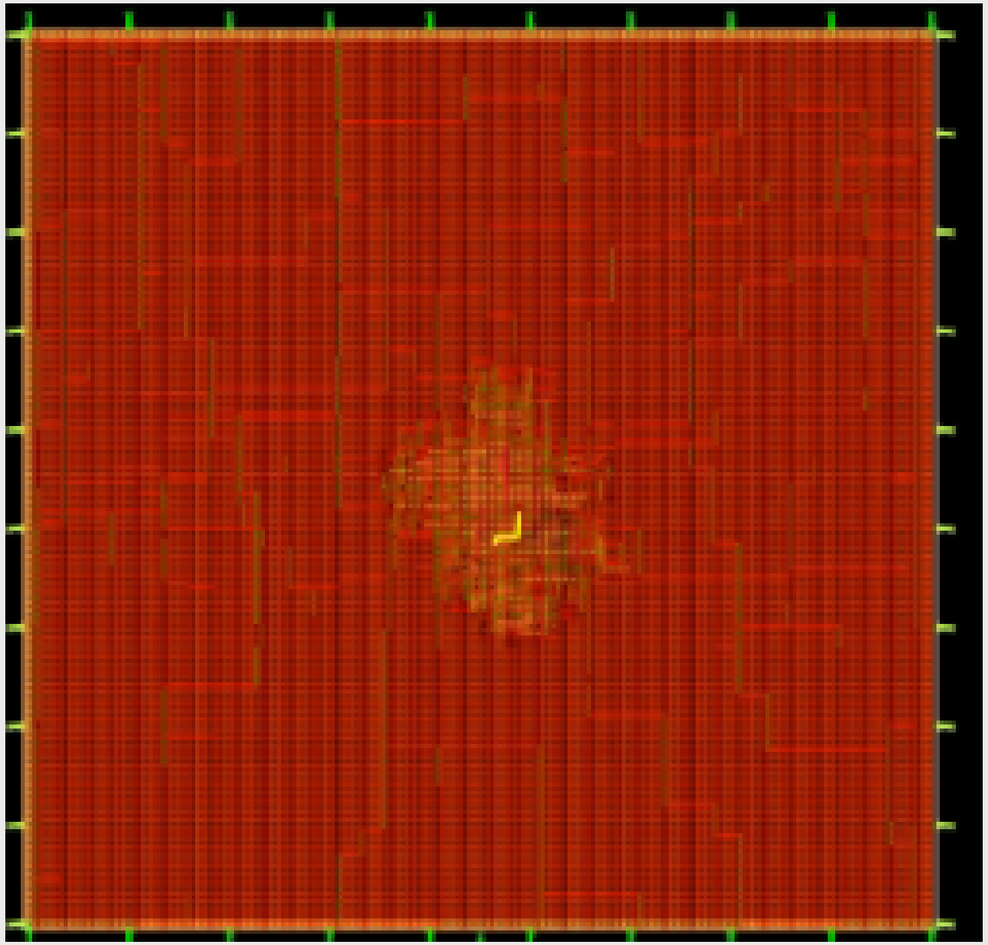
SOC


Greedy Placement Animation - Seed Phase: I/O-connected cells
440/2000 cells placed (22.0%)
Y (microns)
X (microns)
0
200
400
600
800
1000

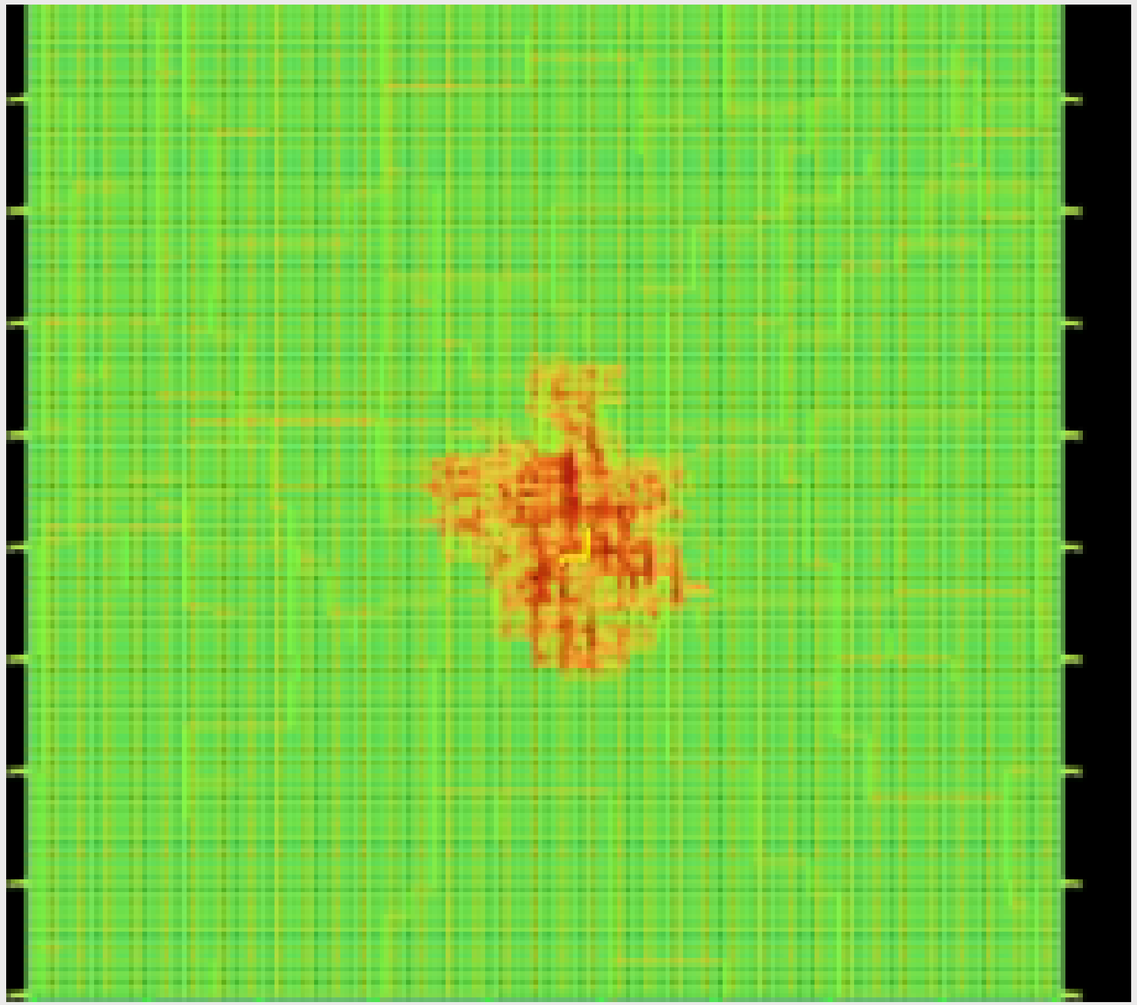
6502-Placement Density



6502-Power Density

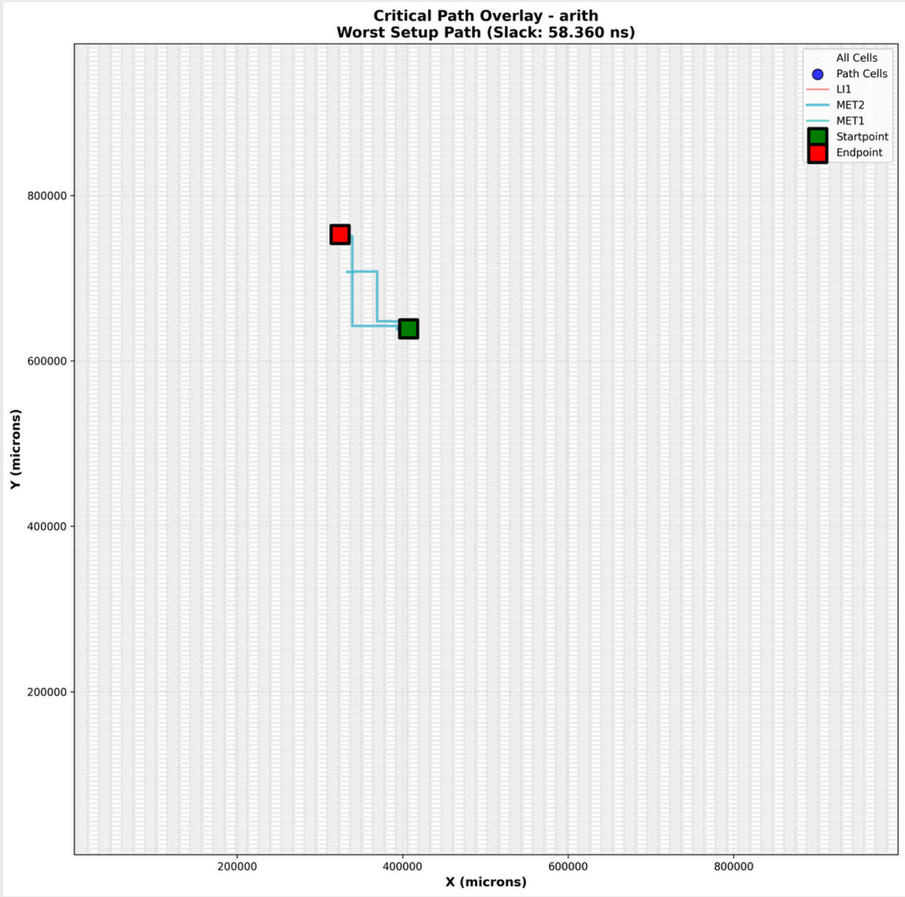


6502-Routing Congestion

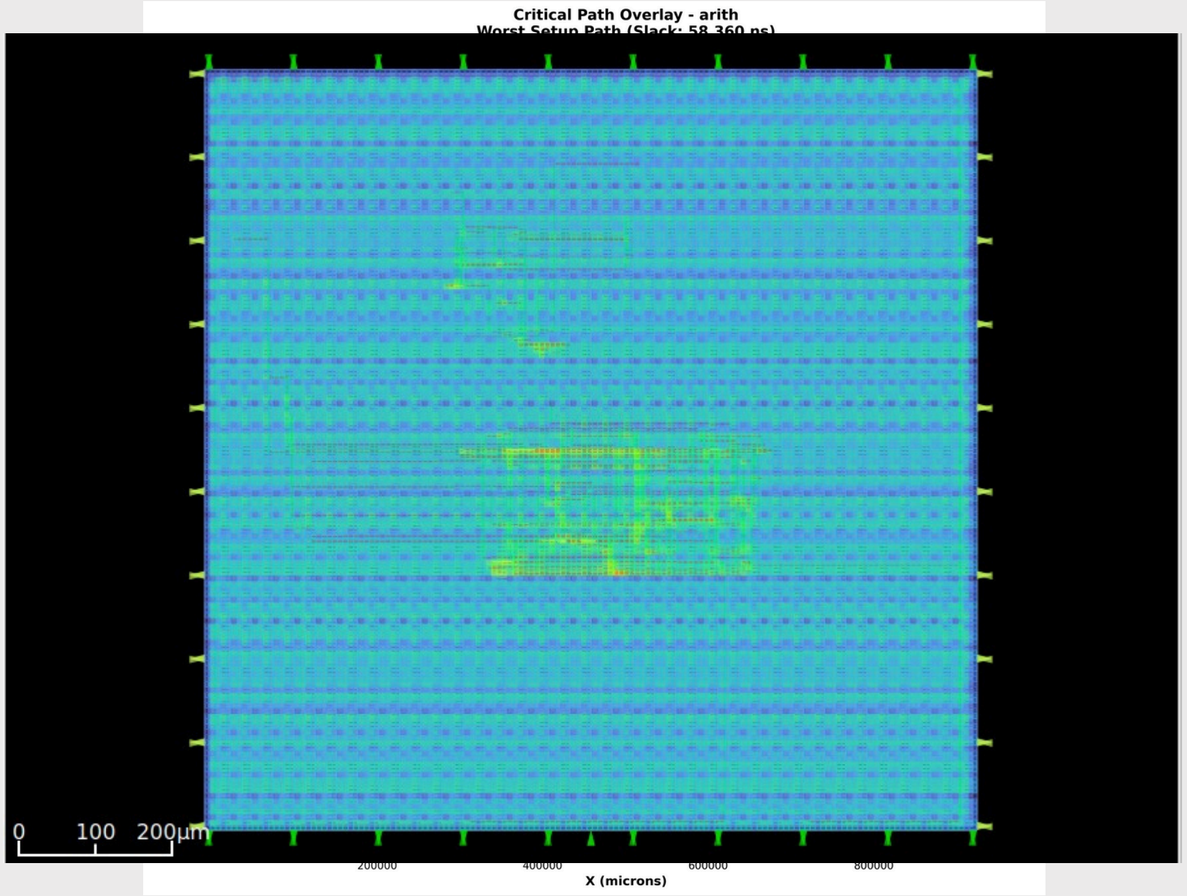


6502- Critical Path

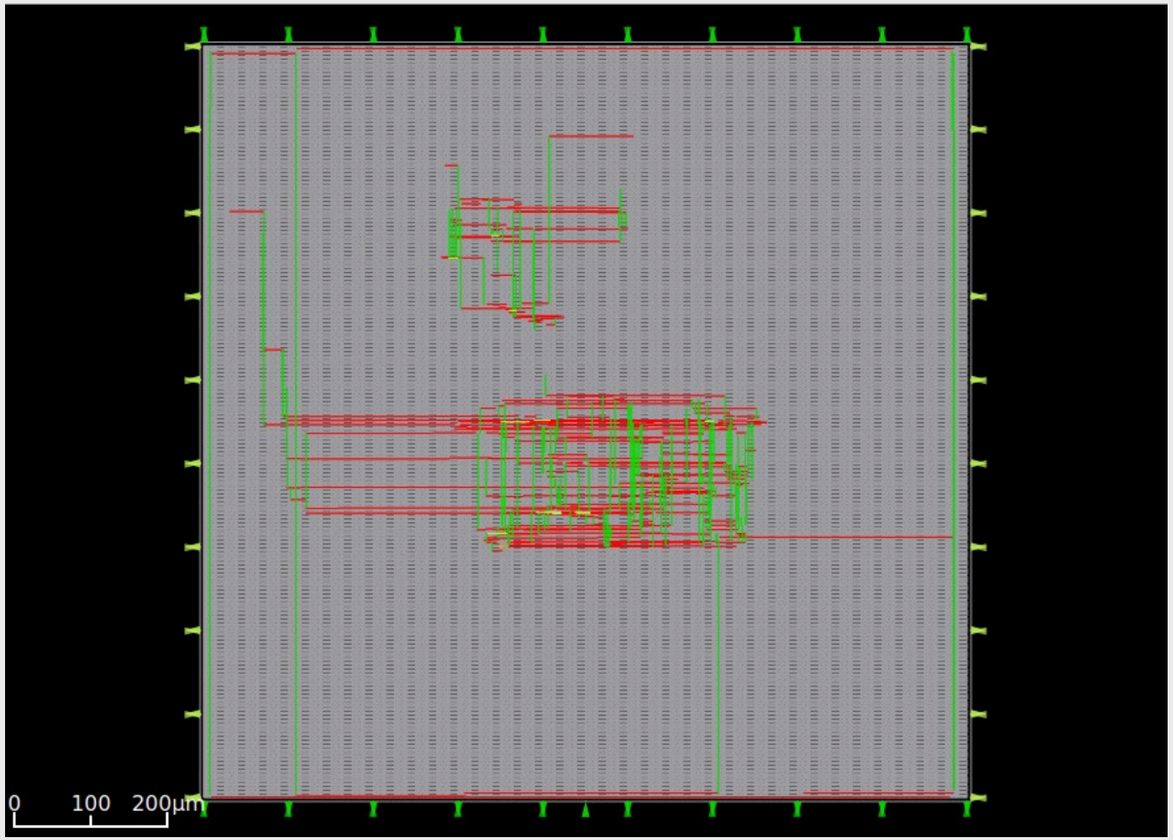
Arith- Critical Path



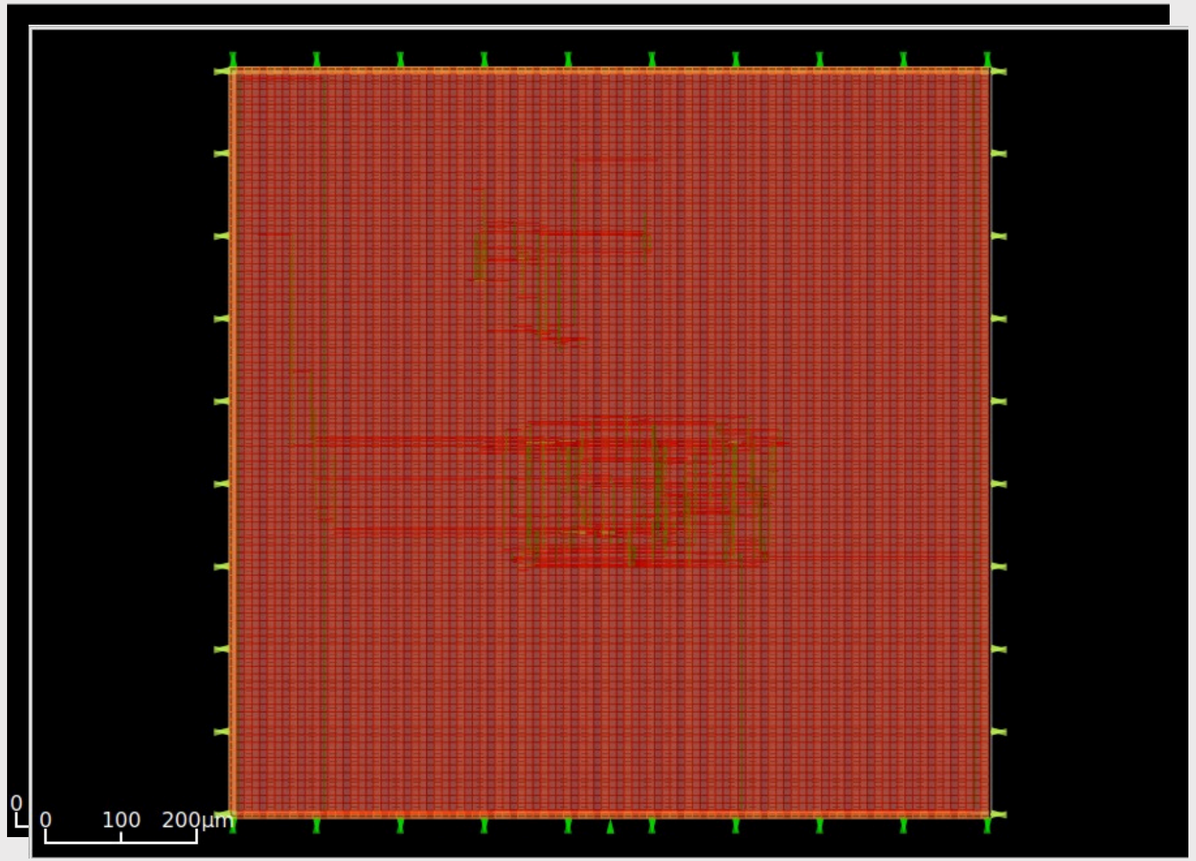
Arith


Critical Path Overlay - arith
0
100
200µm
200000
400000
600000
800000
X (microns)

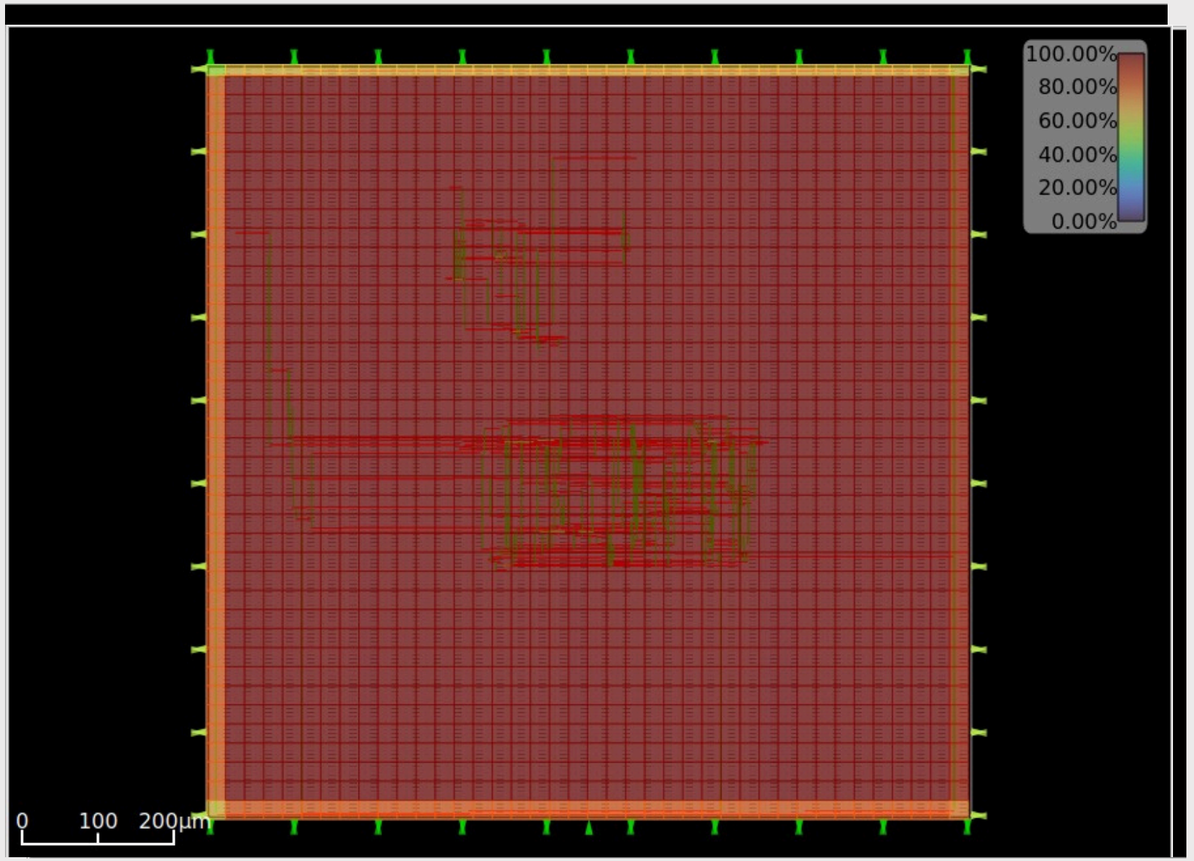
Arith



Arith



Arith



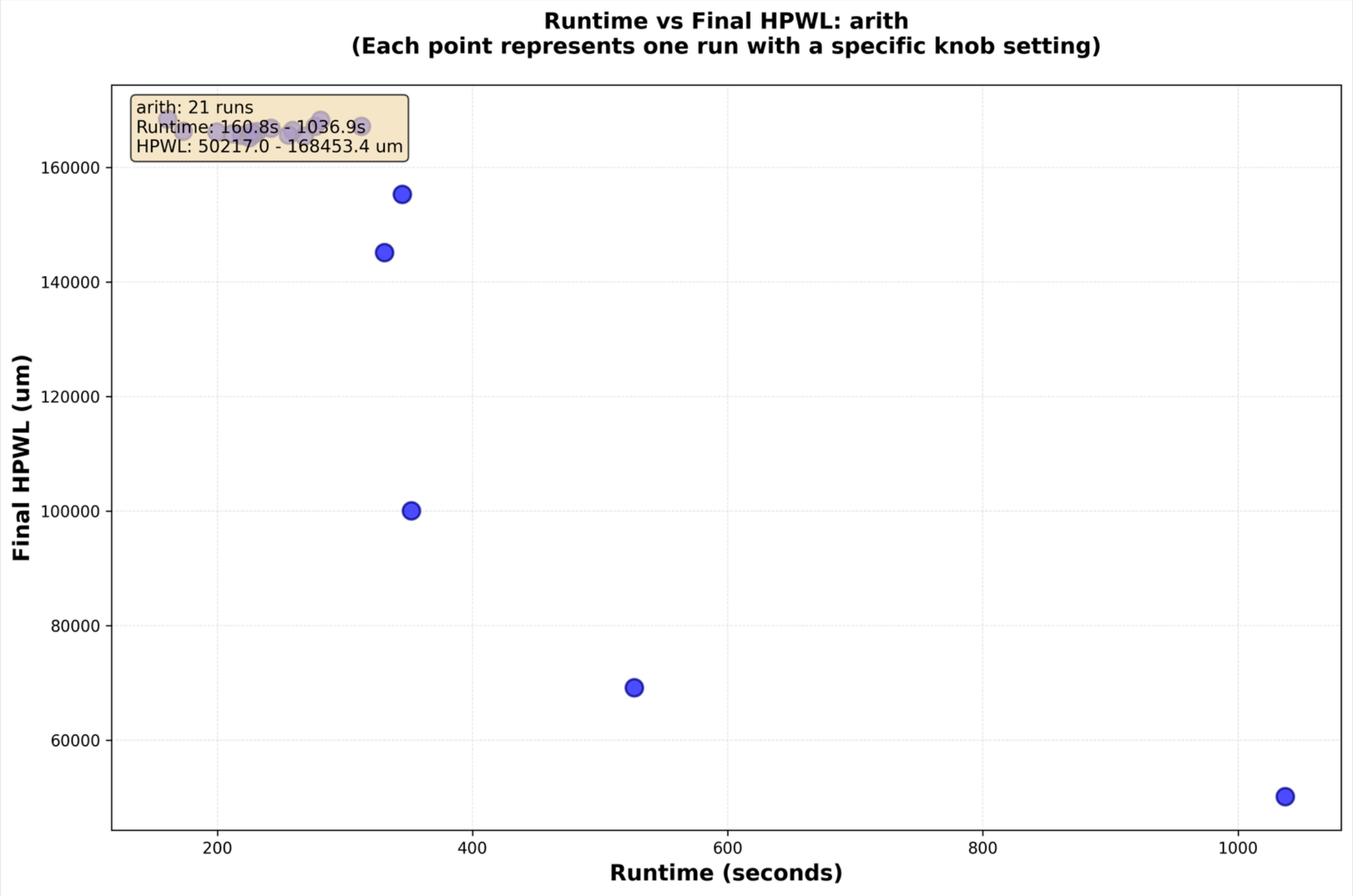
HPWL

Placement Results

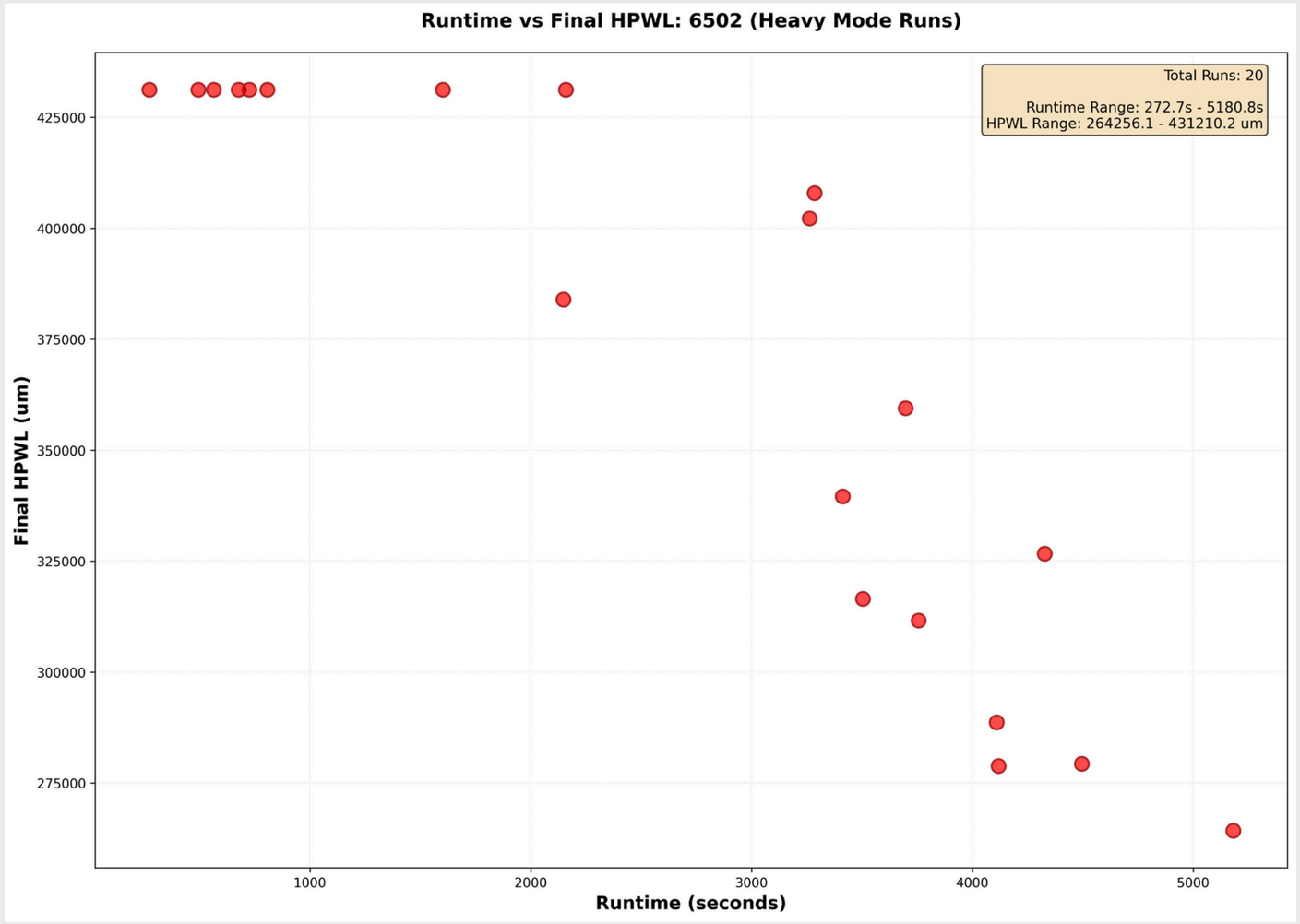
Desktop (plugged to power)
CPU: Intel Core i9-13900F (13th Gen) x 32 cores
GPU: NVIDIA RTX 4080
RAM: 64 GB
Storage: 1 TB
OS: Linux

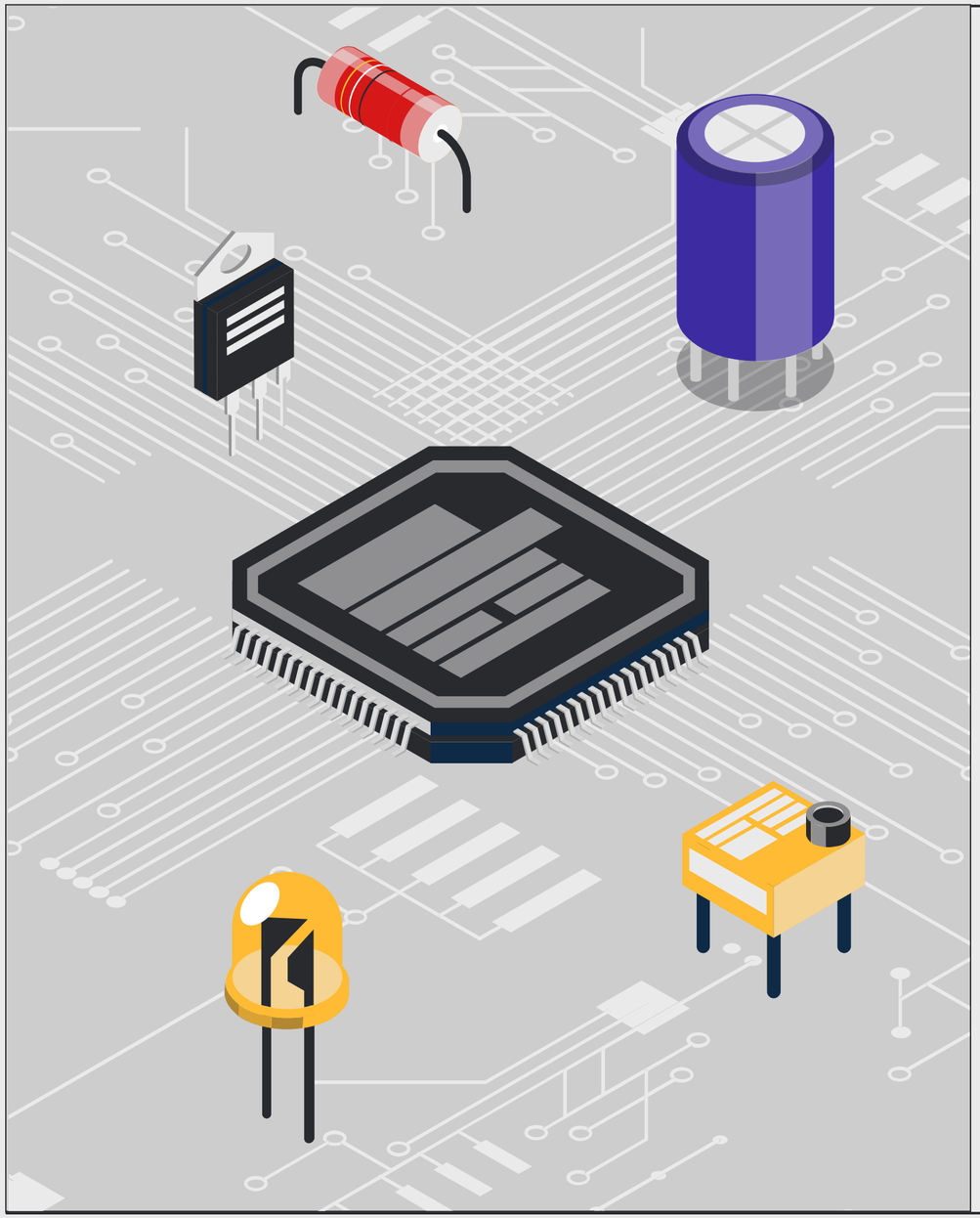
Design	Final HPWL (um)	Runtime (seconds)	Improvement (%)	Notes
6502	101,346.87	107,392.3	75.00%	Runtime from logs
arith	50,216.99	1,329.0	69.59%	Standalone run
z80	1,050,999.43	21,934.0	27.04%	Ran as a single worker
aes	17,103,179.44	216,059.0	0.00%	SA stalled, no HPWL improvement
SoC	N/A	5,400.0	N/A	Greedy only, NA HPWL

SA effect of Arith



SA effect on 6502





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 × 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 × O(N) neighbor lookups = O(N²)
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(N³)

# Optimized: Only rescore cells on same nets as placed cell
place(best_cell)
affected_cells = get_cells_on_same_nets(best_cell)  # O(M × 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 × d) where k ≈ 10, d ≈ 10
# Total: O(N × k × d)
```

Only update cells affected by the just-placed cell

Greedy Key implementation details

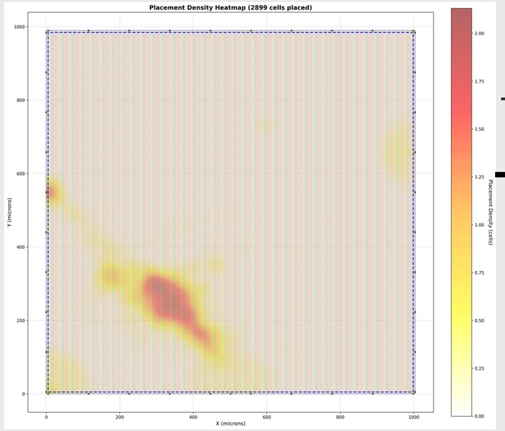
Optimizations done

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

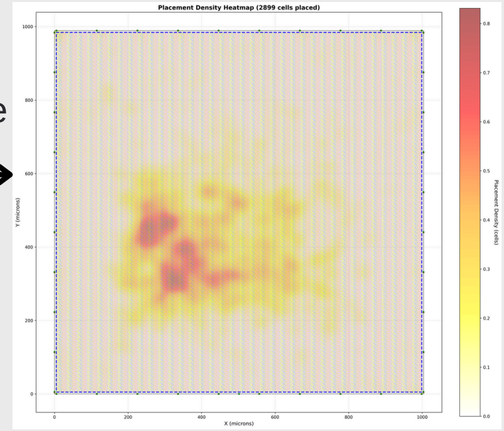
Single pass with pre-computed mappings

```
# 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)
```

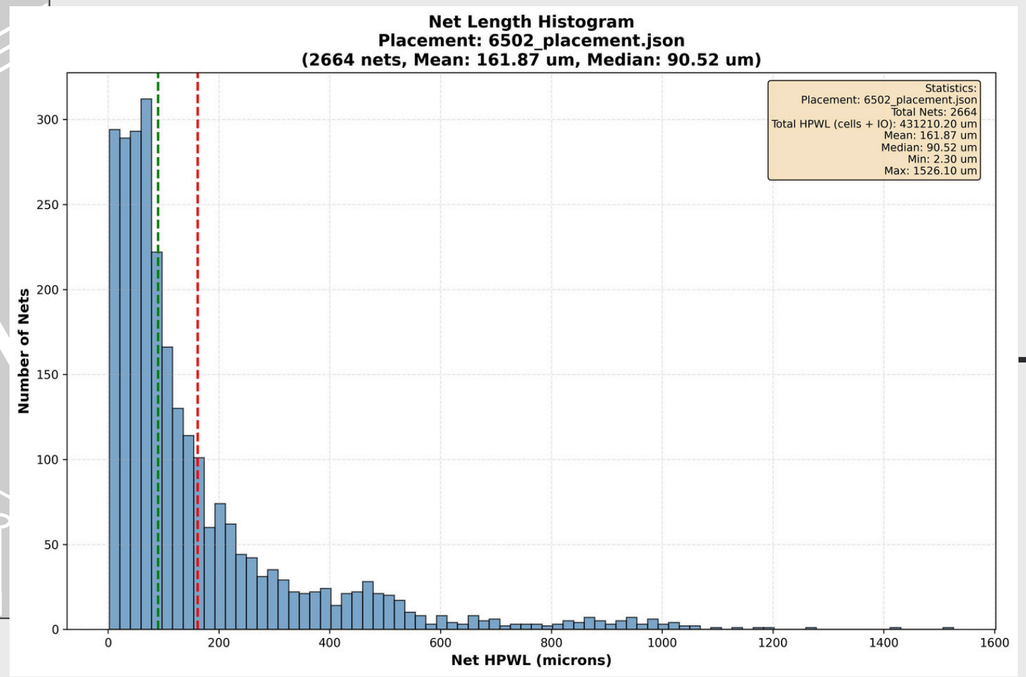
SA effect on 6502



T_initial = 10000 × initial_HPWL
This increased running time unnecssarly, and didn't provide many gains, so we will adjust 10000 to 10s range.

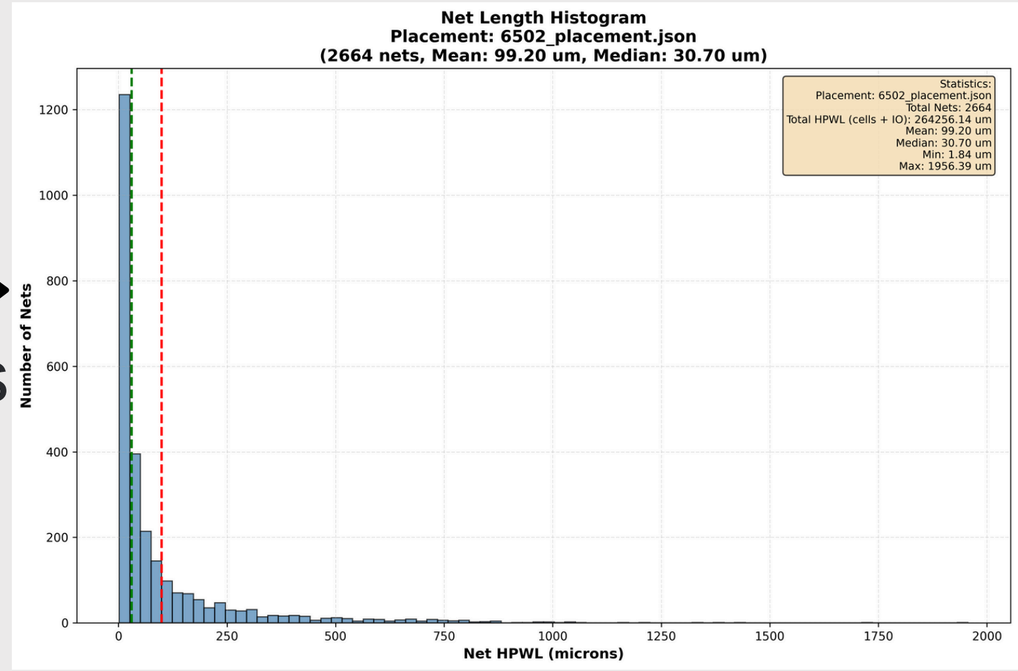


Reduction = 431,210.20 μm - 264,256.14 μm
= 166,954.06μm
Percentage reduction = 38.72% reduction



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.

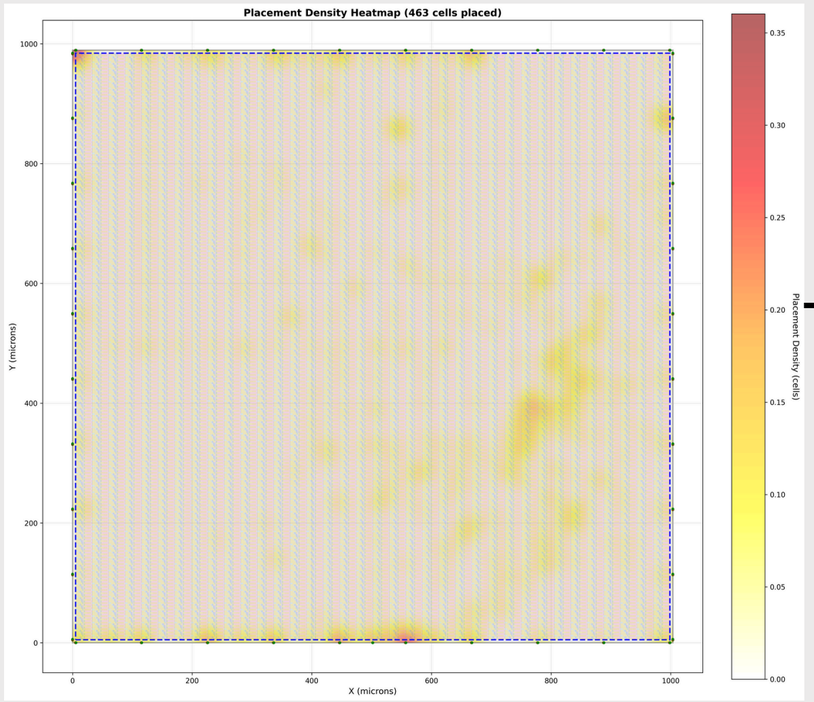
SA:
SA breaks the greedy pattern by:
Random exploration
Cooling-based acceptance of uphill (worse) moves early on
As a result:
Clusters loosen. In this instance loosening clusters reduced total HPWL because many nets were global.

Histogram:

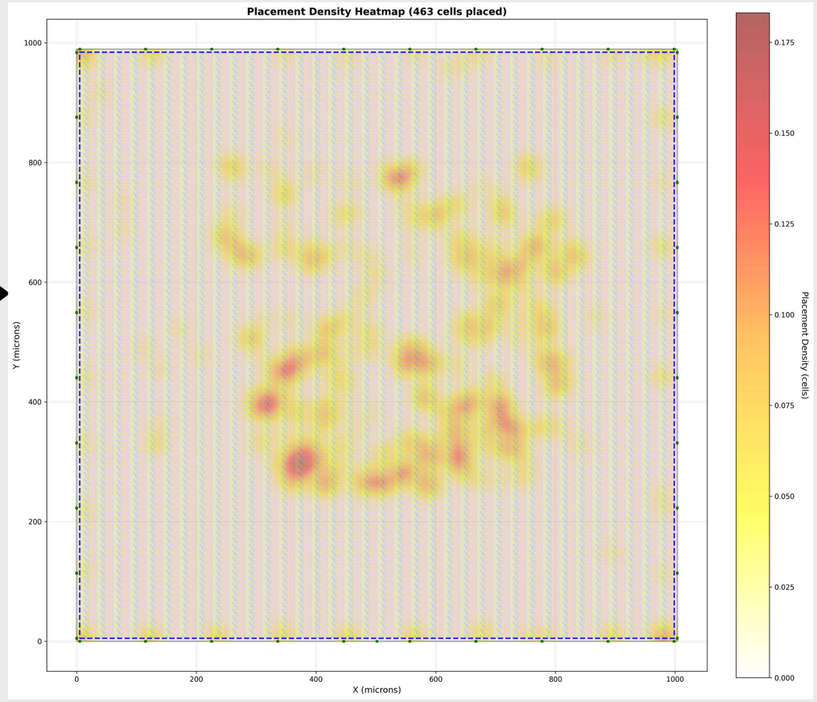
Median(88.8->31.5), mean(155.5->95.6) were reduced noticeably,
SA spreads the design to reduce congestion → some long nets get longer.
But there are very few of them, so overall average HPWL still drops.

SA effect of Arith

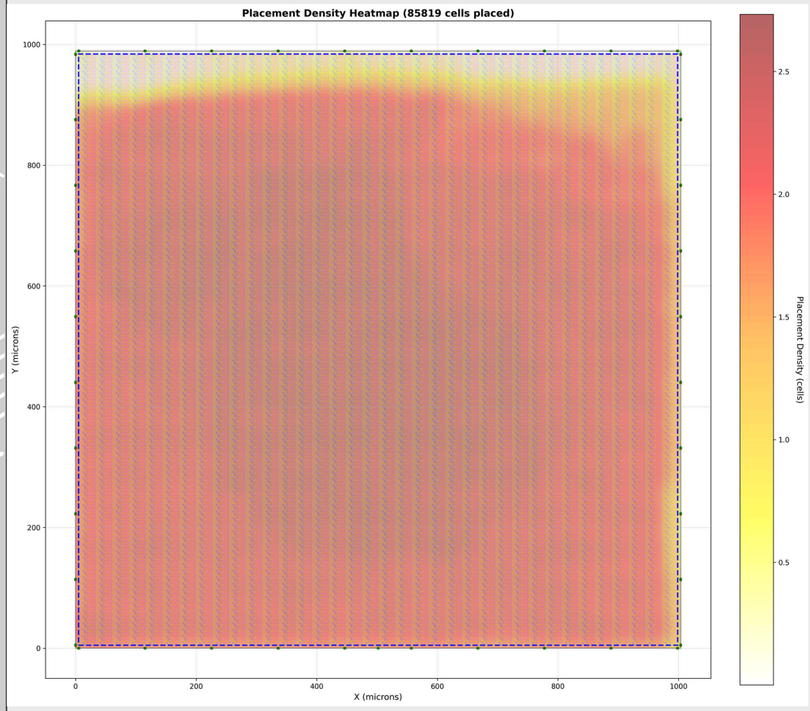
Arith



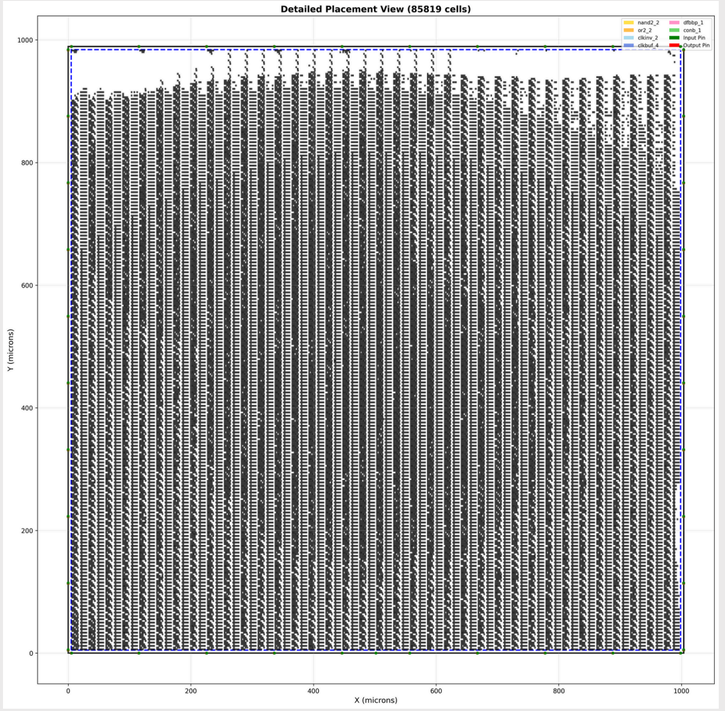
Arith



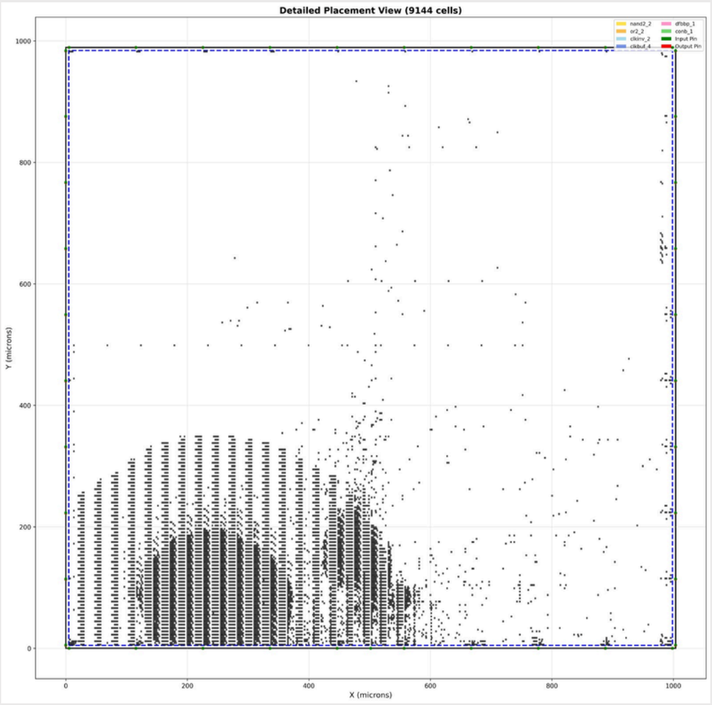
Results - placement density



aes_128

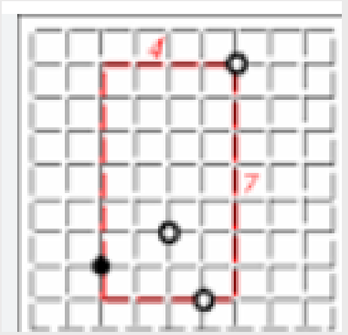


aes_128



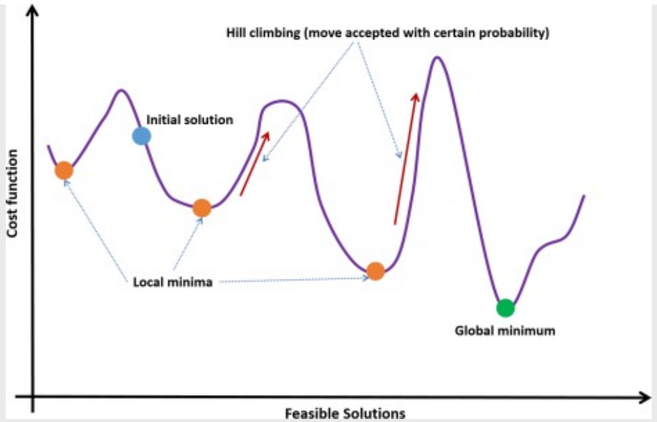
z80

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

$$HPWL = (max\_x - min\_x) + (max\_y - min\_y)$$



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 ΔE < 0
Always accept
Move makes the placement better
Case 2 — If ΔE ≥ 0 (worse move)
Use the Metropolis acceptance probability: P=e−ΔE/TP

Cooling Schedule
Temperature T decreases gradually:
Tnew=αToldT_{new} = \alpha T_{old}Tnew=αTold
Where α ≈ 0.90–0.99.
Effects:
High T → SA accepts many bad moves → global exploration
Low T → SA accepts almost no bad moves → fine refinement

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

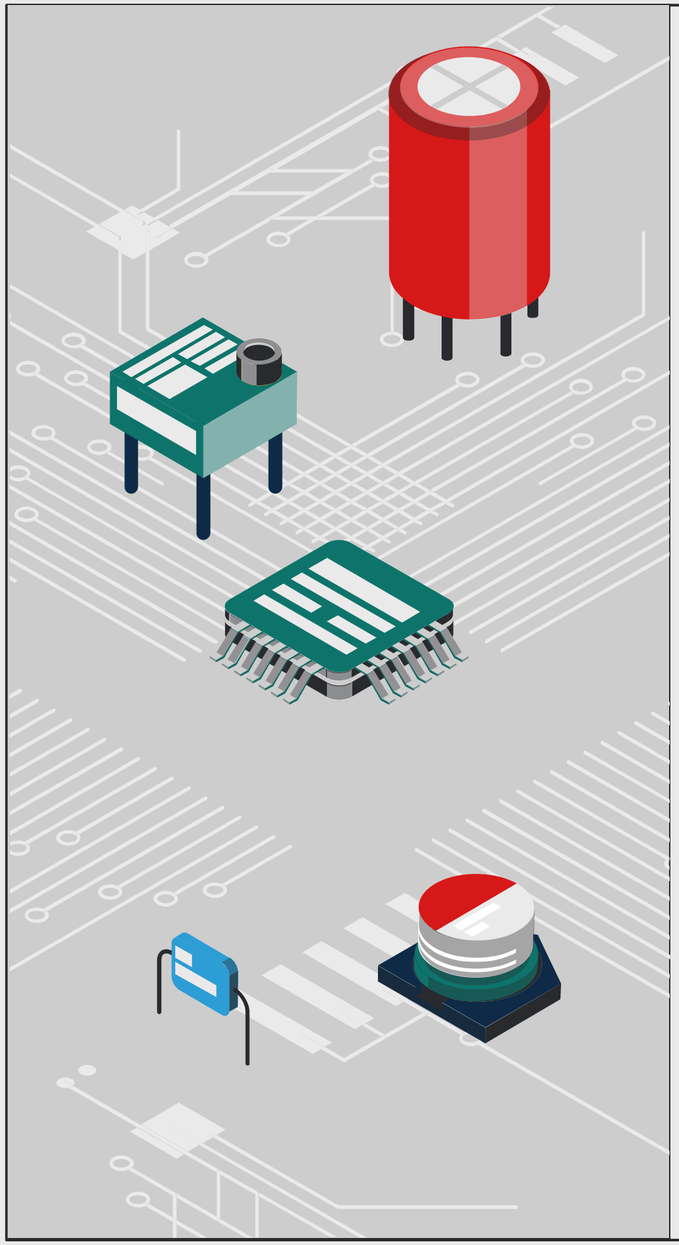
- Enables slot matching, distance/HPWL calculations, and I/O-driven seed placement
- Provides essential physical data required by the placer

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 ?