# ReSynthAI: Physical-Aware Logic Resynthesis for Timing Optimization using AI

Vikram Gopalakrishnan, Rongjian Liang, Atmadip Dey, Yanqing Zhang, and Vidya A. Chhabria

## 0. Version History

04/23/2025: Contest problem released publicly.
05/16/2025: Information about LEC and alpha submission information uploaded for participants.

## 1. Introduction

With scaling and the slowdown in Moore's law, EDA tools must work increasingly harder to achieve power, performance, and area (PPA) specifications. Logic synthesis, a critical part of the design cycle typically does not contain physical information, and any timing optimization that is performed as a part of logic synthesis or immediately after logic synthesis is sub-optimal as it is unaware of the locations of cells, wire lengths, and parasitics. Further, netlist optimization that includes repairing timing and electrical violations is largely heuristic-based at all the stages of the flow, increasing the complexity of the challenge to generate designs with good quality results (QoR).

Netlist optimization for fixing timing and electrical violations can include a variety of transformations, including logic gate sizing and Vt swaps, buffer insertions or removal, gate cloning, and logic restructuring. It is often challenging to decide which of the above netlist transformations or sequence of transformations must be applied and in what order for the best QoR. The problem is challenging due to (i) the non-convexity of the delay models, (ii) the discrete space of available logical equivalent gates in the library, (iii) the number of near-critical paths, (iv) the tradeoff between power and timing, and (iv) the resource availability constraints which prevent the exploration of the complete design space.

This problem has been studied for over three decades now. There have been works that have looked at logic gate sizing [1–5], buffer insertion [6,7], and physical aware logic synthesis [8,9]. However, these works have been looked into individually, with each paper considering one type of transformation. These techniques are based on heuristics and often tradeoff runtime for QoR. With the advent of ML, there have been ML-based approaches in recent research[6, 10–15]. Since ML can help efficiently search a large space via fast analysis or perform faster optimization. Yet, these techniques have only looked at logic optimization individually at the gate sizing problem or the buffer insertion problem. Some of the ML-based approaches that leverage reinforcement learning are still not scalable [13,15]. Simultaneously performing multiple transformations, identifying the best transformation to be applied for a specific timing path, with physical awareness has been unexplored. This contest aims to drive research to consider several transformations simultaneously, along with physical awareness, to generate better post-placement QoR. In this context, the contest serves three purposes:

1) Explore the state-of-the-art algorithms for physical-aware resynthesis to drive academic research to generate scalable algorithms for gate sizing, gate cloning, buffer insertion or removal, Vt swapping, and logic restructuring using ML (supervised, unsupervised, or reinforcement learning-based techniques)

2) Lower the barrier to entry for non-EDA experts by converting traditional EDA problems to an ML-solvable problem through ML-friendly data representation formats.
3) Release benchmarks and create a contest leaderboard for physical-aware resynthesis and QoR improvement post-placement in a FinFET technology node with evaluation scripts to accelerate research.

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

Minimize: $\sum_{i \in I} LeakagePower_{c_i}$ where $c_i \in C_i$

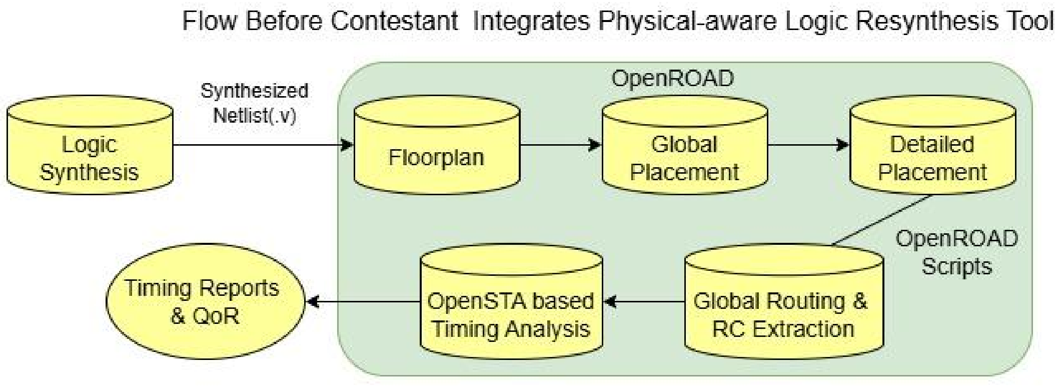
Subject to: $slack(pin_i) \geq 0, \quad \forall\, i \in I$

$slew(pin_i) \leq k_1(pin_i), \ \forall\, i \in I$

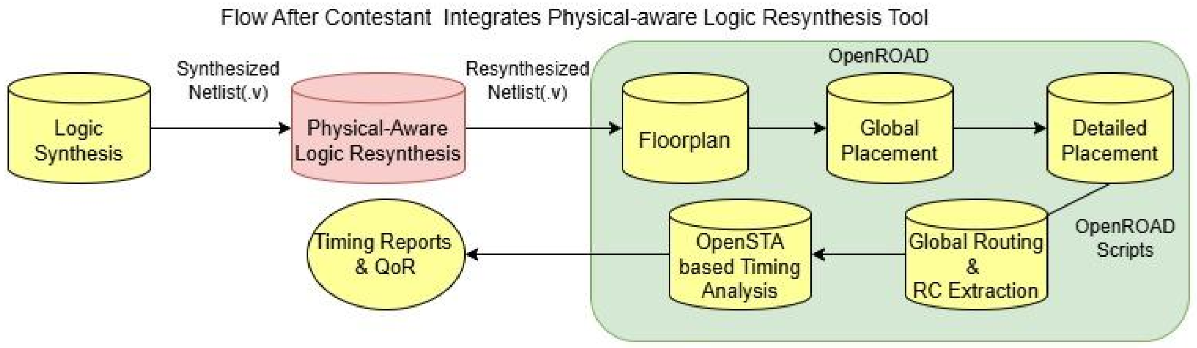
$load(pin_i) \leq k_2(pin_i), \ \forall\, i \in I$

$\#overflow = 0$

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 overflow is that obtained post global route while keeping the design routable. The resynthesized netlist must pass the logic equivalence check versus the synthesized netlist. The goal is to minimize the generated solution's leakage power while meeting slack, max slew, max load, and routability constraints post-global route as shown in the flow below.


Flow Before Contestant Integrates Physical-aware Logic Resynthesis Tool
Logic Synthesis
Synthesized Netlist(.v)
OpenROAD
Floorplan
Global Placement
Detailed Placement
OpenROAD Scripts
Global Routing & RC Extraction
OpenSTA based Timing Analysis
Timing Reports & QoR

After contestants create their tool, it will be integrated into the flow after logic synthesis and before the backend steps. The evaluation flow would look like this:


Flow After Contestant Integrates Physical-aware Logic Resynthesis Tool
Logic Synthesis
Synthesized Netlist(.v)
Physical-Aware Logic Resynthesis
Resynthesized Netlist(.v)
OpenROAD
Floorplan
Global Placement
Detailed Placement
OpenROAD Scripts
Global Routing & RC Extraction
OpenSTA based Timing Analysis
Timing Reports & QoR

## 2. Input/Output Files and Formats

### Input Formats

The contest will provide inputs in two formats. The first is standard EDA files, and the second is CircuitOps' intermediate data representation format. Both formats are explained below:

**(i) Standard EDA file formats:**

- .v file: Gate-level verilog netlist post-synthesis
- .def file: Floorplan of the design with fixed IO, Macro, and boundaries. [This cannot be changed, the same floorplan will be used for evaluation.]
- .sdc file: Constraint file provided in TCL to specify input port delay and transition, output port capacitance, and specified clock period
- .lib file: Library file that consists of the look-up-tables for delay, slew, and power computation with the area of each cell
- .lef file: Technology lef and cell lef

If participants choose to work with these files, they are expected to work with them as input and develop parsers. If participants choose to develop their solutions in C/C++, they can leverage OpenROAD [16] and OpenSTA (submodule in OpenROAD) [17] header files to parse liberty and netlist. If participants choose to develop their solutions in Python, they can use OpenROAD Python APIs to query the necessary information, or they can reuse existing Python packages such as liberty-parser [18] and verilog-parser [19]. Participants can also work with their parsers if they cannot extract the information they need from existing APIs. We have provided example APIs in scripts for querying information from OpenROAD DB if you choose to use OpenROAD as the parser and work within the OpenROAD Python interpreter(link).

**(ii) CircuitOps data representation format**:
The CircuitOps data representation format provides an ML-friendly format consisting of graphs and annotated features. The graph is a labeled property graph that is backed by relational tables. The graph utilizes a node for a net, pin, and cell of a circuit. It has four types of edges to represent connections between pin-to-pin nodes (pin-pin), pin-to-cell nodes (pin-cell), pin-to-net

nodes (pin-net), and cell-to-net nodes (cell-net). Each node and edge has associated relational tables with properties associated with the node and edge. The relational tables will be provided to the contestants, which can easily be parsed into their solution using standard CSV readers in C++/Python. Details on the list of properties and their description and types of nodes and edges can be found in the Appendix, the CircuitOps paper [20], and the GitHub repository [21]. **These files will be in a CSV format.**

The choice of the input format is left to the participants. The first EDA file-based format will provide you with maximum information to develop your physical-aware synthesis tool and the ability to extract custom features for ML models. The second will be limited to the features and properties available in CircuitOps, but it will allow you to skip the tedious parsing tasks and focus on the ML/GPU acceleration aspects. We will release translators from standard EDA file format to CircuitOps format (the translators will leverage OpenROAD APIs and database queries). Please note that the translators are slow and cannot be done iteratively within your sizing algorithm. Some of the properties in CircuitOps can be computed fast from OpenROAD (the properties that come in the intermediate files), but not all properties.

**Intermediate Files**

In case you are using CircuitOps and do not want to build your independent timing engine, we have provided you with a script (link) that takes the temporary output file (see format below), the input EDA files, and generates the following outputs. The runtime for this script is about one minute for the "NV_NVDLA_partition_m" benchmark. These files will provide timing, power, load capacitance, and power information from the OpenROAD timer. The timing information (.csv) will be stored in a file with the format below:

**Output File Format**

The Verilog netlist(.v) file from your tool. We will run placement and global route on this netlist in OpenROAD using a predefined floorplan def file (with fixed boundaries and macro locations, etc.) that will also be provided to you in advance.

## 3. Benchmarks

We will release benchmarks in the ASAP7 technology node (link). The benchmarks will vary in size from 10K-300K instances. We have released 5 small visible benchmarks(more larger ones will be added soon) and we have some hidden benchmarks that we will use to evaluate the solutions from the participants. And we will also provide reasonably good quality .v files that can be used for training purpose in supervised learing-based ML techniques.

## 4. Developing your tool: Rules, assumptions, timer, and support scripts.

**Rules:** Sequential gates are do-not-touch. Macros or sram cells are do-not-touch. Moving Logic across pipeline stages to meet timing will not be permitted. Sequential Optimization is not allowed. Only combination logic can be altered. The nets that connect to the FF should not be renamed. There will be a logical equivalence test between given netlist and resynthesized netlist. Designs that do not meet the test will not be considered while scoring.

**Assumptions:** The contest will focus on optimizing setup time only. All sequential gates in the benchmark will be triggered by a rising edge. The benchmarks will assume an ideal clock. There will be no clock buffers in the circuit (zero skew). The contest will be set up on post-synthesis benchmarks. We have provided global routed results annotated with CircuitOps format (refer to the appendix).

**Timing Engine:** You can implement your timer and power estimator within your tool. However, the final evaluation will use OpenROAD's timer (OpenSTA). You may also use OpenROAD's timer and power estimator for your physical-aware resynthesis. We will provide examples of Python APIs interacting with the OpenROAD timer. The APIs will be in Python and white-boxed. If you are developing your code in C++, you can utilize the corresponding C++ APIs that the Python APIs wrap around within OpenROAD.

**Support Scripts:** We will provide the necessary support scripts for the contest:

(i) Example script that uses OpenROAD Python API to make each transformation to the netlist.
(ii) Script to generate IR tables by mentioning the path to the def file.

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

**Logical equivalence:** To ensure logical equivalence of the netlists, we will ensure that your tool-generated netlist and the original netlist are logically equivalent. To check this we will use two approaches:

1) Simulation-based verification using GL0AM.
2) Formal verification: We perform formal verification using eqy yosys command.

Only netlists generated that pass logical equivalence will be further evaluated for other metrics as described below. We have provided example scripts to verify the logical equivalence for these two tools. Note that the formal verification is extremely slow and we will only evaluate it for the top 5 teams using formal verification. The simulation-based verification leverages GPU acceleration and therefore is faster, and participants are welcome to use this as they build their solvers.

Script to perform logical equivalence checks (LEC) on the updated netlist and golden netlist.
Evaluation script to evaluate your solution and get a sample score.

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

$$
\begin{aligned}
Score = {} & (c + runTimeF) \\
& * ( ( Power_{tot}) \\
& \quad + \alpha * abs(TNS) \\
& \quad + \sum_i \beta_i * (worstSlew(pin_i) - k_1(pin_i)) \\
& \quad + \sum_i \gamma_i * (maxLoad(pin_i) - k_2(pin_i)) \\
& \quad + \delta * \#overflow )
\end{aligned}
$$

where $\alpha, \beta_i, \gamma_i\ and\ \delta$ are constants that will be determined based on the solutions obtained. Units: The $Power_{tot}$ and $OrignialBenchmarkPower_{tot}$ are in micro Watt. The $TNS$ is in nanoseconds. The $(worstSlew(pin_i) - k_1(pin_i)$ is in nanoseconds and $(maxLoad(pin_i) - k_2(pin_i))$ is in femto Farad.

The $runTime$ is the runtime of the benchmark being evaluated, and the $medianRunTime$ is the median of the runtimes of all teams' submissions on the benchmark being evaluated. Note the evaluation will perform placement legalizing post-resynthesis, rerun global routing, and update parasitics, then estimate the above score. The values of $k_1$ and $k_2$ are specific to the pins of each liberty cell and are defined in the liberty file. They can be extracted using the OpenROAD APIs as shown in this script. These values are also provided as a part of the IR tables and can be queried using OpenROAD. Each participating team will be given a score for each benchmark.

In the final evaluation, the score for each test case will be scaled to 100 using the following equation: $100 \times \frac{Best\ least\ score\ across\ all\ team}{contestant's\ score}$. We will set up a runtime . In the process of calculating the median runtime, we will not take the runtime of failed cases into account. The scores will be added across all test cases, and the top three teams with the **highest** scores will win the contest.

**Platforms:** The submitted solutions will run on a computing platform equipped with one A100 GPU, which has a memory capacity of 80GB. The utilization of up to 8 CPU threads will be supported. We have supplied a docker file preconfigured with CUDA and some popular ML libraries to facilitate a standardized development environment. This Dockerfile installs OpenROAD, GL0AM, and all required software packages. Participants must build on top of this dockerfile. We encourage participating teams to capitalize on the potential of GPU acceleration and explore the integration of ML techniques. However, the usage of the GPU is optional, and teams are free to choose their preferred approach, whether it involves leveraging the GPU or not.

**Scripts:** We have released the first version of our evaluation scripts (link) so teams can test their scores on visible benchmarks. The scores reported are only for the power, worst slack, worst slew, and max load components of the score with equal weightage to α, β, δ*,* and γ. The values of the weights will be updated shortly in this document and the script. The runtime-related components will be evaluated on the platform and compared with other Participants.

**Submission Process:** Teams are required to build a Docker image on top of the provided Dockerfile. The Dockerfile must be built on this existing Dockerfile. Dropbox links will be provided to each team with write access only to the primary member of the team. You can submit your docker image to the designated folder with the name strictly in the given format <*teamid_MLCAD25contest_alpha*>. Please refer to the Platforms section to see the hardware details.
Please archive your Docker image to a tar file and use gzip to compress it to a tar.gz file (refer to https://docs.docker.com/engine/reference/commandline/save/). Then upload the tar.gz file to the Dropbox Drive upload link shared by the Contest Organizers in separate emails.

Within the Docker environment, please create a directory named "resynthesizer" under the existing "/app" folder and place the binary/scripts in this directory (/app/resynthesizer). The submission should contain a top-level **"***MLCAD25_resynth.sh*" in the /app/resynthesizer directory which takes the design name as the ONLY input argument and must name the output as <design_name>_resynth.v file using the design name. For example, if the design name is NV_NVDLA_partition_m, the output netlist should be named NV_NVDLA_partition_m_resynth.v.

Please name the Docker image and the final tar.gz file using the team ID. For example, if the team ID is team1, then use team1 for the Docker image and team1.tar.gz for the submitted file. Please **DO NOT** upload other files. Submissions not following the submission process instructions will not be graded.

Example: MLCAD25_resynth.sh file (You are allowed to modify them, such as adding more flags, as long as the top-level MLCAD25_resynth.sh script only takes the design name as the only input.):

Example 1: In case the resynthesis script is a binary file named resynthesizer

```
design_name = $1
/app/resynthesizer/resynthesizer -lef
/app/MLCAD25-Contest-Scripts-Benchmarks/platform/ASAP7/lef -lib
/app/MLCAD25-Contest-Scripts-Benchmarks/platform/ASAP7/lib -def
/app/MLCAD25-Contest-Scripts-Benchmarks/designs/$design_name/EDA_files/$de
sign_name_fp.def.gz -IR_Table
/app/MLCAD25-Contest-Scripts-Benchmarks/designs/$design_name/IR_tables_fp/
-output /app/resynthesizer/output/$design_name_resynth.v
```

Example 2: If the sizer is a Python script file around OpenROAD or even a general Python script:

```
design_name = $1
/app/MLCAD25-Contest-Scripts-Benchmarks/build/src/openroad -python
/app/resynthesizer/resynthesizer.py -lef
/app/MLCAD25-Contest-Scripts-Benchmarks/platform/ASAP7/lef -lib
/app/MLCAD25-Contest-Scripts-Benchmarks/platform/ASAP7/lib -def
/app/MLCAD25-Contest-Scripts-Benchmarks/designs/$design_name/EDA_files/$de
sign_name_fp.def.gz -IR_Table
/app/MLCAD25-Contest-Scripts-Benchmarks/designs/$design_name/IR_tables_fp/
-output /app/resynthesizer/output/$design_name_resynth.v
```

## References

[1] J. P. Fishburn and A. E. Dunlop, “TILOS: A Posynomial Programming Approach to Transistor Sizing,” in Proc. ICCAD, 1985.
[2] J. Hu, et al., “Sensitivity-Guided Metaheuristics for Accurate Discrete Gate Sizing,” in Proc. ICCAD, 2012.
[3] K. Kasamsetty, et al., “A New Class of Convex Functions for Delay Modeling and its Application to the Transistor Sizing Problem,” IEEE T. Comput. Aid. D., vol. 19
[4] A. Sharma, et al., “Fast Lagrangian Relaxation-Based Multithreaded Gat, no. 7, pp. 779–788, 2000.e Sizing Using Simple Timing Calibrations,” IEEE T. Comput. Aid D., vol. 39, no. 7, pp. 1456–1469, 2020.
[5] C.-P. Chen, et al., “Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation,” IEEE T. Comput. Aid D., vol. 18, no. 7,pp. 1014–1025, 1999.
[6]R. Liang, S. Nath, A. Rajaram, J. Hu, and H. Ren. "BufFormer: A Generative ML Framework for Scalable Buffering." *in ASPDAC '23*.
[7]L. Zhang, B. Li, and U. Schlichtmann. "Sampling-based Buffer Insertion for Post-Silicon Yield Improvement under Process Variability." *in DATE '17*
[8]H. Pan, C. Lan, Y. Liu, Z. Wang, L. Shang, X. Zeng, F. Yang, and K. Zhu. "Physically Aware Synthesis Revisited: Guiding Technology Mapping with Primitive Logic Gate Placement." *in ICCAD '24*.
[9]S. Chatterjee and R. Brayton. "An Incremental Placement Algorithm and Its Application to Congestion-Aware Logic Synthesis." *in ICCAD '04.*
[10]B.-Y. Wu, R. Liang, G. Pradipta, A. Agnesina, H. Ren, and V. A. Chhabria. "2024 ICCAD CAD Contest Problem C: Scalable Logic Gate Sizing Using ML Techniques and GPU Acceleration." *in ICCAD '24*.
[11]C. Karfa, C. R. Chigarapalli, H. Bhakkad, S. Bhattacharjee, and A. B. Chowdhury. "A Machine Learning Guided Cut Choices for ASIC Technology Mapping." *in ICCAD '24*.
[12] X. Zhou et al., "Heterogeneous Graph Neural Network-based Imitation Learning for Gate Sizing Acceleration," in Proc. ICCAD 2022.
[13] Y. -C. Lu, S. Nath, V. Khandelwal and S. K. Lim, "RL-Sizer: VLSI Gate Sizing for Timing Optimization using Deep Reinforcement Learning," in Proc DAC, 2022.

[14] S. Nath, G. Pradipta, C. Hu, T. Yang, B. Khailany and H. Ren, "TransSizer: A Novel Transformer-Based Fast Gate Sizer," in Proc. ICCAD, 2022.
[15]W. Jiang, V. A. Chhabria, and S. S. Sapatnekar. "IR-Aware ECO Timing Optimization Using Reinforcement Learning." *in MLCAD '24*.
[16]https://github.com/The-OpenROAD-Project/OpenROAD
[17]https://github.com/The-OpenROAD-Project/OpenSTA
[18]https://pypi.org/project/liberty-parser/
[19]https://pypi.org/project/verilog-parser/
[20]R. Liang, A. Agnesina, G. Pradipta, V. A. Chhabria and H. Ren, "Invited Paper: CircuitOps: An ML Infrastructure Enabling Generative AI for VLSI Circuit Optimization," in Proc. ICCAD 2023.
[21]https://github.com/NVlabs/CircuitOps

## Appendix

**CircuitOps format description:**

**Cell properties:**
Cell_name: name of the cell
Is_seq: whether the cell is sequential (register, flip flop, etc.)
Is_macro: whether it is a macro or a standard cell
Is_in_clk: whether the cell is in clock net
X0, y0, x1, y1: the smallest x-coordinate, the smallest y-coordinate, the largest x-coordinate, and the largest y-coordinate of the cell bounding box
Is_buf, is_inv: whether the cell is a buffer or and inverter
Libcell_name: the libcell name of the cell
Cell_static_power, cell_dynamic power: the static and dynamic power of the cell at the current timing corner

**Libcell_properties:**
Libcell_name: name of the library cell
Func_id: The functional ID of a libcell is a unique identifier assigned based on its functionality. Libcells that perform the same function will share the same functional ID, while those with different functionalities will have distinct IDs.
Libcell_area: area of the libcell
Worst_input_cap: the largest input capacitance
Libcell_leakage: leakage power of the libcell
Fo4_delay: gate delay considering a load capacitance four times larger than the largest input capacitance, while the input slew remains a predefined constant
Libcell_delay_fixed_load: gate delay considering a predefined constant load capacitance and a predefined constant input slew

**Net properties:**
Net_name: name of the net
Net_route_length: total routing path length of the net
Net_steiner_length: total length of the Steiner tree of the net
Fanout: fanout of the net
Total_cap: total capacitance of the net, including wire capacitance and the sink pin capacitance
Net_cap: total wire capacitance
Net_coupling: coupling capacitance
Net_res: wire resistance

**Pin properties:**
Pin_name: name of the pin
X, y: coordinates of the center of the pin
Is_startpoint: whether it is a timing path starting point
Is_endpoint: whether it is a timing path ending point
Dir: direction of the pin, i.e., whether it is an input pin or output pin
Maxcap: capacitance constraint of the pin
Maxtran: transition time constraint of the pin
Num_reachable_endpoint: count of reachable endpoints
Cell_name: name of the corresponding cell
Net_name: name of the corresponding net
Pin_tran: transition time of the pin
Pin_slack: slack of the pin
Pin_rise_arr: arrival time of the pin at rising edge
Pin_fall_arr: arrival time of the pin at falling edgeq
Input_pin_cap: input pin capacitance

**Pin_pin_edge:**
Src: driving pin name
Tar: sink pin name
Is_net: whether the pin2pin edge is a net connection
Arc_delay: timing arc delay from the src to tar