國立臺灣大學電機資訊學院電機工程學系

學士班學生論文

Department of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Bachelor’s Thesis

基於圖神經網路之通用的時序模型萃取方法

GTM: A Generic Graph-Neural-Network-Based Timing Macro Modeling Framework

張凱鈞

Kai-Chun Chang

指導教授：江蕙如 博士

Advisor: Iris Hui-Ru Jiang, Ph.D.

中華民國 111 年 4 月

April 2022

國立臺灣大學學士班學生論文
口試委員會審定書

基於圖神經網路之通用的時序模型萃取方法
GTM: A Generic Graph-Neural-Network-Based Timing Macro Modeling Framework

本論文係張凱鈞君（B07901056）在國立臺灣大學電機工程學系完成之學士班學生論文，於民國 111 年 04 月 06 日承下列考試委員審查通過及口試及格，特此證明

口試委員(3 位)：

江蕙如 （簽名）

（指導教授）

張耀文

林忠緯

系主任： 吳忠幟 （簽名）

（是否須簽章依各院系規定）

Approval Letter from the Oral Examination Committee

Bachelor's Thesis

National Taiwan University

基於圖神經網路之通用的時序模型萃取方法

GTM: A Generic Graph-Neural-Network-Based Timing Macro Modeling Framework

This certifies that this Bachelor's Thesis has been completed by Kai-Chun Chang (with the Student ID No. B07901056) of the Department of Electrical Engineering, National Taiwan University. This Thesis was approved by the examination committee and the author of the Thesis passed the examination administered by the oral examiners on April 6, 2022.

Oral examiners：

Iris Hui-Ru Jiang （Signature）

（Thesis Advisor）

Yaowen Chang

Chung-Wei Lin

Department Chair: （Signature）

(Each college/department will determine whether personal seals are required.)

Acknowledgements

首先，我要感謝我的指導老師江蕙如教授。從確定研究題目、設計演算法、進行實驗到書寫論文，老師總是花費許多心力和我討論，並提供許多富有啟發性的建議，也因此我才能完成這個研究。除了研究外，老師也相當關心我的生涯規劃、給予我很多機會，讓我可以逐步朝學術研究的道路邁進。

接著，我要感謝我的口試委員張耀文教授和林忠緯教授，他們針對研究內容和論文寫作提供的寶貴建議，使這個研究能去蕪存菁、展現出更高的品質。

我也要感謝實驗室的所有學長姐和同學，尤其是李培瑜博士和姜鈞堯學長，他們在我研究過程中提供了很多幫助，讓我能順利完成這篇論文。

再來，我要感謝我的家人們，特別是我的爸媽，他們總是盡其所能地幫助我、鼓勵我，讓我能鼓起勇氣面對研究生涯上的種種挑戰。

最後，我要感謝台大電機系，在這裡我遇到了很多厲害的同學，也從他們身上學到很多東西；也感謝老師們的照顧和啟發。相信這個研究只是我學術路途上的起點，期許自己未來帶著台大電機給我的養分，在學術研究的路途上持續向前邁進！

張凱鈞

國立臺灣大學

2022年6月

基於圖神經網路之通用的時序模型萃取方法

學生：張凱鈞　　　指導教授：江蕙如 博士

國立臺灣大學電機工程學系

摘要

隨著IC設計的複雜度快速地上升，萃取式時序模型(timing macro model)開始被廣泛運用，以實現階層式和平行化的時序分析，進而提升時序分析的效率。萃取式時序模型僅留下對時序分析有重大影響的電路元件接腳，影響輕微者則被捨棄，藉此在壓縮萃取式時序模型大小的同時能夠兼顧時序分析的準確度。因此，產生萃取式時序模型最主要的挑戰就是如何精準辨認出高影響力的電路元件接腳。然而，之前針對萃取式時序模型的研究往往仰賴特定、非一般化的方法，或是要求使用者花費大量心力進行參數調整。因此，本研究提出了一個基於圖神經網路(graph neural network, GNN)的通用萃取式時序模型架構，可以適用在不同的時序延遲模型和多重邊界案例與多重操作模式之時序分析。首先，我們設計了一個量度標準來評估每個電路元件接腳對整體時序分析準確度造成的影響；接著，根據評估的結果，搭配電路的架構，讓圖神經網路來學習、並藉此辨認出高影響力的電路元件接腳。實驗結果顯示，與當前最新的研究相比，可以在保持同樣時序分析準確度的同時、進一步縮小10%的萃取式時序模型大小。此外，以共同路徑悲觀性移除(common path pessimism removal, CPPR)為例，實驗結果證明我們的架構能夠適用在不同的時序分析模式上，展現出高度的一般性。初步研究成果將於電子設計自動化領域旗艦國際會議設計自動化會議(59th Design Automation Conference)發表。

關鍵詞：時序分析、階層式時序分析、萃取式時序模型、圖神經網路、共同路徑悲觀性移除

GTM: A GENERIC GRAPH-NEURAL-NETWORK-BASED TIMING MACRO MODELING FRAMEWORK

Student: Kai-Chun Chang   Advisor: Dr. Iris Hui-Ru Jiang

Department of Electrical Engineering
National Taiwan University

Abstract

Due to rapidly growing design complexity, timing macro modeling has been widely adopted to enable hierarchical and parallel timing analysis. The main challenge of timing macro modeling is to identify timing variant pins for achieving high timing accuracy while keeping a compact model size. To tackle this challenge, prior work applied ad-hoc techniques and threshold setting. In this work, we present a novel and generic timing macro modeling approach based on graph neural networks (GNNs) that is available on various delay models and multi-corner multi-mode (MCMM). A timing sensitivity metric is proposed to precisely evaluate the influence of each pin on the timing accuracy. Based on the timing sensitivity data and the circuit topology, the GNN model can effectively learn and capture timing variant pins. Experimental results show that our GNN-based framework reduces 10% model sizes while preserving the same timing accuracy as the state-of-the-art. Furthermore, taking common path pessimism removal (CPPR) as an example, the generality and applicability of our framework are also validated empirically. The preliminary results have been accepted by the premier conference in Electronic Design Automation, 59th Design Automation Conference.

Keywords: Timing analysis, hierarchical timing analysis, timing macro modeling, interface logic model, common path pessimism removal, multi-corner multi-mode, graph neural networks

Table of Contents

Abstract (Chinese) **v**

Abstract **vii**

List of Figures **xi**

List of Tables **xii**

Chapter 1. Introduction **1**

Chapter 2. Problem Formulation **5**

Chapter 3. Overview of Our Framework **7**

3.1 GNN and Timing Macro Modeling Problem 7

3.2 Our Generic Framework 8

Chapter 4. Timing Sensitivity Data Generation **10**

4.1 Timing Sensitivity (TS) 10

4.2 Insensitive Pins Filtering 11

Chapter 5. GNN-Based Timing Macro Modeling **15**

5.1 GNN Model Training and Prediction 15

5.2 Timing Macro Model Generation 16

5.3 Flexibility and Generality of Our Framework 17

Chapter 6. Timing Macro Modeling for Multi-Corner Multi-Mode **19**

6.1 Multi-Corner Multi-Mode (MCMM) Timing Analysis 19

6.2 Timing Macro Modeling Covering All Corners 20

Chapter 7. Experimental Results **24**

Chapter 8. Conclusions **32**

Bibliography **33**

Publication List **37**

List of Figures

1.1 Hierarchical and parallel timing analysis along with timing macro modeling. The "core" block is analyzed once, and the corresponding timing macro model is reused to all the "core" blocks [5]. 2

2.1 Timing macro modeling and model accuracy evaluation flow. 6

3.1 The analogy between GNN aggregation and timing propagation. Timing values including slew, arrival time, and required arrival time are propagated through edges (blue and green arrows). On the other hand, node features of layer l, $h_i^{(l)}$, are aggregated through edges and transformed into node features of layer $l+1$, $h_i^{(l+1)}$ (red arrows). . . . 8

3.2 Overview of our framework. 9

4.1 Timing sensitivity evaluation flow. 11

4.2 Timing sensitivity distribution of *fft_ispd*. 13

4.3 Slew difference and shielding effect. 13

4.4 Timing sensitivity training data generation flow. 14

5.1 Timing macro model generation. 17

6.1 The default flow to generate timing macro models for all the corners. 20

6.2 Our framework to generate timing macro models for all the corners. . 22

6.3 An example of our timing macro modeling framework for all corners. 23

7.1 Separated TS distribution based on the insensitive pins filtering. . . . 27

List of Tables

5.1 Training features. The first eight features are basic features, while the last feature is a dedicated feature for CPPR mode. 16

7.1 Testing data statistics. 25

7.2 Experimental results on TAU 2016 [5] and TAU 2017 [1] benchmarks with CPPR. For the avg. and the max error, we adopt the absolute value of difference between the result of macro model and the one of full netlist. If the results of macro model are more optimistic, the difference is further weighted by 2. For the model file size, we adopt the size of the library for late timing. Difference 1 and ratio 1 are compared with iTimerM [13]. Difference 2 and ratio 2 are compared with [12]. Difference = compared result - our result. Ratio = compared result / our result. Note that [12] is only evaluated on TAU 2016 benchmark in their work. 28

7.3 Experimental results with and without CPPR-dedicated features. . . 29

7.4 Experimental results on TAU 2017 benchmark without CPPR. For the avg. and the max error, we adopt the absolute value of difference between the result of macro model and the one of full netlist. If the results of macro model are more optimistic, the difference is further weighted by 2. For the model file size, we adopt the size of the library for late timing. Difference 1 and ratio 1 are compared with iTimerM [13]. Difference 2 and ratio 2 are compared with ATM [10]. Difference = compared result - our result. Ratio = compared result / our result. We additionally include the circuit *mgc_matrix_mult_iccad* to evaluate since ATM [10] also adopts it as one test case. 30

7.5 Validation on insensitive pins filtering. 31

Chapter 1

Introduction

During the IC design flow, static timing analysis (STA) is regarded as a crucial and essential step to achieve timing closure. As the evolution of the IC industry, the design complexity grows rapidly, and timing analysis has thus become a bottleneck due to its high computational cost. To improve the efficiency of timing analysis, hierarchical and parallel timing analysis has been widely adopted. During hierarchical and parallel timing analysis, a large design is partitioned into several blocks, each block is then analyzed once, and a corresponding timing macro model is generated. The macro model could be reused for duplicate blocks in the following analysis, thus expediting the whole process while preserving the quality. (see Figure 1.1.)

Several timing macro modeling approaches have been proposed in literature. Interface logic models (ILMs) and extracted timing models (ETMs) [2] are two pioneering paradigms. ILM contains all the interface logic while eliminating register-to-register logic, and ETM builds context-independent timing arcs between input and output ports. The later works start from either of the two paradigms and attempt to improve the timing accuracy or model size. ILM-based approaches aim to preserve high timing accuracy, but they often generate larger models. On the other hand, ETM-based approaches generate relatively smaller models at the cost of high timing accuracy loss. Moreover, it is not trivial to extend ETM-based approaches to handle common path pessimism removal (CPPR), which is commonly considered in

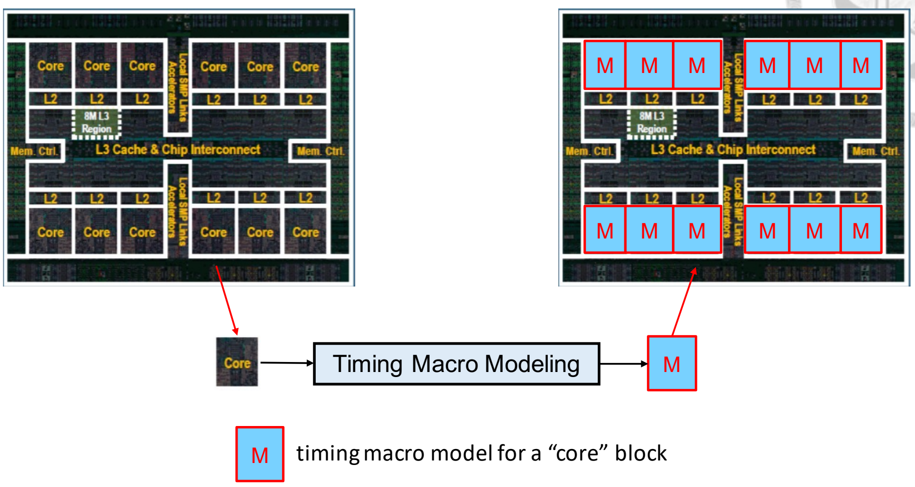

Core
L2
8M L3 Region
L3 Cache & Chip Interconnect
Mem. Ctrl.
Local SMP Links Accelerators
Timing Macro Modeling
M
timing macro model for a "core" block

Figure 1.1: Hierarchical and parallel timing analysis along with timing macro modeling. The "core" block is analyzed once, and the corresponding timing macro model is reused to all the "core" blocks [5].

modern design. For ILM-based approaches, LibAbs [11] and its following work [12] perform tree-based graph reduction, preserve roots and leaves of maximal in-trees, and construct primary output segments for output load. iTimerM [13] propagates minimum/maximum slew values through the timing graphs, and pins with slew range exceeding a user-defined tolerance are preserved. ATM [10] is an ETM-based approach; it marks those pins with slew range exceeding a threshold as dirty, selects checkpoints from dirty pins, and builds groups as well as timing arcs accordingly.

The main challenge of timing macro modeling is to identify timing variant pins for achieving high timing accuracy while keeping a compact model size. First, to tackle this challenge, previous work adopts some heuristic techniques during their timing macro modeling procedure, which may cause degradation on the solution quality. For instance, LibAbs [11, 12] applies in-tree and out-tree graph reductions alternatively, based on the observation on the timing arc forms of cells or nets. Second, some works need to set a threshold for variant pins identification, which requires considerable engineering effort, and the same threshold may not be applicable for various circuit designs. For example, iTimerM [13] uses a threshold to separate the variant regions with the constant region, and ATM [10] uses a threshold to determine which pins are dirty. Lastly, for advanced node timing analysis models or modes such as CPPR, existing methods have to design specific algorithms for different timing analysis models to meet the corresponding requirements, which may be time-consuming and limited.

Therefore, there is still room for improvement. Recently, graph-learning-based methods have been validated to outperform the traditional heuristic-based approaches on multiple EDA problems on graphs, such as tier partitioning in 3D ICs [16], predictions on parasitics and device parameters [21], and multiple patterning lithography decomposition (MPLD) [15], etc. To overcome the deficiencies of prior work on timing macro modeling, we introduce graph neural networks (GNN) to learn the timing variant pins from the circuit topology and timing propagation properties. In this work, we first design a timing sensitivity metric that can capture the influence of each pin on the overall timing accuracy, and generate the training data for GNN models accordingly. Then, due to the applicability of GNN on the timing macro modeling problem, the timing properties of circuit pins could be learnt effectively. Eventually, we establish a flexible and general GNN-based timing macro modeling framework that can achieve better solution quality than previous work.

The main contributions of this work are summarized below:

- We take a brand new graph-learning-based approach to the timing macro modeling problem, in view of the high applicability of GNN on the problem.
- We propose a timing sensitivity metric that can evaluate the timing criticality of each circuit pin accurately. The metric is then used to generate training data for GNN models.

- We propose a flexible timing macro modeling with GNN framework which is available on general designs, as we only include small designs during our training phase while our framework could achieve superior quality on large designs.

- Our framework can easily be applied to different advanced node timing analyses. We use CPPR as an example, while the same strategy could be extended to other analyses such as advanced on-chip-variation (AOCV), parametric on-chip-variation (POCV), and composite current source (CCS) model. We demonstrate how to generalize our framework to multi-corner multi-mode (MCMM).

As an ILM-based approach, experimental results show that our framework achieves the best timing accuracy in comparison with state-of-the-art works. Moreover, we improve the model size by 10% than iTimerM [13], the most accurate state-of-the-art work. Besides, our framework generates high-quality solutions no matter whether the CPPR mode is turned on, which implies the generality and applicability of our framework.

The remainder of this thesis is organized as follows: Chapter 2 formulates the timing macro modeling problem. Chapter 3 introduces GNN along with its applicability to the problem and illustrates our framework. Chapter 4 details our timing sensitivity metric as well as the data generation flow. Chapter 5 details the GNN model training, the timing macro model generation, along with the generality of our framework. Chapter 6 introduces the MCMM timing analysis and how to extend our framework for various corners. Chapter 7 shows experimental results. Finally, Chapter 8 concludes this work.

Chapter 2

Problem Formulation

In this work, we follow the problem formulation from TAU 2016 and 2017 contests [1,5], which is also adopted by most previous work. The **timing macro modeling** problem can be defined as follows:

Given a circuit design with its gate-level netlist and net parasitics, the early and late cell libraries, and the boundary timing information (including slew and arrival time of primary inputs, and output load and required arrival time of primary outputs), the goal is to generate a timing macro model that encapsulates the timing behaviors of the design.

The generated timing macro model is evaluated by its model accuracy, model size, model generation performance, and model usage performance, where model accuracy is validated by comparing timing analysis results using our timing macro model and the original flat design, as shown in Figure 2.1. We adopt iTimerM [13] as our reference timer, and the results are also aligned with OpenTimer [6].

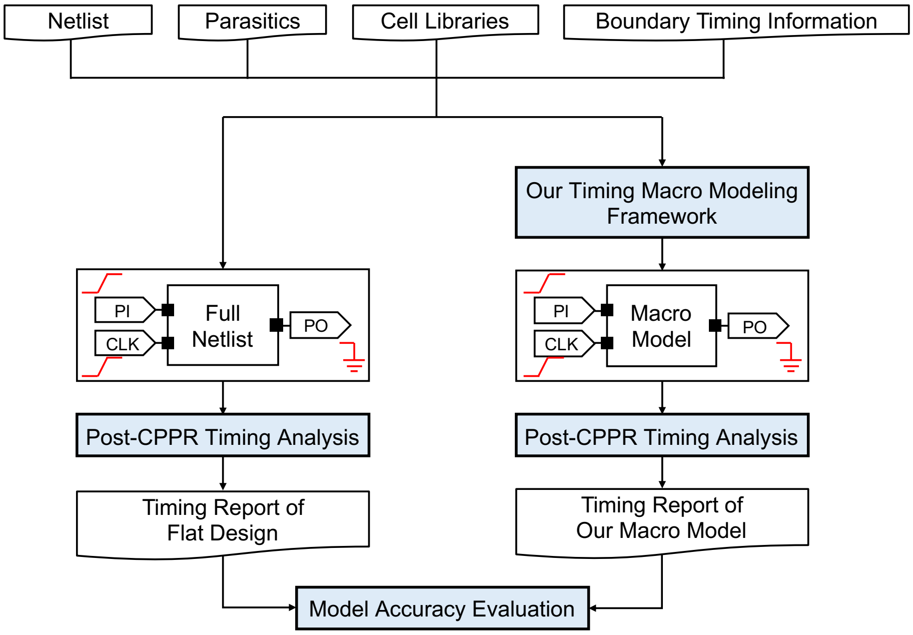

Netlist
Parasitics
Cell Libraries
Boundary Timing Information
Our Timing Macro Modeling Framework
PI
CLK
Full Netlist
PO
PI
CLK
Macro Model
PO
Post-CPPR Timing Analysis
Post-CPPR Timing Analysis
Timing Report of Flat Design
Timing Report of Our Macro Model
Model Accuracy Evaluation

Figure 2.1: Timing macro modeling and model accuracy evaluation flow.

Chapter 3

Overview of Our Framework

3.1 GNN and Timing Macro Modeling Problem

Encouraged by the success of deep learning paradigms on a variety of tasks, graph neural networks (GNN) have been developed to apply deep learning methods to graph data [17, 22]. In a typical GNN scheme, node information is aggregated and transformed between neighbors recursively. After several neural network layers, a high-level representation of each node is extracted, which encapsulates the features and structures of the node's neighborhood [16, 21].

There are several reasons that GNN is suitable for the timing macro modeling problem. First, the evaluation of timing criticality on circuit pins is usually challenging for heuristic-based methods. Nevertheless, graph-learning-based methods could capture implicit properties of circuit pins and thus evaluating timing importance more precisely. Second, the aggregation of node attributes in GNN is similar to the propagation of timing values on timing graphs, as shown in Figure 3.1. Consequently, the timing properties of circuit pins could be captured and learned by GNN models smoothly. Third, due to the information exchange mechanism in GNN, the final representations of adjacent nodes tend to become similar. This property is desired in timing macro modeling since neighbor pins are usually of comparable degrees of timing criticality. Lastly, it is natural to represent circuit netlists by

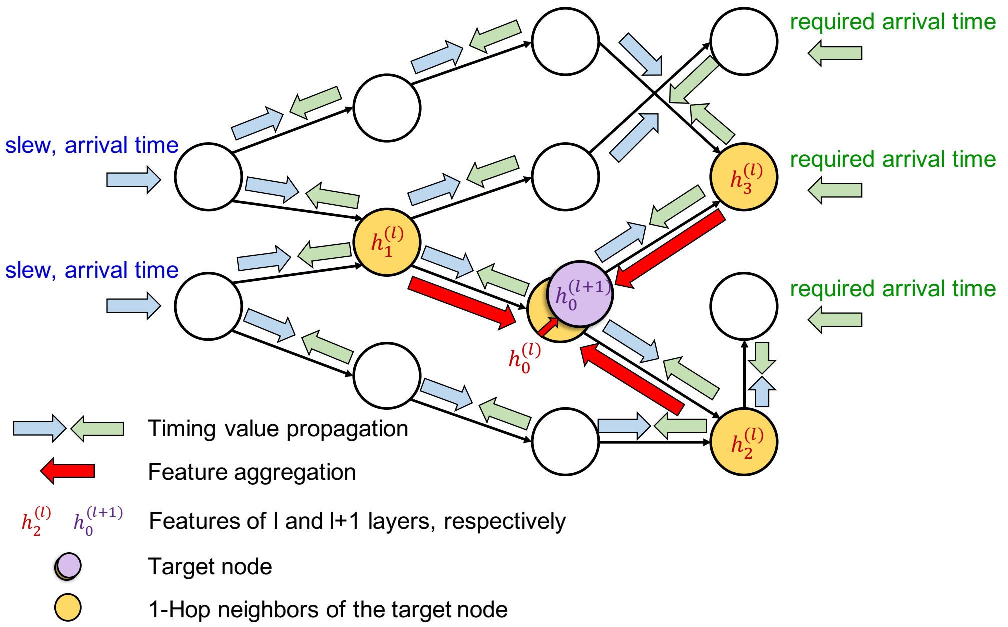


Figure 3.1: The analogy between GNN aggregation and timing propagation. Timing values including slew, arrival time, and required arrival time are propagated through edges (blue and green arrows). On the other hand, node features of layer l, $h_i^{(l)}$, are aggregated through edges and transformed into node features of layer $l+1$, $h_i^{(l+1)}$ (red arrows).

graphs, and thus GNNs could be easily embedded into the timing macro modeling framework.

3.2 Our Generic Framework

Figure 3.2 illustrates the proposed timing macro modeling framework. In the first stage, the *timing sensitivity* of each circuit pin is evaluated to reflect the influence of each pin on the overall timing accuracy. Then, the training data is generated accordingly. In the second stage, we adopt GNN models to learn the properties of circuit designs and predict the timing sensitivities of testing data. Finally, starting from the interface logic netlist (ILM), timing macro models are generated based on our timing sensitivities prediction. Different from previous work,

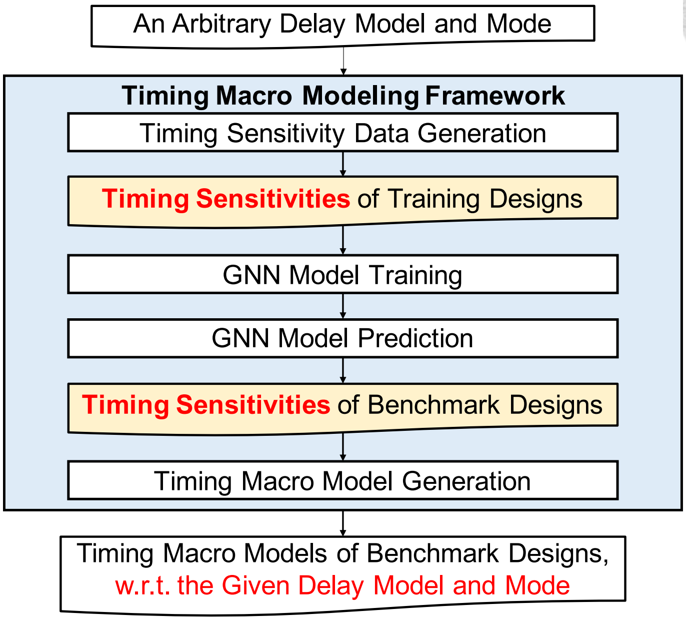

An Arbitrary Delay Model and Mode
Timing Macro Modeling Framework
Timing Sensitivity Data Generation
Timing Sensitivities of Training Designs
GNN Model Training
GNN Model Prediction
Timing Sensitivities of Benchmark Designs
Timing Macro Model Generation
Timing Macro Models of Benchmark Designs,
w.r.t. the Given Delay Model and Mode

Figure 3.2: Overview of our framework.

which mainly focuses on non-linear delay model (NLDM), our framework could also be applied to other advanced node timing analysis models such as CCS, AOCV, and POCV, or different timing modes like CPPR. The generality of our framework comes from the fact that timing sensitivities could be adaptively evaluated depending on the given timing delay model. Moreover, the GNN models could effortlessly capture the corresponding timing properties.

Chapter 4

Timing Sensitivity Data Generation

4.1 Timing Sensitivity (TS)

In order to generate a high-quality timing macro model, we need to precisely evaluate the influence of each circuit pin on the timing accuracy of the whole design. Then, pins with subtle influences could be waived to reduce model size and meanwhile the timing accuracy will not be degraded.

Figure 4.1 shows how we evaluate the timing sensitivity (TS) of each pin. Given the input circuit graph, we first randomly generate several [1] sets of boundary timing constraints. Between each set of timing constraints, incremental timing analysis [14] is performed on the ILM and the results are stored as references. In the timing sensitivity evaluation stage, we remove a pin from the circuit each time. After the removal, we perform timing propagation based on each set of boundary timing constraints generated and compute the differences between the current and the reference timing values (including slew, arrival time (at), required arrival time (rat), and slack) at the boundary pins. Finally, TS of a pin (for convenience, denoted as A in the following discussion) is set as the average of timing value differences under the different timing constraints. Equations (4.1) and (4.2) define the TS of pin

[1] We generate ten sets of boundary timing constraints in our experiments. Using more sets, e.g. twenty, identifies extremely few extra sensitive pins. Thus, ten sets of constraints are sufficient to cover the given operating conditions and find almost all sensitive pins.

A, where C denotes the collection of generated boundary timing constraints, and $slew^c_{P,before}$ (*resp.* $slew^c_{P,after}$) denotes the slew value of a boundary pin P under the timing constraint c before (*resp.* after) pin A's removal. The definitions of Δat^c_A, Δrat^c_A, and $\Delta slack^c_A$ are similar to that of $\Delta slew^c_A$ (i.e., Equation (4.2)).

$$TS_A = AVG_{c \in C}(AVG(\Delta slew^c_A, \Delta at^c_A, \Delta rat^c_A, \Delta slack^c_A)) \tag{4.1}$$

$$\Delta slew^c_A = \frac{1}{|PI \cup PO|} \Sigma_{P \in PI \cup PO} \frac{slew^c_{P,after} - slew^c_{P,before}}{slew^c_{P,before}} \tag{4.2}$$

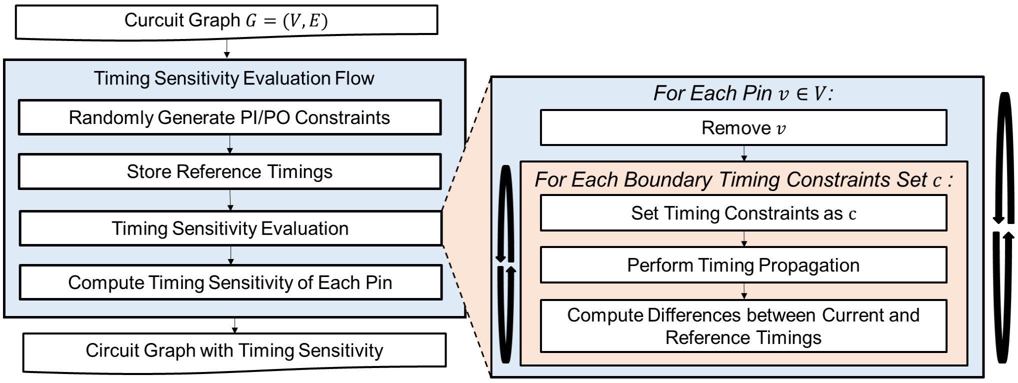

Curcuit Graph $G = (V, E)$
Timing Sensitivity Evaluation Flow
Randomly Generate PI/PO Constraints
Store Reference Timings
Timing Sensitivity Evaluation
Compute Timing Sensitivity of Each Pin
Circuit Graph with Timing Sensitivity
For Each Pin $v \in V$:
Remove v
For Each Boundary Timing Constraints Set c :
Set Timing Constraints as c
Perform Timing Propagation
Compute Differences between Current and Reference Timings

Figure 4.1: Timing sensitivity evaluation flow.

4.2 Insensitive Pins Filtering

Although the TS evaluation flow could accurately compute the influence of each pin on the overall timing accuracy, running the flow for all the pins is time-consuming as we need to perform timing propagation once in each iteration. To enhance the efficiency, we first observe that the majority of the pins have extremely small or even zero TS. It is due to the nature of timing graph that most of the pins have subtle influences on the overall timing accuracy. For example, the TS distribution of circuit *fft_ispd* is shown in Figure 4.2, where 70% pins have zero TS, while only few pins have large TS. Therefore, if we can find a rapid screening method to filter the insensitive pins first, we could perform TS evaluation flow on the potential critical pins only.

Timing value difference propagation is a suitable method for insensitive pins filtering. At each primary input (PI) or primary output (PO) port, two timing values, $t_{\min}$ and $t_{\max}$, are set up. We then propagate the timing values through the design and monitor the difference between the two timing values at each pin. According to the shielding effect, as shown in Figure 4.3, the difference decays after several levels, and pins with small difference tend to have subtle influence on the overall timing accuracy. Inspired by previous works [10, 13], we choose slew to propagate from each PI. After the propagation, the slew difference (SD) at each pin is standardized, and pins with SD less than a threshold is filtered out. As mentioned in Chapter 1, thresholds to distinguish crucial pins are also adopted in some previous works, where the thresholds must be tuned delicately to obtain favorable results. In contrast, the threshold here is not required to be precise since it only helps reduce the number of pins to be evaluated, and thus the quality of generated timing macro models from our framework is independent of the threshold. Actually, we have never tuned the threshold value during our experiments. In addition, last stage pins and pins connected to some output net are also remained for output load variant.

After the insensitive pins filtering, more than 88% pins are filtered out from the TS evaluation flow, which implies the flow becomes almost 10 times faster. As a result, the training data could be generated efficiently. Figure 4.4 illustrates the whole training data generation flow.

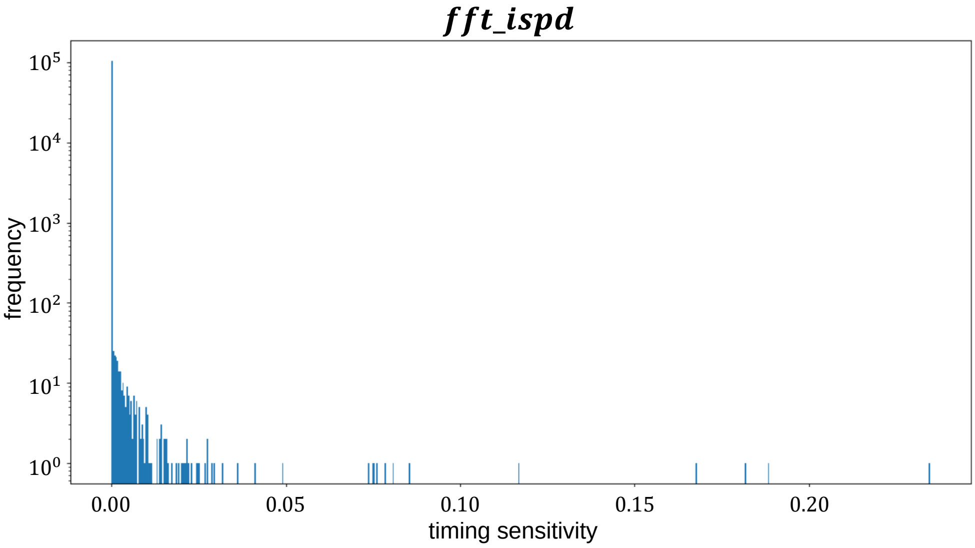


Figure 4.2: Timing sensitivity distribution of *fft_ispd*.

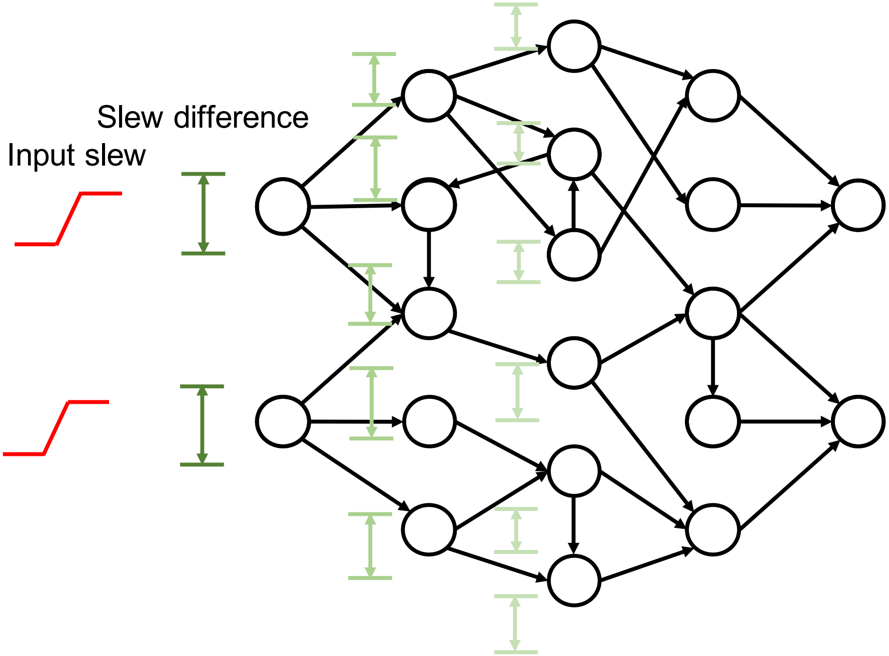


Figure 4.3: Slew difference and shielding effect.

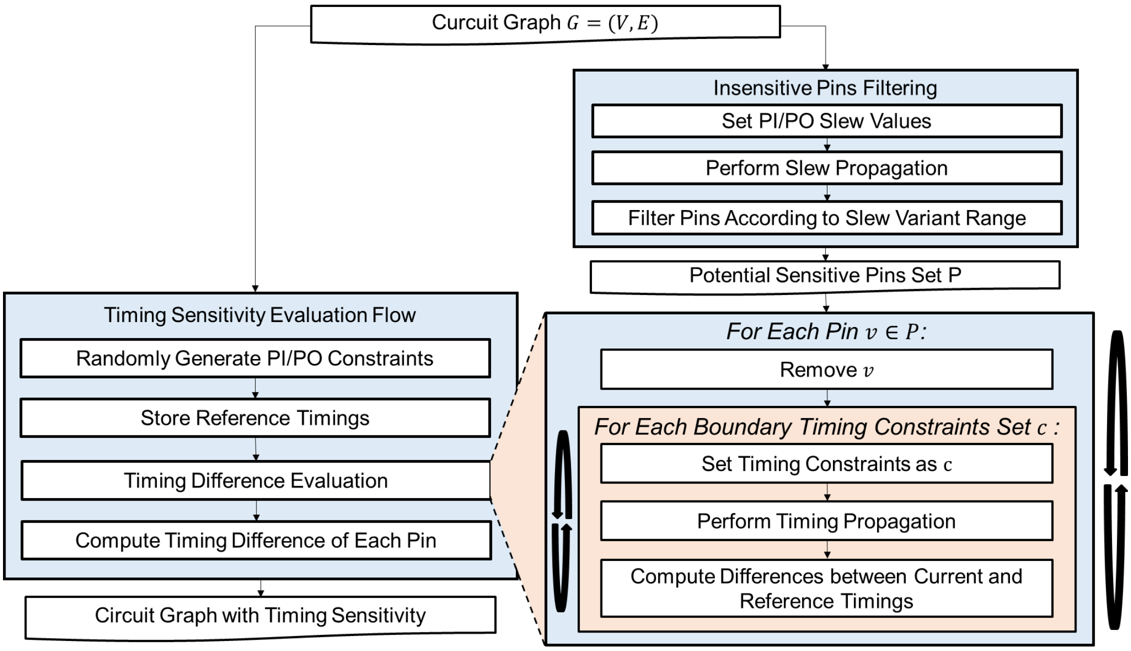

Curcuit Graph $G = (V, E)$
Insensitive Pins Filtering
Set PI/PO Slew Values
Perform Slew Propagation
Filter Pins According to Slew Variant Range
Potential Sensitive Pins Set P
Timing Sensitivity Evaluation Flow
Randomly Generate PI/PO Constraints
Store Reference Timings
Timing Difference Evaluation
Compute Timing Difference of Each Pin
Circuit Graph with Timing Sensitivity
For Each Pin $v \in P$:
Remove v
For Each Boundary Timing Constraints Set c :
Set Timing Constraints as c
Perform Timing Propagation
Compute Differences between Current and Reference Timings

Figure 4.4: Timing sensitivity training data generation flow.

Chapter 5

GNN-Based Timing Macro Modeling

5.1 GNN Model Training and Prediction

With the timing sensitivity training data, GNN models could learn and predict accordingly. In this work, we adopt GraphSAGE [4] as our main GNN engine. For node v, it first aggregates the node features from its neighborhood $\mathcal{N}(v)$ through Equation (5.1), then Equation (5.2) concatenates and encodes the representation of node v with the aggregated vector. In the experiments, only four rounds of aggregations and encodings are performed, as the timing property of a node is mostly influenced by its neighborhood. Other existing GNN models such as GCN [9] or even self-defined GNN models could also be embedded with our framework.

$$h^k_{\mathcal{N}(v)} \longleftarrow AGGREGATE_k(h^{k-1}_u, \forall u \in \mathcal{N}(v)) \tag{5.1}$$

$$h^k_v \longleftarrow \sigma(W^k \cdot CONCAT(h^{k-1}_v, h^k_{\mathcal{N}(v)})) \tag{5.2}$$

As we treat the GNN prediction as a classification problem for the most part, we need to convert the training labels of pins to $\{0, 1\}$. We set the label of a pin to 1 if and only if its TS is not zero. The conversion is reasonable because a non-zero TS implies the corresponding pin may have some influence on the overall timing accuracy. In addition, for CPPR mode, labels of multiple-fan-out pins of clock networks are also set to 1, since previous works [7,14] point out that this kind

Table 5.1: Training features. The first eight features are basic features, while the last feature is a dedicated feature for CPPR mode.

Feature	Description
level_from_PI	The minimum level from a PI to the pin
level_to_PO	The min. level from the pin to a PO
is_last_stage_fanout	If the pin is the fanout of a last stage pin
is_last_stage	If the pin is the last stage of the timing graph
is_first_stage	If the pin is the first stage of the timing graph
out_degree	The number of output edges of the pin
is_clock_network	If the pin belongs to clock network
is_ff_clock	If the pin is the clock pin of a flip-flop
is_CPPR	If the pin is crucial for CPPR

of pins may be the common points of the clock paths of sequential elements pair, which is essential for CPPR calculation.

The training features are listed in Table 5.1. The features are all basic circuit properties which could be extracted within linear time. Features beginning with *"is"* are of $\{0, 1\}$ Boolean values. For *level_from_PI*, *level_to_PO*, and *out_degree*, the values are normalized to $[0, 1]$ so that each feature have the same level of influences.

5.2 Timing Macro Model Generation

Figure 5.1 details the timing macro model generation stage. First, we capture the interface logic netlist to construct ILM. Second, based on the predictions from GNN models, we perform serial and parallel mergings on timing edges iteratively to remove insensitive pins. For serial merging, the delay of a merged edge is the sum of the original ones, while the slew inherits the last edge. For parallel merging, delay or slew is the minimum (resp. maximum) of the original edge values in the early (resp. late) mode. Afterward, we apply the lookup table index selection method proposed in [13], where indices that minimize the interpolation timing error are selected. Lastly, the timing macro model is generated.

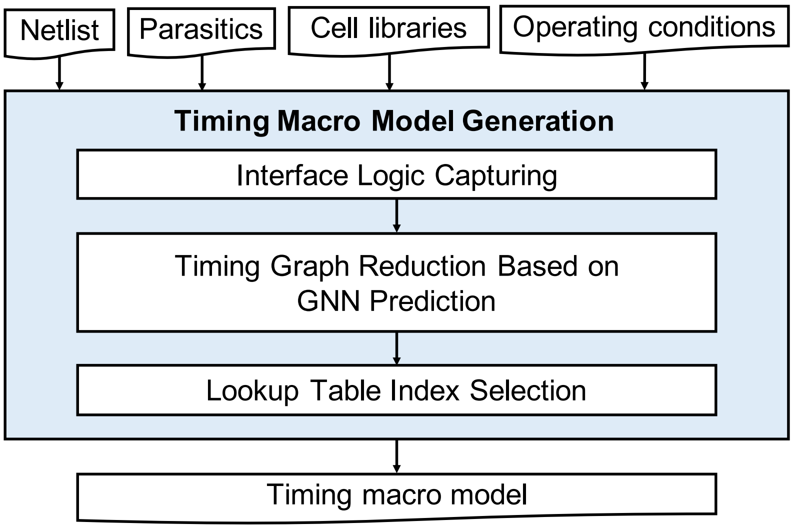

Netlist
Parasitics
Cell libraries
Operating conditions
Timing Macro Model Generation
Interface Logic Capturing
Timing Graph Reduction Based on GNN Prediction
Lookup Table Index Selection
Timing macro model

Figure 5.1: Timing macro model generation.

5.3 Flexibility and Generality of Our Framework

As mentioned in Chapter 3, our framework can be applied to different timing analysis models or modes. The reason is that the timing-sensitivity-based training labels, the basic features, and the circuit netlist structure are enough to reflect the importance of each pin, either in an explicit or implicit manner. However, to help GNN model training, we may leverage domain knowledge for each specific timing model or mode. Take CPPR as an example. As we know, multiple-fan-out pins of clock networks are crucial for CPPR calculation. Thus, we could add a dedicated training feature for CPPR to indicate this kind of pins, called is_CPPR. Before adding the special feature into GNN model training, the other features such as out_degree and $is\_clock\_network$ along with the timing sensitivities could implicitly indicate multiple fan-out pins of clock networks; therefore, we could already obtain high-quality timing macro models. After including the dedicated feature to explicitly identify this kind of pins, we could further enhance the results, and the training process becomes more efficient. The technique could be applied to other timing models as well.

In addition, our training designs are of 10^4 to 10^6 pins, while testing designs mostly have millions of pins. However, as experimental results show, our framework could capture the timing properties from small designs and obtain good results on large designs. It implies that our framework could be directly used to generate timing macro models for general designs.

Lastly, the GNN prediction in our framework could also be treated as a regression problem, i.e., timing sensitivities are set as training labels directly, and the framework could not only learn which pins are critical for timing accuracy, but also capture the relative criticality between pins.

Chapter 6

Timing Macro Modeling for Multi-Corner Multi-Mode

6.1 Multi-Corner Multi-Mode (MCMM) Timing Analysis

In today's advanced technology, different PVT corners (a combination of process, voltage, and temperature parameters) and operation modes (a set of timing constraints, supplying voltage, etc.) result in divergent timing analysis results. Ideally, STA should be performed under all the corners and modes to guarantee that timing constraints are met and enhance the design quality. However, as the number of corners and modes grows exponentially in modern processes, the exhaustive approach becomes impractical. To tackle the complexity of MCMM timing analysis, several works [3,18–20] attempt to find the worst-delay corner or an upper bound for all the corner delays, so that the checking of timing violations could be performed in practical time. Recently, a learning-based approach [8] leverages the timing analysis results of known corners to predict those of unobserved corners. Thus, analysis results of all the corners could be obtained while only a limited number of corners and paths are required to be analyzed.

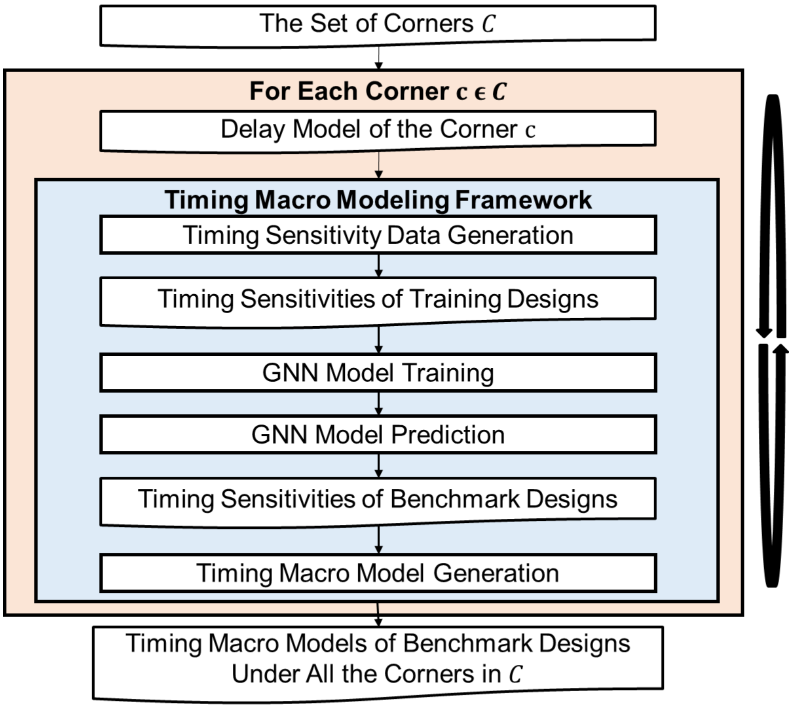

The Set of Corners C
For Each Corner c ϵ C
Delay Model of the Corner c
Timing Macro Modeling Framework
Timing Sensitivity Data Generation
Timing Sensitivities of Training Designs
GNN Model Training
GNN Model Prediction
Timing Sensitivities of Benchmark Designs
Timing Macro Model Generation
Timing Macro Models of Benchmark Designs Under All the Corners in C

Figure 6.1: The default flow to generate timing macro models for all the corners.

6.2 Timing Macro Modeling Covering All Corners

To the best of our knowledge, however, no previous work has dealt with MCMM for the generation of timing macro models. As described in Chapter 5, our framework is available for various timing analysis models or modes. Thus, intuitively, to generate timing macro models for all the corners, we could run our framework once for each corner as shown in Figure 6.1. Similar to the exhaustive STA, the method is impractical since timing sensitivity data generation and GNN model training are time-consuming.

To generate macro models for all the corners efficiently, we observe that a linear model is often assumed for the relation between delay/slew and the parameters in previous works [3, 19] as Equation (6.1) shows, where H is the real value of delay/slew, α_0 is the nominal value of delay/slew, X_i's are the parameters that are normalized to $[-1,1]$, and α_i's are the sensitivities of H to the corresponding parameters. In addition, Kahng et al. [8] point out the correlation between path delays of different corners.

$$H = \alpha_0 + \alpha_1 X_1 + \alpha_2 X_2 + ... + \alpha_n X_n \tag{6.1}$$

Inspired by these ideas, we propose a two-stage timing macro modeling framework for all corners (see Figure 6.2). Firstly, all the corners (C) are divided into two disjoint sets C_{obs} and C_{unobs}. Then, only the corners in C_{obs} are fed into the first stage. After the timing sensitivities of benchmark designs under corners from C_{obs} are extracted, we adopt deep neural networks (DNN) to predict the timing sensitivities of corners from C_{unobs} in the second stage, in view of the linear dependence of timing values to process parameters and the correlation between timing analysis results of different corners. After that, timing macro models for corners in C can be generated accordingly. Finally, the macro models are further merged into a single timing macro model based on the similarities of timing values among different corners.

An example is illustrated in Figure 6.3. In the first stage, only corners in C_{obs} (Corners 1, 2, and 3) undergo the timing macro modeling framework, and the timing sensitivities of pins under each corner are determined (red for sensitive pins and white for insensitive pins). On the other hand, the sensitivities of pins under corners in C_{unobs} remain unknown. In the second stage, for each pin in the timing graph, a DNN model is trained with the generated timing sensitivities (as training labels) and the parameters (as training features) of the observed corners. Then, the timing sensitivities of pins under the unobserved corners could be inferred by the DNN models and the corresponding parameters.

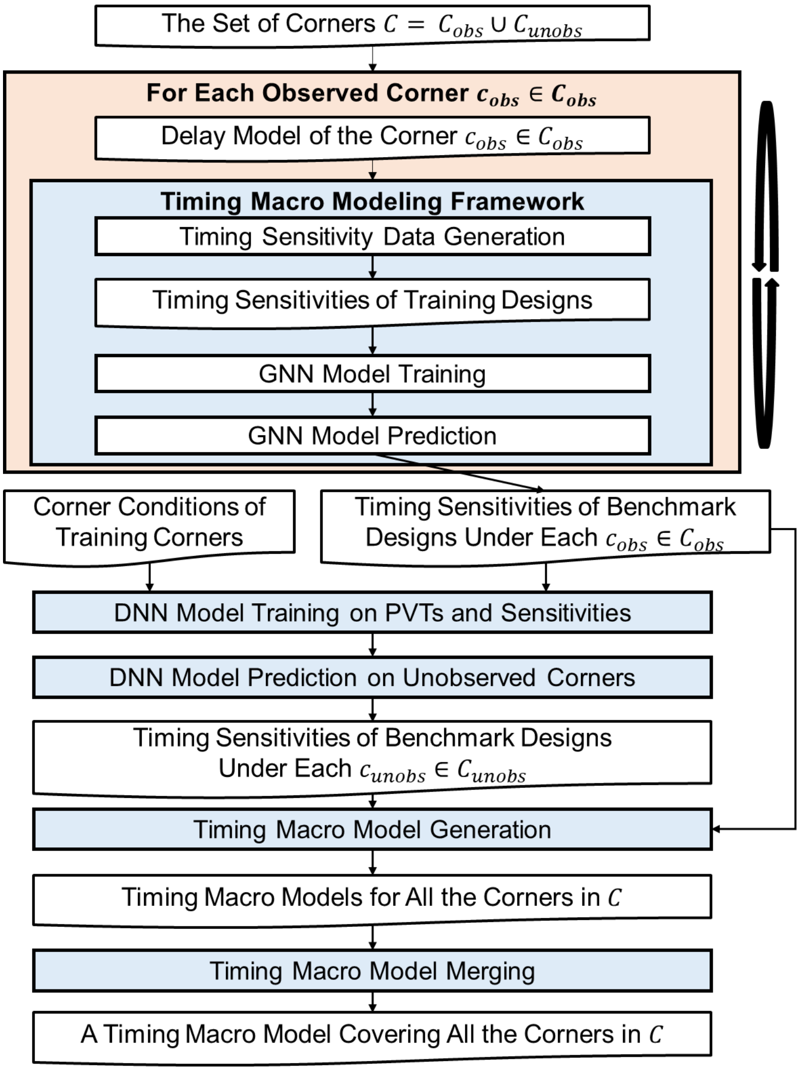
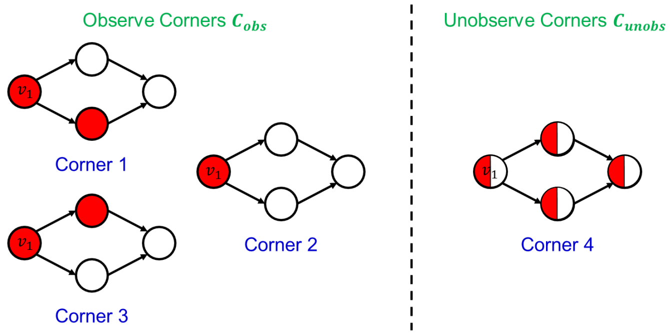
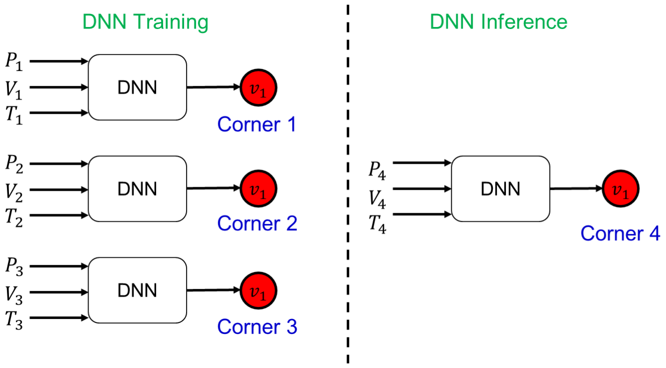

The Set of Corners $C = C_{obs} \cup C_{unobs}$
For Each Observed Corner $c_{obs} \in C_{obs}$
Delay Model of the Corner $c_{obs} \in C_{obs}$
Timing Macro Modeling Framework
Timing Sensitivity Data Generation
Timing Sensitivities of Training Designs
GNN Model Training
GNN Model Prediction
Corner Conditions of Training Corners
Timing Sensitivities of Benchmark Designs Under Each $c_{obs} \in C_{obs}$
DNN Model Training on PVTs and Sensitivities
DNN Model Prediction on Unobserved Corners
Timing Sensitivities of Benchmark Designs Under Each $c_{unobs} \in C_{unobs}$
Timing Macro Model Generation
Timing Macro Models for All the Corners in C
Timing Macro Model Merging
A Timing Macro Model Covering All the Corners in C

Figure 6.2: Our framework to generate timing macro models for all the corners.



(a) The first stage.



(b) The second stage.

Figure 6.3: An example of our timing macro modeling framework for all corners.

Chapter 7

Experimental Results

In our framework, the timing sensitivity data generation and timing macro model generation are implemented in the C++ programming language, while the GNN model training and prediction are implemented in Python3 programming language with the PyTorch library. The experiments are conducted on a Linux workstation with 3.7 GHz CPU, 192 GB RAM, and a NVIDIA RTX 3090 GPU. Our framework is validated on the benchmark suite released by TAU 2016 and TAU 2017 contests [1,5]. Table 7.1 list the statistics of the benchmarks.

Table 7.2 shows the results on TAU 2016 [5] and TAU 2017 [1] benchmarks considering CPPR and the comparisons with two state-of-the-art ILM-based works iTimerM [13] and [12]. Among all the criteria, max error and model file size are viewed as the most crucial ones. Our framework achieves extremely high timing accuracy as all the max errors are less than 0.1ps, which is same as iTimerM [13] and 9 times better than [12]. As for model file size, our result is about 10% smaller than iTimerM [13] and 45% smaller than [12]. To summarize, our framework preserves the highest timing accuracy in terms of max errors among the state-of-the-art works, while further improving the model size by 10% than the same-accuracy-level work. Our framework also achieve similar or even better results in terms of model generation performance and model usage performance. The average errors of our framework are slightly higher than those of iTimerM [13]; however, the difference

Table 7.1: Testing data statistics.

Design	#Pins	#Cells	#Nets
mgc_edit_dist_iccad_eval	581319	224113	224101
vga_lcd_iccad_eval	768050	286597	286498
leon3mp_iccad_eval	4167632	1534489	1534410
netcard_iccad_eval	4458141	1630171	1630161
leon2_iccad_eval	5179094	1892757	1892672
mgc_edit_dist_iccad	450354	164266	164254
vga_lcd_iccad	679258	259251	259152
leon3mp_iccad	3376832	1248058	1247979
netcard_iccad	3999174	1498565	1498555
leon2_iccad	4328255	1617069	1616984
mgc_matrix_mult_iccad	492568	176084	174484

is only a few femtoseconds and thus can be neglected. Although the max errors of previous works are also fractions of picosecond, timing errors may be propagated and accumulated to a larger amount since timing macro models are often cascaded during hierarchical and parallel timing analysis in industrial applications. As macro models are used more frequently in larger designs, our accuracy improvement becomes more significant.

As mentioned in Chapter 5, we could leverage the domain knowledge to help GNN model training for different timing models or modes. We use CPPR as an example, and the result is shown in Table 7.3. We adopt the results of iTimerM [13] as the baseline and calculate the differences and ratios as described in Table 7.2. Before adding the CPPR-dedicated feature (i.e., *is_CPPR*), our framework could already achieve the same timing accuracy as iTimerM [13] while reducing the model size by 6%. After the *is_CPPR* feature is included, our framework still preserves the same timing accuracy while improving the model size by 10%. The result tells that our framework could achieve superior quality with only the basic features, while the dedicated features could capture the timing properties of designs more precisely.

Table 7.4 displays the results on the TAU 2017 [1] benchmark without CPPR. Our results are compared with the ILM-based work iTimerM [13] and the ETM-based work ATM [10]. In comparison with ATM [10], our framework achieves 9 times better max error and 25 times better average error, but it suffers from a larger model size. It is as our expectation since our framework is ILM-based while ATM [10] is ETM-based. Besides, we also achieve 17 times faster model generation runtime than ATM [10]. As for the ILM-based work iTimerM [13], we preserve the same timing accuracy while improving the model size by 9%. The result validates the applicability and generality of our framework on different timing modes (CPPR on and CPPR off), and it may be further inferred to various timing delay models and modes.

As mentioned in Chapter 4, the goal of the insensitive pins filtering is to exclude non-critical pins rapidly, under the premise that the timing accuracy is not degraded. Figure 7.1 shows the timing sensitivities of pins in the training design *systemcaes*. TS of pins that are filtered out are shown in the left histogram, and those of the potential sensitive pins are shown in the right histogram. It can be seen that a majority of filtered pins indeed have zero TS, while many remained pins have non-zero TS. It validates the consistency between the insensitive pins filtering and the TS evaluation, which implies the insensitive pins filtering is suitable for accelerating the training data generation flow. To further ensure the timing accuracy is not degraded by the insensitive pins filtering, we conduct an experiment in which the training labels of all the remained pins after the insensitive pins filtering are set to 1. The result is shown in Table 7.5. The results of iTimerM [13] are adopted as the baseline, and the differences and ratios are calculated as described in Table 7.2. The results achieve the same timing accuracy as iTimerM [13] which is of the best accuracy among the previous works. Therefore, it is validated that the insensitive

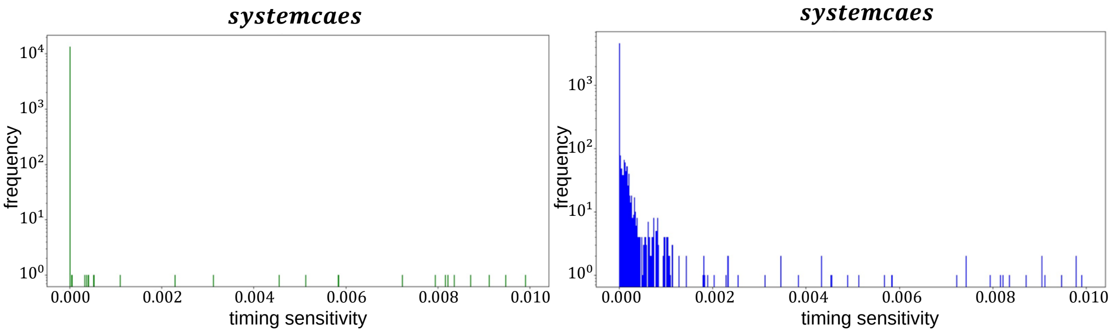


Figure 7.1: Separated TS distribution based on the insensitive pins filtering.

pins filtering does not degrade the resulting timing accuracy.

Lastly, to evaluate our framework's efficiency when we encounter new benchmarks under the same NLDM libraries, we only need to consider the GNN model inference runtime and the model generation runtime since our framework is available on general designs under the NLDM. The GNN model inference time usually takes less than 5 seconds for each design, which is much less than the model generation time listed in the above tables. Thus, our framework spends comparable or even better runtime than previous work for unseen test data under the NLDM. As for other timing delay models such as AOCV, POCV, and CCS, we need to further consider the training data generation time and the GNN model training time. The timing sensitivity training data generation takes several minutes to several hours, depending on the size of the design, and the GNN model training consumes about 30 minutes. However, since our framework could be directly applied to perform timing macro modeling no matter which timing model is chosen, users do not need to spend a great deal of time designing specific algorithms for different timing delay models and tuning a bunch of parameters. As a consequence, our framework still shows high applicability and efficiency on the timing macro modeling problem.

Table 7.2: Experimental results on TAU 2016 [5] and TAU 2017 [1] benchmarks with CPPR. For the avg. and the max error, we adopt the absolute value of difference between the result of macro model and the one of full netlist. If the results of macro model are more optimistic, the difference is further weighted by 2. For the model file size, we adopt the size of the library for late timing. Difference 1 and ratio 1 are compared with iTimerM [13]. Difference 2 and ratio 2 are compared with [12]. Difference = compared result - our result. Ratio = compared result / our result. Note that [12] is only evaluated on TAU 2016 benchmark in their work.

Design		Avg. Error (ps)	Max Error (ps)		Model File Size (MB)	Generation Runtime (s)	Generation Memory (MB)	Usage Runtime (s)	Usage Memory (MB)
mgc_edit_dist_iccad_eval	Ours	0.0000	0.007	Ours	56	11	1087	8	475
	iTimerM	0.0000	0.007	iTimerM	64	10	1043	8	550
	[12]	N.A.	0.158	[12]	79	15	5	4	5
vga_lcd_iccad_eval	Ours	0.0006	0.040	Ours	45	12	1204	6	383
	iTimerM	0.0006	0.040	iTimerM	50	13	1208	7	402
	[12]	N.A.	0.255	[12]	72	24	399	4	5
leon3mp_iccad_eval	Ours	0.0004	0.052	Ours	35	50	4908	5	324
	iTimerM	0.0004	0.052	iTimerM	45	58	4807	6	395
	[12]	N.A.	0.220	[12]	86	78	5	5	5
netcard_iccad_eval	Ours	0.0000	0.004	Ours	213	89	6609	29	1757
	iTimerM	0.0000	0.004	iTimerM	220	65	6513	29	1822
	[12]	N.A.	0.203	[12]	372	101	12616	23	4332
leon2_iccad_eval	Ours	0.0002	0.016	Ours	369	89	8298	64	3034
	iTimerM	0.0002	0.016	iTimerM	372	82	7865	61	3056
	[12]	N.A.	0.241	[12]	676	105	15299	38	5315
TAU 2016 Average	Difference 1	0.0000	0.000	Ratio 1	1.116	0.961	0.975	1.099	1.094
	Difference 2	N.A.	0.192	Ratio 2	1.809	1.448	0.818	0.738	0.851
mgc_edit_dist_iccad	Ours	0.0029	0.052	Ours	60	16	1054	8	514
	iTimerM	0.0003	0.052	iTimerM	66	12	1063	9	537
vga_lcd_iccad	Ours	0.0024	0.080	Ours	56	16	1455	7	474
	iTimerM	0.0023	0.080	iTimerM	58	15	1429	8	487
leon3mp_iccad	Ours	0.0031	0.046	Ours	37	68	5407	5	332
	iTimerM	0.0016	0.046	iTimerM	46	67	5281	6	406
netcard_iccad	Ours	0.0013	0.029	Ours	239	101	7814	35	1938
	iTimerM	0.0003	0.029	iTimerM	248	98	7545	33	1993
leon2_iccad	Ours	0.0027	0.095	Ours	438	125	8171	62	3613
	iTimerM	0.0013	0.095	iTimerM	440	109	8049	64	3625
TAU 2017 Average	Difference	-0.0013	0.000	Ratio	1.084	0.903	0.984	1.070	1.065

Table 7.3: Experimental results with and without CPPR-dedicated features.

Benchmark		Avg. Error	Max Error		Model File Size	Generation Runtime	Generation Memory	Usage Runtime	Usage Memory
TAU2016 (avg.)	Difference Before	0.0000	0.000	Ratio Before	1.064	1.055	0.959	1.133	1.048
	Difference After	0.0000	0.000	Ratio After	1.116	0.961	0.975	1.099	1.094
TAU2017 (avg.)	Difference Before	-0.0001	0.000	Ratio Before	1.060	0.828	0.994	1.115	1.037
	Difference After	-0.0013	0.000	Ratio After	1.084	0.903	0.984	1.070	1.065

Table 7.4: Experimental results on TAU 2017 benchmark without CPPR. For the avg. and the max error, we adopt the absolute value of difference between the result of macro model and the one of full netlist. If the results of macro model are more optimistic, the difference is further weighted by 2. For the model file size, we adopt the size of the library for late timing. Difference 1 and ratio 1 are compared with iTimerM [13]. Difference 2 and ratio 2 are compared with ATM [10]. Difference = compared result - our result. Ratio = compared result / our result. We additionally include the circuit *mgc_matrix_mult_iccad* to evaluate since ATM [10] also adopts it as one test case.

Design		Avg. Error (ps)	Max Error (ps)		Model File Size (MB)	Generation Runtime (s)	Generation Memory (MB)	Usage Runtime (s)	Usage Memory (MB)
mgc_edit_dist_iccad	Ours	0.0033	0.052	Ours	59	14	1069	9	563
	iTimerM	0.0007	0.052	iTimerM	65	13	1062	9	523
	ATM	0.0960	0.402	ATM	2	833	N.A.	0.36	N.A.
vga_lcd_iccad	Ours	0.0026	0.080	Ours	52	18	1457	7	442
	iTimerM	0.0023	0.080	iTimerM	55	17	1420	9	450
	ATM	0.0400	0.160	ATM	0.3	85	N.A.	0.06	N.A.
leon3mp_iccad	Ours	0.0033	0.046	Ours	31	78	5392	5	275
	iTimerM	0.0018	0.046	iTimerM	31	102	5257	4	286
	ATM	0.1070	0.460	ATM	0.6	740	N.A.	0.09	N.A.
netcard_iccad	Ours	0.0033	0.029	Ours	226	124	7804	32	1795
	iTimerM	0.0005	0.029	iTimerM	229	104	7539	33	1838
	ATM	0.0540	0.246	ATM	1.6	618	N.A.	0.27	N.A.
leon2_iccad	Ours	0.0027	0.095	Ours	408	193	8156	60	3378
	iTimerM	0.0013	0.095	iTimerM	410	152	7782	59	3390
	ATM	0.0400	0.240	ATM	2.4	1055	N.A.	0.34	N.A.
mgc_matrix_mult_iccad	Ours	0.0032	0.054	Ours	124	27	1106	18	924
	iTimerM	0.0020	0.054	iTimerM	171	29	1114	24	1098
	ATM	0.1300	0.450	ATM	12	629	N.A.	1.63	N.A.
Average	Difference 1	-0.0016	0.000	Ratio 1	1.093	0.980	0.978	1.085	1.033
	Difference 2	0.0748	0.267	Ratio 2	0.028	17.910	N.A.	0.029	N.A.

Table 7.5: Validation on insensitive pins filtering.

Benchmark	Avg. Error	Max Error	Model File Size
TAU2016	0.0000	0.000	1.040
TAU2017	0.0000	0.000	1.009

Chapter 8

Conclusions

In this thesis, we propose a generic timing macro modeling framework that is applicable on various timing analysis models and modes. In our framework, we first evaluate the timing criticality of each pin through a timing sensitivity metric, and generate the training data accordingly. Then, due to the analogy between the GNN and the timing macro modeling, GNN model can capture the timing properties effectively. Eventually, high-quality macro models could be generated. Experimental results based on TAU 2016 [5] and TAU 2017 [1] contests show our framework achieves extremely high timing accuracy while further improving the model size than the most accurate state-of-the-art work. Moreover, taking CPPR as an example, the generality and applicability of our framework is also validated empirically. We also demonstrate a generalized framework for MCMM. Future work includes timing analysis of MCMM timing macro models in a heterogeneous integration system.

Bibliography

[1] S. Chen, A. Khandelwal, X. Zhao, and X. Chen, "TAU 2017 timing contest on macro modeling," in *International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, 2017. [Online]. Available: https://sites.google.com/site/taucontest2017/

[2] A. J. Daga, L. Mize, S. Sripada, C. Wolff, and Q. Wu, "Automated timing model generation," in *39th Design Automation Conference (DAC)*, pp. 146–151, 2002.

[3] L. G. e Silva, L. M. Silveira, and J. R. Phillips, "Efficient computation of the worst-delay corner," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1617–1622, 2007.

[4] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *31st Conference on Neural Information Processing Systems (NIPS)*, pp. 1025–1035, 2017.

[5] J. Hu, S. Chen, X. Zhao, and X. Chen, "TAU 2016 timing contest on macro modeling," in *International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, 2016. [Online]. Available: https://sites.google.com/site/taucontest2016/

[6] T.-W. Huang and M. D. F. Wong, "OpenTimer: A high-performance timing

analysis tool," in *International Conference on Computer-Aided Design (ICCAD)*, pp. 895–902, 2015.

[7] T.-W. Huang, P.-C. Wu, and M. D. F. Wong, "Fast path-based timing analysis for CPPR," in *International Conference on Computer-Aided Design (ICCAD)*, pp. 596–599, 2014.

[8] A. B. Kahng, U. Mallappa, L. Saul, and S. Tong, ""Unobserved corner" prediction: Reducing timing analysis effort for faster design convergence in advanced-node design," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 168–173, 2019.

[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016. [Online]. Available: arXiv:1609.02907

[10] K.-M. Lai, T.-W. Huang, P.-Y. Lee, and T.-Y. Ho, "ATM: A high accuracy extracted timing model for hierarchical timing analysis," in *26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 278–283, 2021.

[11] T.-Y. Lai, T.-W. Huang, and M. D. F. Wong, "LibAbs: An efficient and accurate timing macro-modeling algorithm for large hierarchical designs," in *54th Design Automation Conference (DAC)*, pp. 65:1–65:6, 2017.

[12] T.-Y. Lai and M. D. F. Wong, "A highly compressed timing macro-modeling algorithm for hierarchical and incremental timing analysis," in *23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 166–171, 2018.

[13] P.-Y. Lee and I. H.-R. Jiang, "iTimerM: A compact and accurate timing macro model for efficient hierarchical timing analysis," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 4, pp. 48:1–48:21, 2018.

[14] P.-Y. Lee, I. H.-R. Jiang, C.-R. Li, W.-L. Chiu, and Y.-M. Yang, "iTimerC 2.0: Fast incremental timing and CPPR analysis," in *International Conference on Computer-Aided Design (ICCAD)*, pp. 890–894, 2015.

[15] W. Li, J. Xia, Y. Ma, J. Li, Y. Lin, and B. Yu, "Adaptive layout decomposition with graph embedding neural networks," in *57th Design Automation Conference (DAC)*, pp. 200:1–200:6, 2020.

[16] Y.-C. Lu, S. S. K. Pentapati, L. Zhu, K. Samadi, and S. K. Lim, "TP-GNN: A graph neural network framework for tier partitioning in monolithic 3D ICs," in *57th Design Automation Conference (DAC)*, pp. 64:1–64:6, 2020.

[17] Y. Ma, Z. He, W. Li, L. Zhang, and B. Yu, "Understanding graphs in EDA: From shallow to deep learning," in *International Symposium on Physical Design (ISPD)*, pp. 119–126, 2020.

[18] J.-J. Nian, S.-H. Tsai, and C.-Y. Huang, "A unified multi-corner multi-mode static timing analysis engine," in *15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 669–674, 2010.

[19] S. Onaissi and F. N. Najm, "A linear-time approach for static timing analysis covering all process corners," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1291–1304, 2008.

[20] S. Onaissi, F. Taraporevala, J. Liu, and F. Najm, "A fast approach for static timing analysis covering all PVT corners," in *48th Design Automation Conference (DAC)*, pp. 777–782, 2011.

[21] H. Ren, G. F. Kokai, W. J. Turner, and T.-S. Ku, "ParaGraph: Layout parasitics and device parameter prediction using graph neural networks," in *57th Design Automation Conference (DAC)*, pp. 124:1–124:6, 2020.

[22] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

Publication List

[1] Kevin Kai-Chun Chang, Chun-Yao Chiang, Pei-Yu Lee, and Iris Hui-Ru Jiang, "Timing Macro Modeling with Graph Neural Networks," in *Proceedings of 59th Design Automation Conference (DAC)*, San Francisco, CA, USA, July 2022 (*to appear*).