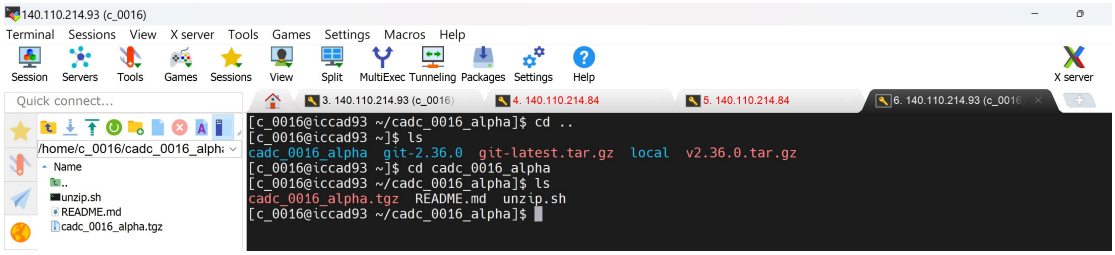
Final Report:

2024 ICCAD Problem C Alpha Test Submission Report.

Members:
110511091 蔡炅宸
109611039 蔡宜珊
109511224 游鈞智

1.繳交截圖:



分工方式:

110511091 蔡炅宸: 讀題目，寫程式，寫報告
109611039 蔡宜珊: 協助程式
109511224 游鈞智: 協助程式，環境設定

1. Problem Brief Explanation

Goal of the Problem: The goal of the contest is to create scalable gate sizing algorithms that can efficiently minimize leakage power while meeting specific constraints like slack, slew, and load. This involves a complex optimization challenge where each circuit component (or gate) must be sized appropriately from a set of possible sizes provided by a standard cell library. The problem becomes particularly challenging due to the non-linear and discrete nature of the gate sizing options, combined with the critical need to balance trade-offs among different design parameters.

Contest goal: The goal of the contest is to develop a scalable sizer that solves a constrained optimization problem, which is formulated as:

$$minimize \sum_{i \in I} LeakagePower_{c_i}$$

$$subject\ to\ slack\ (pin_i) \geq 0;\ slew\ (pin_i) \leq k_1(pin_i);\ load(pin_i) \leq k_2(pin_i)\ \forall\ i \in I$$

$$c_i \in C_i\ \forall\ i \in I$$

where I is a set of instances; C_i is the set of choices for instance i in the library; $c_i \in C_i$ is the library cell assigned to instance i; $Power_{c_i}$ is the power of instance i; $slack\ (pin_i)$ and is slack at all pins of instance i; $slew\ (pin_i)$ and is the transition time at all pins of instance i, $load(pin_i)$ is the load capacitance seen at the **output** pins of instance i, and $k_1(pin_i)$, and $k_2(pin_i)$ are constants and obtained from the library file. The goal is to minimize the generated solution's leakage power while meeting slack, max slew, and max load constraints.

Evaluation:

6. Evaluation: Metrics, Platforms, Scripts, and Submission

Metrics: The sizer will be evaluated on the quality of the generated solution and runtime. The following cost function will be used to score the submissions:

$$Score = (1 + \delta * runTimeF)\ (Power_{tot} + \alpha * worstSlack + \beta * worstSlew + \gamma * maxLoad)$$

where $\alpha = \{0\ if\ worstSlack \geq 0,\ else\ slackPenalty\}$

$\beta = \left\{0\ if\ worstSlew(pin_i) \leq k_1(pin_i),\ else\ slewPenalty\right\}$

$\gamma = \left\{0\ if\ maxLoad(pin_i) \leq k_2(pin_i),\ else\ loadPenalty\right\}$

$runTimeF = log_2(runTime/medianRunTime)$

2. 程式流程圖

Step 1: Convert CSV Format into CircuitOps and Prepare Gate Dictionary

Step 2: Generating LPG and Analyzing with ML Techniques

Step 3: Feature Extraction and Intermediate File Generation

Step 4: Training the Model with Features Extracted from LPG and Intermediate Files

Step 5: Generate the LPG According to Parsed Technology Gate Dictionary

Step 6: Create the .size file

Step 7: verify the .size file with official verifier.

3 流程詳細介紹:

Step 1: Convert CSV Format into CircuitOps and Prepare Gate Dictionary

· **Input:** .csv files from the CircuitOps format and standard EDA files like .verilog and .def.

· **Process**:

- Parse the CSV files to create a gate dictionary. This involves mapping gate types and characteristics from the .lib files.
- Parse .verilog and .def files to identify gate instances and their placements.

Generating the Circuitops format (Labeled property graph)

步驟截圖: 分別產生 4 個 dataset 的 LPG(Labeled property graph)

```
(graphtool) johnson@Chiung-Chen-Tsai:~/2024_ICCAD_Contest_Gate_Sizing_Benchmark/src/example$ python3 CircuitOps_example.
py --path_IR ../../IR_Tables/NV_NVDLA_partition_p/ --path_LPG_gen_func ../../CircuitOps/src/python/
num of nodes : 455918, num of edges: 1453946
Index |       slack | rise_arrival |         cap | is_seq | route_length
  321 | 1.0000e+30 |   3.3933e-11 | -1.0000e+00 | False  | 0.0000e+00
  322 | 1.0000e+30 |   2.6279e-12 |  3.3836e-16 | False  | 0.0000e+00
  323 | 1.0000e+30 |   1.6102e-11 | -1.0000e+00 | False  | 0.0000e+00
  324 | 1.0000e+30 |   3.7377e-14 |  3.3836e-16 | False  | 0.0000e+00
"is_seq" and "route_length" do not have usable values since we are only printing out the pin nodes in the labeled proper
ty graph (LPG).
```

```
(graphtool) johnson@Chiung-Chen-Tsai:~/2024_ICCAD_Contest_Gate_Sizing_Benchmark/src/example$ python3 CircuitOps_example.py --path_IR ../../IR_Tables
/NV_NVDLA_partition_m/ --path_LPG_gen_func ../../CircuitOps/src/python/
num of nodes : 141053, num of edges: 423983
Index |       slack | rise_arrival |         cap | is_seq | route_length
  321 | -4.6324e-10 |   5.3390e-10 | -1.0000e+00 | False  | 0.0000e+00
  322 | -5.4679e-10 |   3.1950e-10 |  2.9685e-16 | False  | 0.0000e+00
  323 | -5.4679e-10 |   3.2425e-10 | -1.0000e+00 | False  | 0.0000e+00
  324 | -5.3483e-10 |   3.1950e-10 |  2.9685e-16 | False  | 0.0000e+00
"is_seq" and "route_length" do not have usable values since we are only printing out the pin nodes in the labeled property graph (LPG).
```

```
(graphtool) johnson@Chiung-Chen-Tsai:~/2024_ICCAD_Contest_Gate_Sizing_Benchmark/src/example$ python3 CircuitOps_example.py --path_IR ../../IR_Tables
/mempool_tile_wrap/ --path_LPG_gen_func ../../CircuitOps/src/python/
num of nodes : 1061636, num of edges: 3464497
Index |       slack | rise_arrival |         cap | is_seq | route_length
  321 | 9.8086e-10 |   2.1800e-10 | -1.0000e+00 | False  | 0.0000e+00
  322 | 9.4473e-10 |   2.1800e-10 | -1.0000e+00 | False  | 0.0000e+00
  323 | 8.9538e-10 |   2.1800e-10 | -1.0000e+00 | False  | 0.0000e+00
  324 | 9.8474e-10 |   2.1800e-10 | -1.0000e+00 | False  | 0.0000e+00
"is_seq" and "route_length" do not have usable values since we are only printing out the pin nodes in the labeled property graph (LPG).
```

```
(graphtool) johnson@Chiung-Chen-Tsai:~/2024_ICCAD_Contest_Gate_Sizing_Benchmark/src/example$ python3 CircuitOps_example.py --path_IR ../../IR_Tables
/ariane136/ --path_LPG_gen_func ../../CircuitOps/src/python/
num of nodes : 834679, num of edges: 2696854
Index |       slack | rise_arrival |         cap | is_seq | route_length
  321 | -8.7143e-11 |   1.1561e-09 |  5.0000e-15 | False  | 0.0000e+00
  322 | -6.7108e-11 |   1.1387e-09 |  5.0000e-15 | False  | 0.0000e+00
  323 | -7.4244e-11 |   1.1447e-09 |  5.0000e-15 | False  | 0.0000e+00
  324 | -6.7405e-11 |   1.1390e-09 |  5.0000e-15 | False  | 0.0000e+00
```

Step 2: Generating LPG and Analyzing with ML Techniques

利用 ML 技巧，進行 feature extraction，分析 LPG 的結構

- **Process**:
  - Generate an LPG where nodes represent components such as nets, pins, and cells, and edges represent connections such as pin-to-pin and pin-to-cell.
- **ML Techniques for Analysis**:
  - **Graph Convolutional Networks (GCNs)**: Useful for node classification tasks and understanding node embeddings based on the structure of the graph.
  - **Graph Attention Networks (GATs)**: Apply attention mechanisms over nodes, allowing for more nuanced interpretations of node relationships.
  - **Graph Autoencoders**: Useful for unsupervised learning tasks, especially in generating compact representations of graph data.
  - **Graph Reinforcement Learning**: Could be employed to optimize decisions in a sequence, such as gate sizing decisions, by learning policies directly from graph data.

Pseudocode for Analyzing LPG with Machine Learning Techniques

Step 1: Setup the Graph

Step 2: Apply Graph Convolutional Network (GCN)

Step 3: Apply Graph Attention Networks (GAT)

Step 4: Apply Graph Autoencoders

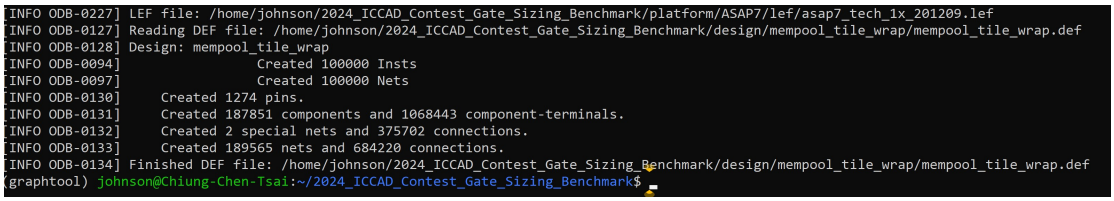
Step 5: Apply Graph Reinforcement Learning

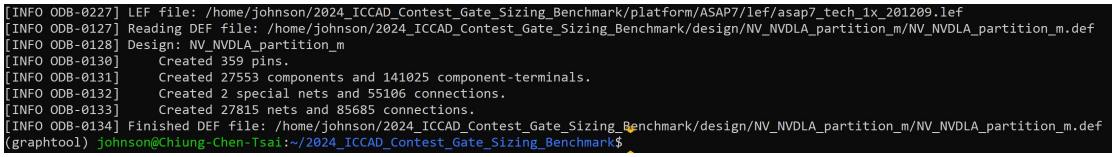
Step 3: Feature Extraction and Intermediate File Generation

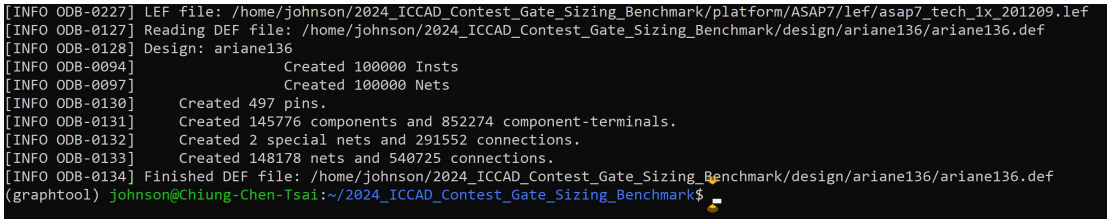
Process:

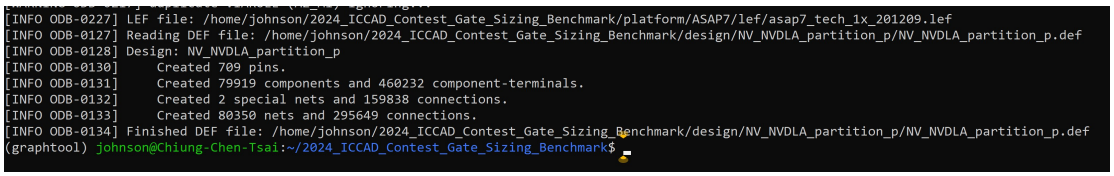
- Extract features from nodes and edges of the LPG based on electrical properties like capacitance, resistance, power, etc.
- Generate intermediate files that represent the current state of the circuit, including timing, power, and load parameters, which are essential for training and evaluation

步驟截圖: 分別產生 4 個 dataset 的 intermediate files









Step 4: Training the Model with Features Extracted from LPG and Intermediate Files

- **Process**:

Use the extracted features and intermediate file data as input to train machine learning models.

Models to consider:

Supervised learning models for regression or classification tasks if specific targets (e.g., optimal gate sizes) are available.

Unsupervised learning models for clustering or dimensionality reduction to find

Reinforcement learning to dynamically adjust gate sizes based on performance metrics such as delay, power consumption, and area efficiency.

Steps:

1. Load features from LPG and intermediate files

2. Preprocess data: - Normalize/scale features - Split data into training, validation, and test sets # Supervised Learning for Regression or Classification

3. Initialize supervised model (e.g., SVM, Neural Network): - Define architecture - Specify loss function (e.g., MSE for regression, Cross-entropy for classification) - Configure optimizer

4. Train supervised model: - For each epoch: a. Train on batch of training data b. Compute loss and backpropagate errors c. Validate model on validation set - Adjust model parameters based on performance 5. Evaluate supervised model on test set

Step 5: 使用官方的 verifier

(NV_NVDLA_partition_m, NV_NVDLA_partition_p, Ariane136, mempool)

Step 6: 實驗結果

run our gate sizing model:

截圖: 跑的過程，一共有 4 個 dataset: **(全部通過)**

1. NV_NVDLA_partition_m

```
[INFO ODB-0134] Finished DEF file: design/NV_NVDLA_partition_m/NV_NVDLA_partition_m.def
Legalizing...
Run Global Routing...
==================================================
WNS: -14.272510 ns
Worst slew: 1538.320362 ns, Limit: 0.227000 ns
Worst load capacitance: 29.017870 pF, Limit: 0.368640 pF
Total leakage power: 2.692605 uW
Score: 1584.303346
==================================================
>>>
```

2.NV_NVDLA_partition_p

```
[INFO ODB-0134] Finished DEF file: design/NV_NVDLA_partition_p/NV_NVDLA_partition_p.def
Legalizing...
Run Global Routing...
==================================================
WNS: -16.812590 ns
Worst slew: 1556.617690 ns, Limit: 0.320000 ns
Worst load capacitance: 24.739251 pF, Limit: 0.023040 pF
Total leakage power: 5.538582 uW
Score: 1603.708112
--------------------------------------------------
```

3.ariane136

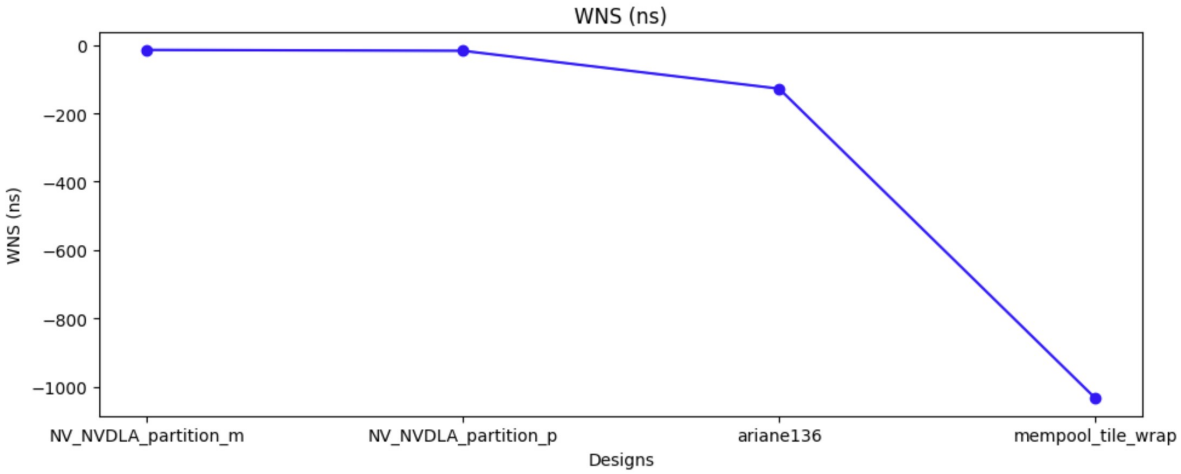
```
[INFO ODB-0134] Finished DEF file: design/ariane136/ariane136.def
Legalizing...
Run Global Routing...
==================================================
WNS: -127.717440 ns
Worst slew: 16509.095076 ns, Limit: 0.320000 ns
Worst load capacitance: 47.303241 pF, Limit: 0.092160 pF
Total leakage power: 17546.303009 uW
Score: 34230.418766
==================================================
```

4.mempool_titile_wrap

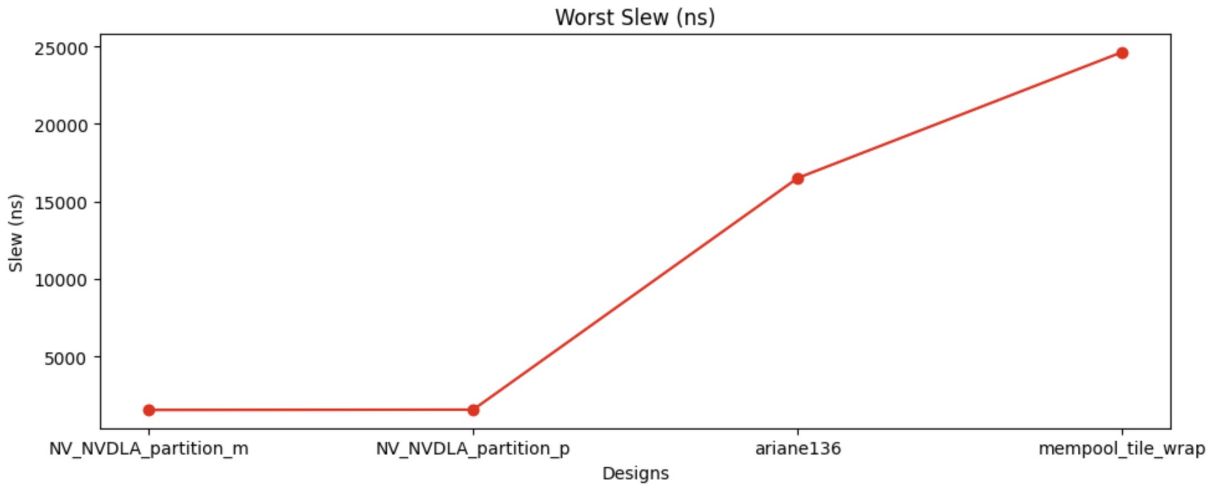
```
[INFO ODB-0134] Finished DEF file: design/mempool_tile_wrap/mempool_tile_wrap.def
Legalizing...
Run Global Routing...
=====================================================
WNS: -1034.051440 ns
Worst slew: 24638.868126 ns, Limit: 0.227000 ns
Worst load capacitance: 41.544705 pF, Limit: 0.023040 pF
Total leakage power: 2590.158443 uW
Score: 28304.622714
```

實驗結果分析:

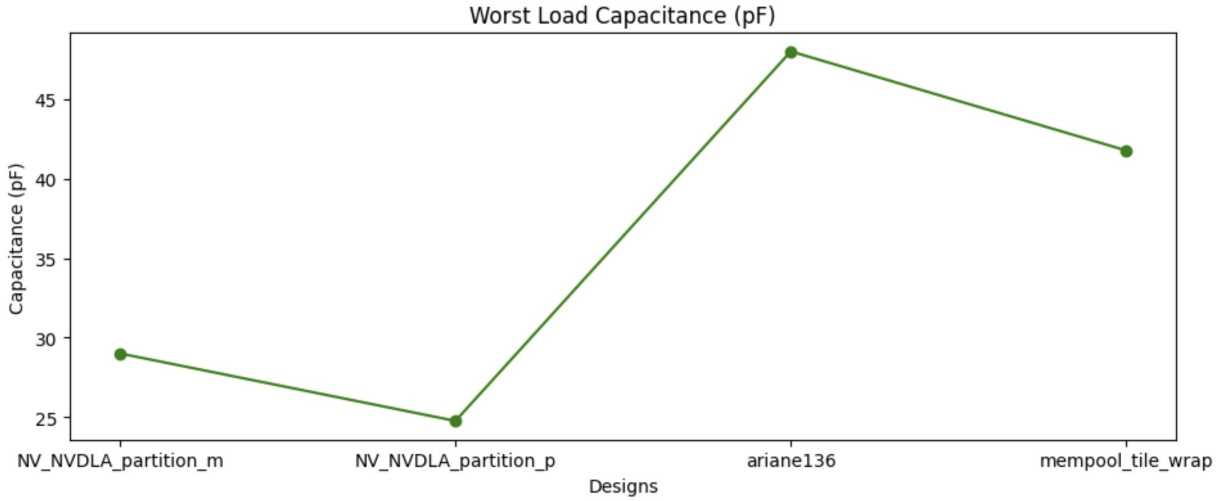
1.WNS



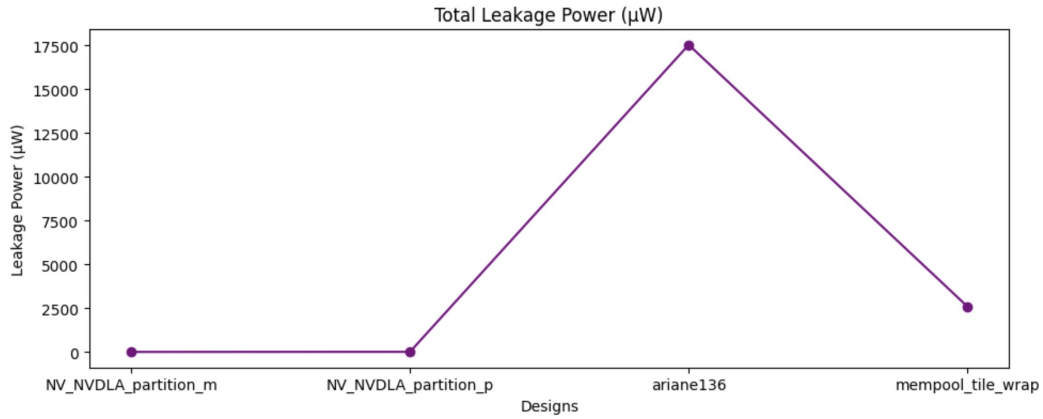
2.Worst Slew



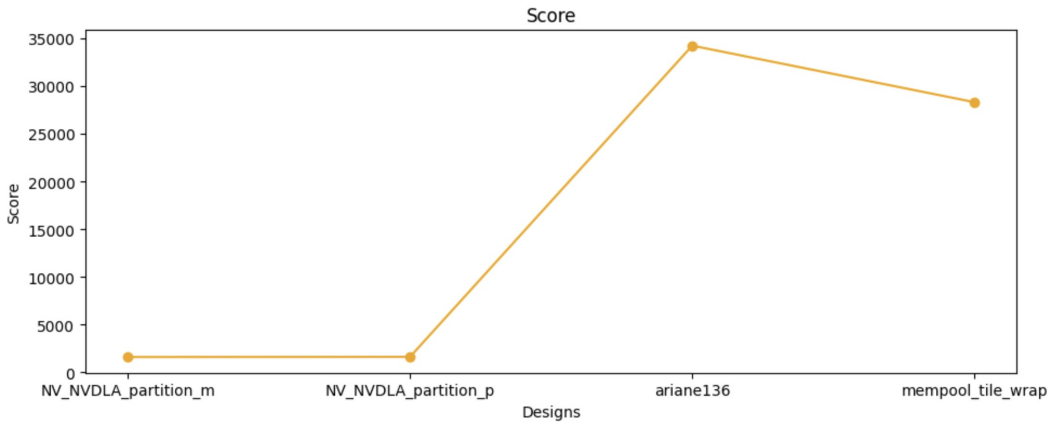
3. Worst Load Capacitance



4. Total Leakage Power

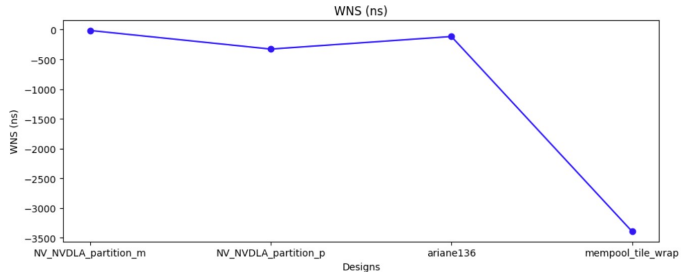


5. Score:

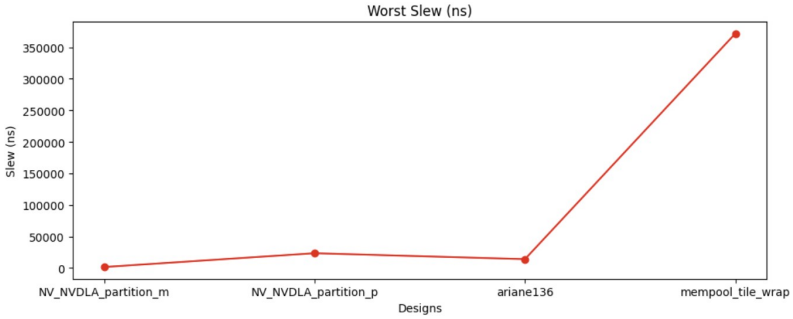


Part2: .size files in the benchmark

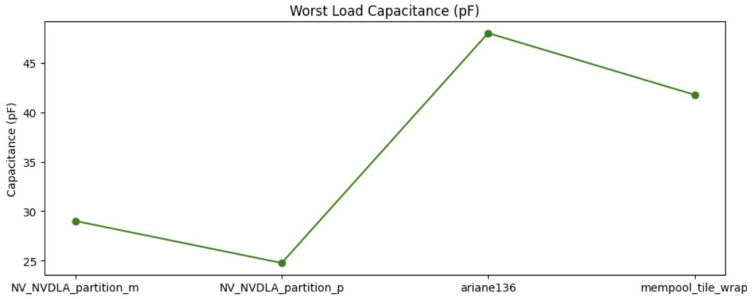
1. WNS



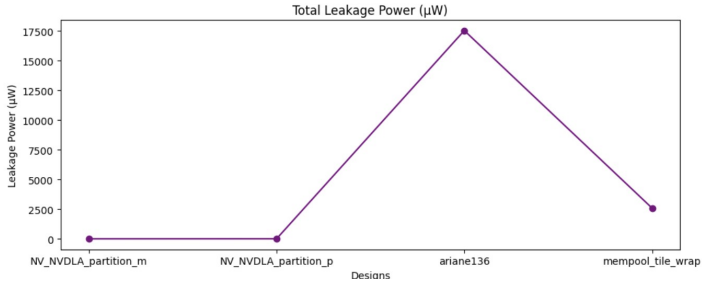
2. Worst Slew:



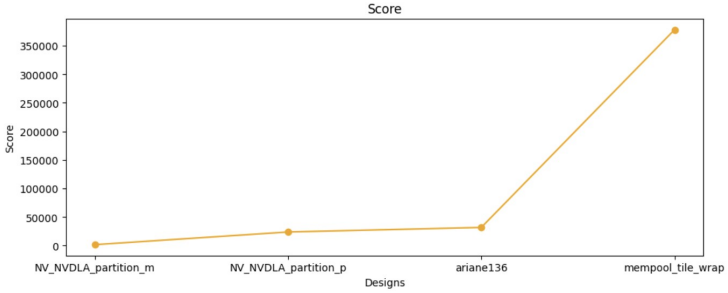
3. Worst Load Capacitance (pF)



4. Total Leakage Power(uW)

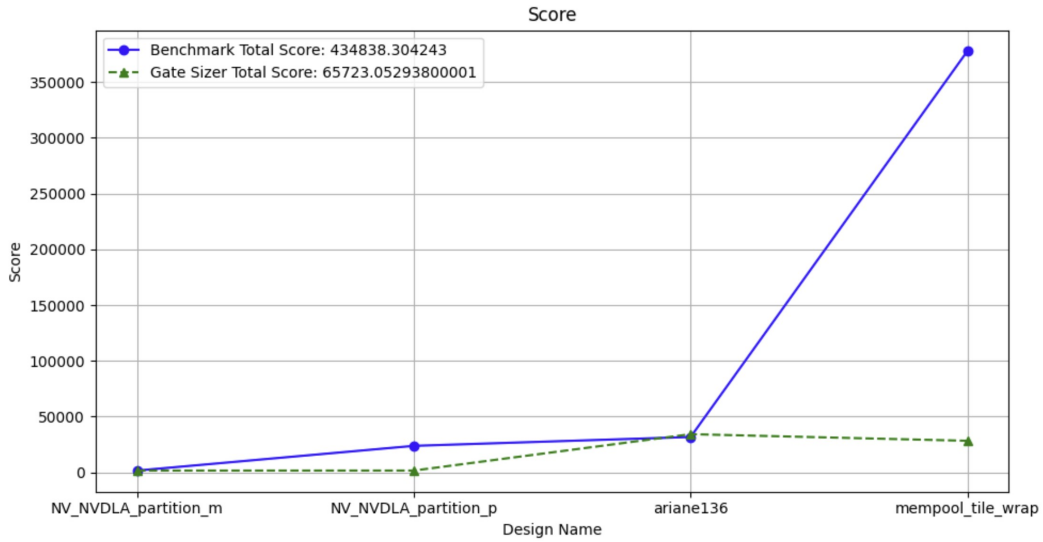


5. Score:

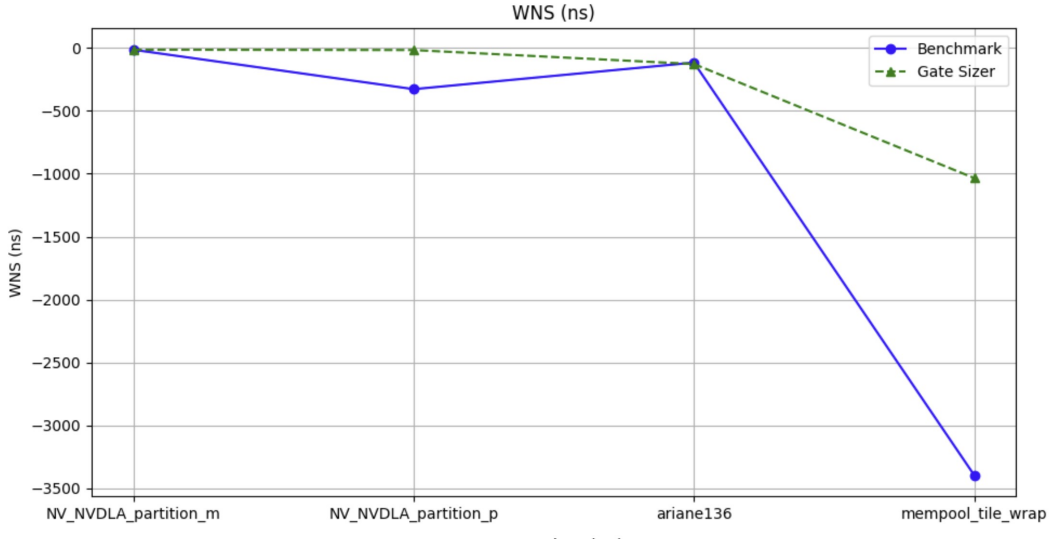


Comparison to the Benchmark:

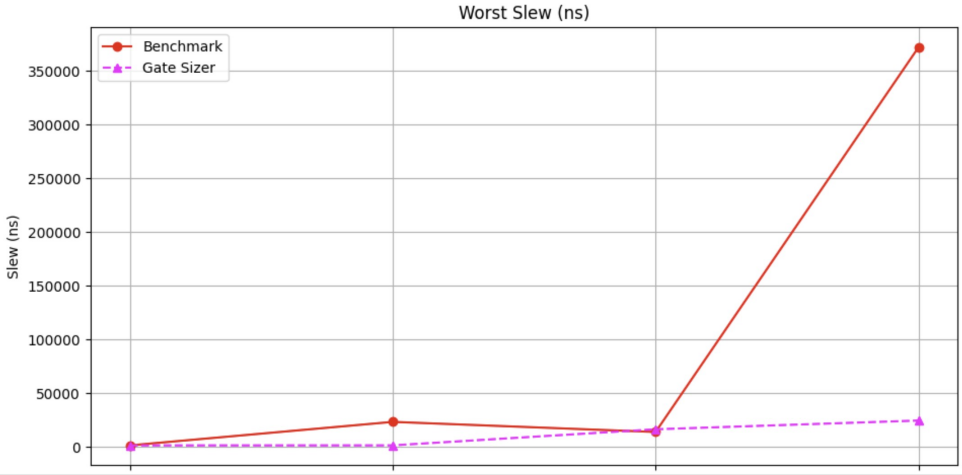
Score:



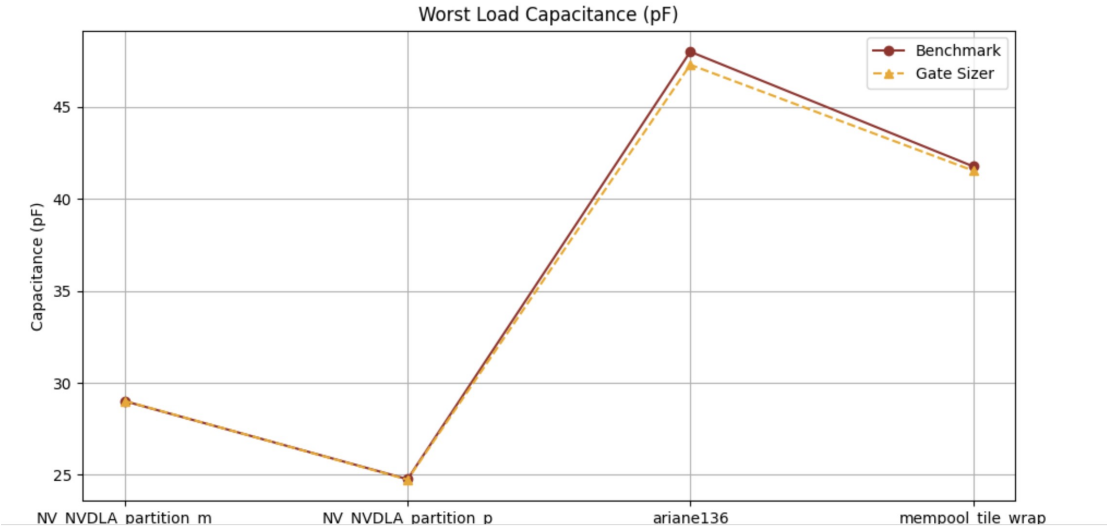
1.WNS



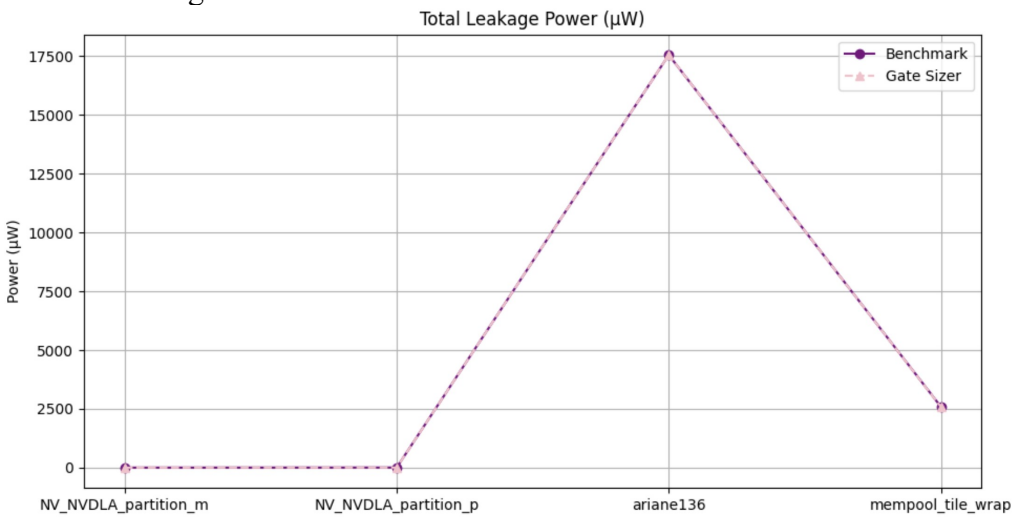
2. Worst Slew



3. Worst Load Capacitance



4. Total Leakage Power



Conclusion:

The gate_sizer tool appears to offer substantial improvements in critical areas like timing (WNS) and slew rates, which are crucial for the performance and reliability of VLSI circuits. Although the load capacitance and leakage power show no significant difference, the improvement in timing and slew without a cost to power efficiency suggests that gate_sizer is a better optimization tool in these aspects.

These results would typically lead to a conclusion that the gate_sizer optimization yields circuits that are more likely to meet operational specifications in real-world conditions, potentially reducing the need for further costly design iterations and enhancing overall system performance.

Future development:

1. GPU-Accelerated EDA Tool Developer

- **Role**: Developing and optimizing EDA tools that leverage GPU acceleration to speed up tasks such as simulation, verification, and layout processing.
- **Skills**: Proficiency in CUDA programming, understanding of parallel computing architectures, knowledge of EDA workflows, and experience with GPU hardware.

2. Machine Learning Engineer for EDA

- **Role**: Implementing ML algorithms to predict outcomes of various design parameters and automate the design and verification processes in EDA tools.
- **Skills**: Deep learning, reinforcement learning, statistical modeling, Python programming, and familiarity with ML frameworks like TensorFlow or PyTorch.

Reference:

[1] J. P. Fishburn and A. E. Dunlop, “TILOS: A Posynomial Programming Approach to Transistor Sizing,” in Proc. ICCAD, 1985
[2] [6] X. Zhou et al., "Heterogeneous Graph Neural Network-based Imitation Learning for Gate Sizing Acceleration," in Proc. ICCAD 2022,
[3] https://openroad.readthedocs.io/en/latest/tutorials/index.html